On Mobility of Software Processes*

Mingshu Li[1,2], Qiusong Yang[1,3], Jian Zhai[1,3], and Guowei Yang[1,3]

[1] Laboratory for Internet Software Technologies,
Institute of Software, Chinese Academy of Sciences, Beijing 100080, China
{mingshu, qiusong_yang, zhaijian, yangguowei}@itechs.iscas.ac.cn
[2] State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences, Beijing 100080, China
[3] Graduate University of Chinese Academy of Sciences, Beijing 100039, China

Abstract. In this paper, the *mobility of software processes* is proposed as a novel concept. It is defined as the structural change in a software process resulting from interactions among linked process elements. The concept addresses the essential change in a software process which brings a high variability and unpredictability to process performance. Three categories of the *mobility* that lead to the structural change are identified and expounded upon. A reference model for describing the concept is put forward based on the polyadic π-calculus. With the *mobility of software processes*, it is possible to design a new PCSEE and associated PML with increased flexibilities.

1 Introduction

The research on software processes is to enable people to produce high quality software systems and evolve them in an economic and timesaving fashion. The main stream of effort has been on concepts definition, languages and complete process-centered software engineering environments (PCSEEs). The process "culture" is widely recognized and adopted. However, existing PCSEEs fail today in satisfying the market's evolution and the demand that may be summarized by [1]: the support of long lived and widely distributed, heterogeneous, evolving and flexible processes. The notion of flexible process support costs an extra price. The more flexible and adaptable PCSEEs are (in other words, the wider the variety of processes which can be supported), the weaker is the support for a concrete process [2].

A software process is still human intensive and almost impossible to be improved by a product view like in classic manufacturing. It is a set of activities or operations that needs to always change for a variety of reasons. In order to improve process support technology, we have to answer the following questions:

- What is the essential change in software processes?
- Based on the essential change, is it possible to define a novel concept?

* Supported by the National Natural Science Foundation of China under grant No. 60273026, 60473060, 60573082 and the Hi-Tech Research and Development Program (863 Program) of China under grant No. 2004AA112080, 2005AA113140.

Q. Wang et al. (Eds.): SPW/ProSim 2006, LNCS 3966, pp. 105–114, 2006.

– Is there a reference model that can be devised to describe the concept?
– Can this model be used to design a PCSEE/PML (process modelling language) to support the essential change in software processes?

2 Mobility of Software Processes

It is widely accepted that the quality of software is related to not only the product, but the organization and the production process. According to Webster's dictionary, a *process* is "a series of operations performed in the making or treatment of a product" or "a series of actions, changes, or functions bringing about a result". Various definitions of the software process have been put forward from different angles:

– A software process can be defined as a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals) [3].
– A set of partially ordered process steps, with sets of related artifacts, humans and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables [4].
– A sequence of tasks, actions, or activities, including the transition criteria for progressing from one to the next, that brings about a result [5].

In this paper, a *software process* is defined as a set of process elements, links and interactions. The execution of a software process constitutes a trace of interactions among linked elements. Process elements are the basic entities of a software process, including activities, humans, artifacts, computerized resources, etc. A link is the abstraction of a certain type of relationship or a communication channel between two process elements. Each element can interrelate with other ones. The performance of a software process is a trace of interactions among interrelated elements. The ordering of those interactions is regulated by some constraints, methods, or practices. In addition, an interaction is carried out along a link for the purpose of sending a piece of data or some information for control between process elements. The control flow and the information flow of a software process are described through specifying its connecting structures and interactions types.

2.1 Conception of Software Process Mobility

The structure of a software process states the way in which the process elements are connected with each other through links, and the set of possible interactions that can be carried out among linked process elements. In fact, it may change during process performance as a result of interactions among process elements. It is possible that new process elements are added to a software process, existing ones deleted, and one process element replaced by another. For example, a new human agent (a process element) may be added for the enrollment of a new

staff. On the other hand, a new link can be setup between two process elements who are unknown to each other in advance and two linked process elements may be disconnected. For example, a test engineer's affiliation with the test manager (a link) is shifted to a program manager when he or she is reassigned to the team for implementation. Furthermore, the set of possible interactions of a software process are altered correspondingly when the process elements or the links change. It is the essential change in a software process that its structure is altered during performance. It brings a high variability and unpredictability to software processes and may cause inconsistencies between process enactment and process performance.

Concerning the essential change in software processes, a novel conception, the *mobility of software processes* is proposed. According to Webster's dictionary, the word *mobility* means the "the quality of moving freely". The *mobility*'s synonyms within context are: changeableness, sensibility (and commonalty, motion). Thus, the *mobility of software processes* is defined as the structural change in a software process, resulting from interactions among process elements through links. As the logical relations among process elements remain immobile, the physical movement of a process element is not treated as the mobility of software processes. In addition, the situation that the internal state of a process element is updated or one process element seizes control from another is also not taken into account.

According to the definition of the mobility of software processes, it is the interactions that result in the structural change in a software process. On the other hand, the structure of a software process determines what interactions can be carried out along links connecting process elements. In the mobility of software processes, a software process is surveyed from the negativity of self-denial point of view and interactions among linked process elements constitute the momentum of process performance. Hence, based on the interactions among linked process elements, it is possible to describe the mobility of software processes in a modest but profound way.

2.2 Category of Software Process Mobility

Two basic categories of the mobility and a combination of them can be identified according to the mobile unit during an interaction:

- Element Mobility: A process element is mobile without links.
- Link Mobility: A link is mobile without process elements.
- Combined Mobility: Both process elements and the links among them are mobile.

Element Mobility. A process element is the mobile unit during an interaction. The received element will be connected with other ones existing in the new context. In addition, the creation of a new process element can also be expressed in the element mobility, in which a new element is added to the environment along the link between the element's producer and the environment. The behavior and the internal structure of the receiver can be dynamically updated.

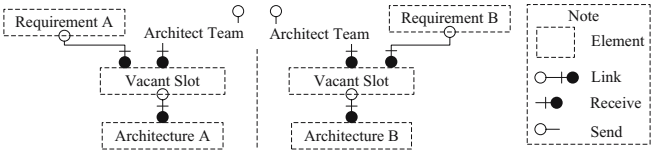

Fig. 1. An Example of the Element Mobility

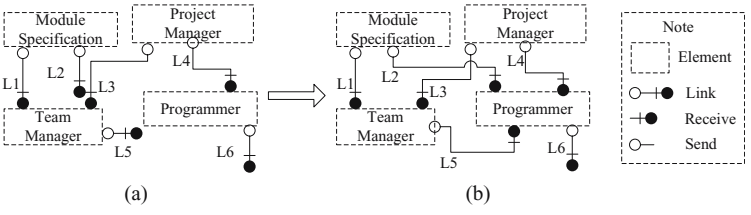

Fig. 2. An Example of the Link Mobility

In Fig. 1, there are more than one project that are simultaneously developed within an organization. But the architecture of each project is developed by the same `Architect Team`, responsible for devising an elegant architecture according to the given `Requirement`. In general, there is only one project that is scheduled for the `Architect Team`, which becomes mobile among those projects. Each project receives the `Architect Team` from a link and collaborates with it to produce an `Architecture`.

Link Mobility. It is a link to be mobile during an interaction. One process element sends a link, which is already connected to another element, to the third one. Thus, a new relationship can be set up between the latter two elements, who are unknown to each other in advance. The link mobility sticks to the fact that some process elements are dominated by some other ones or a meta-process which has the necessary knowledge to maintain a whole software process. It reflects the intrinsic dynamics in the control flow and information flow of a software process.

Fig. 2 denotes a demonstration on how the incremental definition of a software process is described in the link mobility. As shown in Fig. 2(a), a project manager assigns a programmer to a specific team and the team manager will have the programmer implementing a module according to the module's specification. As it is in a highly dynamic environment, neither the team manager nor the programmer is aware of the existence of the other before the performance of the software process. In Fig. 2(b), the project manager sends the link `L5` to the programmer. The programmer establishes a new connection with the team manager through the link. The team manager sends the link `L2` to the programmer and the programmer retrieves the module specification through the received link. Lastly, the programmer outputs the produced source code of the module through the link `L6`.

Combined Mobility. A fragment of a software process, including elements and links, is mobile. The combined mobility shows that a part of development

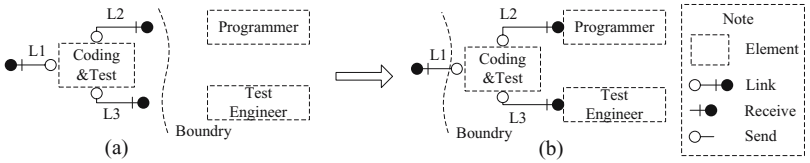

Fig. 3. An Example of the Combined Mobility

is delegated to a partner or a development team. The receiver of the fragment is responsible for establishing appropriate connecting structure for the received fragment. A fragment

In Fig. 3, the `Coding&Test` fragment is migrated along a link across the boundary. Connections can be constructed among the migrated fragment with those process elements on the other side of the boundary. A potential usage for the combined mobility is to present a process along with a software outsourcing contract. Thus, not only the milestones but also the development process adopted by the contractor can be fully specified. This provides a solution to problems caused by ineffective communication between contractors.

3 Formal Semantics

This section presents formal definitions of the mobility of software processes and three categories of the mobility. In addition, the polyadic π-calculus [6][7] proves to be a perfect candidate for constructing a new PCSEE supporting the novel concept.

Let a software process is represented as $SP = S\langle \mathcal{E}, \mathcal{L}, \mathcal{I} \rangle$, where, $\mathcal{E}$, $\mathcal{L}$, and $\mathcal{I}$ respectively represent the set of process elements, links and interactions of the software process, and S denotes the process's structure. In addition, $i\langle l \rangle$ represents an interaction along the link l between two linked process elements and m denotes the mobile unit during the interaction. Then, a formal definition of the mobility of software processes can be given as:

Definition 1 (Mobility of Software Processes). *The mobility of software processes is the structural change in a software process resulting from an interaction:*

$$S\langle \mathcal{E}, \mathcal{L}, \mathcal{I} \rangle \frac{i\langle l \rangle}{m} S'\langle \mathcal{E}', \mathcal{L}', \mathcal{I}' \rangle$$

where, $S \neq S'$ (S and S' are the structure of the software process before and after the interaction respectively).

The mobility of software processes is classified into three categories, i.e. *Element Mobility*, *Link Mobility*, *Combined Mobility*, according to the mobile unit m during an interaction. Let $RU(l)$ and $SU(l)$ denote a process element which respectively receives and sends a mobile unit from the link l. We then have three similar definitions but significant differences of mobile unit:

Definition 2 (Element Mobility). *Let $n \geq 1$ and $e \in \mathcal{E}$ denotes a mobile process element. The element mobility constitutes a series of interactions:*

$$\frac{i\langle l_1\rangle}{e}, \frac{i\langle l_2\rangle}{e}, \cdots \frac{i\langle l_n\rangle}{e}$$

where, $\forall i(i \geq 1 \wedge i \leq n-1)$, $RU(l_i) = SU(l_{i+1})$. Then, $RU(l_n)$ instantiates the mobile process element e and sets up an appropriate connecting structure for it.

Definition 3 (Link Mobility). *Let $n \geq 1$ and $l \in \mathcal{L}$ denotes a mobile link. The link mobility constitutes a series of interactions:*

$$\frac{i\langle l_1\rangle}{l}, \frac{i\langle l_2\rangle}{l}, \cdots \frac{i\langle l_n\rangle}{l}$$

where, $\forall i(i \geq 1 \wedge i \leq n-1)$, $RU(l_i) = SU(l_{i+1})$. Then, a new link is set up between $RU(l_n)$ and the process element to which the link l is initially connected.

Definition 4 (Combined Mobility). *Let $n \geq 1$ and $l_{\&}e$ denotes a set of linked process elements. The combined mobility constitutes a series of interactions:*

$$\frac{i\langle l_1\rangle}{l_{\&}e}, \frac{i\langle l_2\rangle}{l_{\&}e}, \cdots \frac{i\langle l_n\rangle}{l_{\&}e}$$

where, $\forall i(i \geq 1 \wedge i \leq n-1)$, $RU(l_i) = SU(l_{i+1})$. Then, $RU(l_n)$ sets up an appropriate connecting structure for $l_{\&}e$ and existing links in $l_{\&}e$ are still there.

In addition, the mobility of software processes and three categories of the mobility can be modelled in the polyadic π-calculus. With the formalism, it is fairly straightforward for working out a new PCSEE and associated PML supporting the concept. Based on the polyadic π-calculus, an process element is represented as a process in the untyped polyadic π-calculus (called π-process in this paper). A link between two process elements is modelled as a channel connecting the two corresponding π-processes. Interactions among process elements will be transformed into events of concurrently combined π-processes. With the operator of *abstraction*, a process element can be represented as:

$$\texttt{Element} \stackrel{def}{=} (\widetilde{ch}).\ (\nu\widetilde{g}, \widetilde{s})\ (U_p\langle\widetilde{g}, \widetilde{s}, \widetilde{0}\rangle \mid M_p\lfloor\langle\widetilde{g}, \widetilde{s}, \widetilde{ch}\rangle\rfloor) \tag{1}$$

$$U_p \stackrel{def}{=} (\widetilde{g}, \widetilde{s}, \widetilde{v}).\ (V\lfloor\langle g_1, s_1, v_1\rangle\rfloor | \cdots | V\lfloor\langle g_m, s_m, v_m\rangle\rfloor) \tag{2}$$

$$V \stackrel{\triangle}{=} (g, s, u).\ (g(r).\ \bar{r}u.\ V\lfloor\langle g, s, u\rangle\rfloor) + s(v).\ V\lfloor\langle g, s, v\rangle\rfloor \tag{3}$$

$$M_p \stackrel{\triangle}{=} (\widetilde{g}, \widetilde{s}, \widetilde{ch}).\ (Action\langle\widetilde{g}, \widetilde{s}, \widetilde{ch}\rangle\ .\ M_p\lfloor\langle\widetilde{g}, \widetilde{s}, \widetilde{ch}\rangle\rfloor) \tag{4}$$

In (1), $\widetilde{ch}$ represents links connected to a process element. We assume that an element has a state and presents a certain type of behavior pattern (action).The two processes, U_p and M_p, represent the state and the action respectively. They share the channels $\widetilde{g}$ and $\widetilde{s}$. Thus in the body of the action, state variables can be respectively *get* or *set* through $\widetilde{g}$ and $\widetilde{s}$. The access to a variable is modelled

by the process (3). In (2), processes for each variable are concurrently combined together to represent the private store of an element. The action of an element has the form $(\widetilde{g}, \widetilde{s}, \widetilde{ch}).P$.

A set of linked elements is also modelled as a π-process through the *application* notation. For example, a new linked element can be constructed from previously defined ones:

$$\texttt{ElementA} \stackrel{def}{=} (\langle in, out\rangle)\texttt{ElementABody}$$
$$\texttt{ElementB} \stackrel{def}{=} (\langle in, out\rangle)\texttt{ElementBBody}$$
$$\texttt{ElementC} \stackrel{def}{=} (\langle in, out\rangle)(\nu ch)(\texttt{ElementA}\langle in, ch\rangle|\texttt{ElementB}\langle ch, out\rangle)$$

As for the link mobility, it can be modelled by the name-passing of π-calculus. For example, the `Programmer` in Fig. 2 can be defined as:

$$Programmer \;=\; (\langle l_7, l_9\rangle)(l_7(l_8).\; l_8(l_5).\; l_5(content).\; coding.\; \overline{l_9}code) \qquad (5)$$

where, the state of a *Programmer* is not taken into account.

For the reason that an process element and a set of linked elements are both modelled as a π-process, the element mobility and the combined mobility are represented by the process-passing of high order π-calculus. For example, the equation

$$Fig3(b) \;=\; (\langle l_1, l_2, l_3\rangle)(l_0(codingest).codingtest\langle l_1, l_2, l_3\rangle) \qquad (6)$$

depicts Fig. 3(b) that the migrated `Coding&Test` is received and invoked. A high order π-calculus can be faithfully compiled down to the polyadic π-calculus (a first-order calculi) according to [7].

4 Implementation in SoftPM

In this section, an example is presented to show how a process for testing is expressed in SoftPM based on the mobility of software processes. SoftPM [8] is a toolkit for software process management and has been widely adopted in Chinese software organizations. The development teams of a customer are distributed across the whole city and there is one department, named *Quality Assurance Department*, who is responsible for testing all the projects within the organization. As an independent department assuming sole responsibility for its profits and losses, it is necessary to manage all testing activities by creating a new project in SoftPM.

However, it is difficult to predict the number of projects that are being tested in advance and the schedule of a test is heavily depends on the progress of the corresponding project. Thus, those process elements, including the project manager, developers, test cases, and source code, have to be dynamically allocated or deleted. In addition, to ensure that a bug is timely fixed, the tester conducts tests on the source code against given test cases and sends any identified bug to

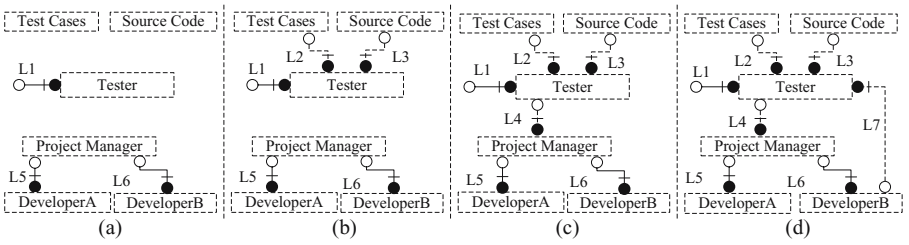

Fig. 4. A Process for Testing in the Link Mobility

the manager of the project being tested. Then, the bug is delegated to a developer according to its type. The developer fixes the bug and the result is fed back to the tester. The project manager and developers that a tester should communicate with are not prescribed and the relationships among them are difficult to be defined in a prescriptive manner.

The mobility of software processes surveys those problems from the angle that the structure of a software process may change as a result of interactions among process elements. In Fig. 4, the process for test is expressed in the link mobility. The process commences with Fig. 4(a), in which the `Tester` has not been assigned to a testing activity and is ready for accepting new tasks from the link `L1`. Then, the links to the `Test Cases` and the `Source Code` of the project to be tested is sent to the `Tester` along `L1`. New links, `L2` and `L3`, are set up as shown in Fig. 4(b). After the two previous interactions, a link to the project manager is also sent to the `Tester` along the link `L1`. As a result, the link `L4` between the `Tester` and the `Project Manager` is created in Fig. 4(c). Through the `Project Manager`, the link of the `Tester` is sent to the `DeveloperB` and the link `L7` is built up. In Fig. 4(d), a structure for communication among those process elements is appropriately configured. As you can see, the high variability and unpredictability of process performance is effectively addressed in the mobility of software processes.

5 Conclusion

In this paper, a software process is abstracted as a set of process elements, links and interactions. The execution of a software process constitutes a series of interactions among linked process elements. The intrinsically complex interrelationships among those entities involved during software development are described by the structure of a software process. The structural change imposed by interactions among linked process elements is considered as the essential change in a software process and brings a high variability and unpredictability to process performance. The mobility of software processes is presented as a novel concept to address the structural change. It reflects the fact that a software process is not static and it is changed through the negativity of self-denial driven by interactions. According the mobile unit during an interaction, three categories of the mobility are identified.

The mobility of software processes has a fundamental difference with the evolution of software processes [9][10]. The latter mainly focuses on solutions used for guiding how to apply an outer change request to a process or a model. The concept of evolution generally assumes that the structure of a software process is static, while the mobility states what a software process should be and exploits the momentum for structural changes. It is also different from the mobile software process described in [2] or [11][12], in which process parts, tools, participants tend to change their site allocation during the process or a process fragment is distributed in different workspaces. The dynamic ordering that the ordering of activities can be dynamically built and modified is an identified requirement for assessing a list of PCSEEs in [1]. However, the phrase is intended for expressing the non-determinism in the building constructs of PMLs.

As a novel concept, some aspects of the mobility of software processes can be exploited further:

- The mobility of software processes focuses on the structural change of a software process. The evolution of software processes can be taken as any change which takes place in software processes. In this way, the mobility of software processes can be thought of as a special type of evolution. However, as a novel concept, its correspondence with the evolution should be further clarified.
- It is necessary to exploit strategies and policies for managing the mobility of software processes. Inconsistencies between the process performance and the process enactment can be minimized with appropriate control criteria and policies for the mobility of software processes. A modelling approach based on the polyadic π-calculus can be further studied to support the novel concept. In particular, new techniques for analyzing software processes can be put forward based on the formalism. In addition, some other formalisms can also be examined to support the mobility of software processes.
- The mobility of software processes is classified into three categories according to the mobile unit during an interaction. It is possible that a new standard is adopted to produce different categories that define the extension of the concept.
- Moreover, a new PCSEE and associated PML can be developed based on the novel concept.

References

1. Arbaoui, S., Derniame, J.C., Oquendo, F., Verjus, H.: A comparative review of Process-Centered Software Engineering Environments. Annal of Software Engineering **14**(1-4) (2002) 311–340
2. Gruhn, V.: Process-centered software engineering environments, a brief history and future challenges. Annals of Software Engineering **14**(1-4) (2002) 363–382
3. Paulk, M.C., Curtis, B., Chrissis, M.B., Weber, C.V.: Capability maturity model for software, version 1.1. Technical Report CMU/SEI-93-TR-024, SEI, CMU (1993)
4. Lonchamp, J.: A structured conceptual and terminological framework for software process engineering. In: ICSP. (1993) 41–53

5. IEEE Std. 1220-1998: IEEE standard for application and management of the systems engineering process (1998)
6. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes – part I and II. Journal of Information and Computation **100** (1992) 1–77
7. Sangiorgi, D., Walker, D.: The π-calculus: a Theory of Mobile Processes. Cambridge University Press (2001)
8. Wang, Q., Li, M.: Software process management: Practices in China. In Li, M., Boehm, B.W., Osterweil, L.J., eds.: ISPW. Volume 3840 of LNCS., Springer (2005) 317–331
9. Conradi, R., Fernström, C., Fugetta, A.: Concepts for evolving software processes. In A. Finkelstein, J. Kramer, B.N., ed.: Software Process Modelling and Technology, John Wiley and Sons (1994) 9–31
10. Bandinelli, S., Nitto, E.D., Fuggetta, A.: Policies and mechanisms to support process evolution in PSEEs. In: ICSP. (1994) 9–20
11. Ben-Shaul, I.Z., Kaiser, G.E.: A paradigm for decentralized process modeling and its realization in the Oz environment. In: Proceedings of the Sixteenth International Conference on Software Engineering, IEEE Computer Society Press (1994) 179–188
12. Wang, A.I.: Support for mobile software processes in CAGIS. In Conradi, R., ed.: EWSPT. Volume 1780 of LNCS., Springer-Verlag (2000) 115–130