# Normed Dinosaurs in a Creative Chaos? Software Development as Controlled Learning Process

**Claus Lewerentz, Heinrich Rust. Reachable at: BTU, Lehrstuhl für Software-Systemtechnik, Postfach 10 13 44, D-03013 Cottbus, Germany; Tel. +49(355)69-3803, Fax.:-3810;** (cl|rust)@informatik.tu-cottbus.de

## 1 A Problem of Management Centered Software Development Processes

Since the end of the 1960s, the software engineering discipline strives to develop models for work processes for developing software. The 'software crisis', i.e. the phenomenon that the cost for hardware was falling while the cost for software was rising, was one of the reasons to investigate into the work processes which are adequate for developing software, since the chaotic and often unsystematic work process in software development was seen as one of the main reasons for this crisis.

As the most important goal in many situations, project success in the form of customer satisfaction has been identified. In practical projects, two causes of customer dissatisfaction are paramount: Defects in the product, and cost as well as time overruns in the process. In this context, a base for planning was searched for as well as some means which allow to establish transparency with respect to the reached level of progress in the process and of correctness of the implemented part of the product. This transparency is to be used to recognize problems with respect to process or product as early as possible, in order to be able to find countermeasures. Since the developed software product is the result of the software development process, and thus measures for better product quality are are to be integrated into a software development process, it is the software process where enhancements should start.

In the course of the three decades since the end of the sixties, a plethora of models and meta models for conceptualizing and fixing software development processes by describing them have been developed. Examples are the cleanroom approach, the capability maturity model, SPICE, ISO 9000 and the V model. Each of them has a descriptive component, i.e. each presents a conceptual model to describe software development processes. Some have an evaluative component, i.e. they are used for assessing software development processes. Some of them also have a prescriptive component, i.e. they prescribe some necessary content of adequate processes, or they make at least recommendations about the next steps to be taken to enhance an existing software development process.

Typically, it is the middle management which has the responsibility to implement some process enhancement. This primary assignment of responsibility for the software development process to the management is one reason for the enhancement activities often being ineffective. The problem lies in the acceptance of the processes by the software developers. Often, the latter feel the described processes to be not very helpful. They often perceive prescribed processes and the clerical work connected to them as an almost useless and nasty overhead with no benefit. The fixed process definition is not understood as an instrument which helps to find the right direction for the work process, but rather as some kind of chains which one tries to get rid of. Additionally, developers see written process not so much as one of their work means, but rather as the property of a methods

group, a quality assurance group or a certification group which is imposed on the developers. This creates the danger that developers do not accept written processes as a means of their own work, and that they only work by it as long as they are exposed to pressure.

The fixed character of rigidly normed development processes makes them inflexible. This justifies the dinosaur metaphor in our title. What has to change for them to be better adaptable to changing environments, and to be better accepted by developers?

## 2 A Solution: Developer Centered Software Development Processes

The danger just described can not be countered as long as developers see software processes primarily as a control instrument of the management. In the last years, other types of processes have became more important which focus on the level of the developers. Instead of emphasizing the transparency of the project for the management, these processes stress the increase of control which the developer might enjoy by the use of some process. A well known example is the Personal Software Process (PSP) by Watts S. Humphrey [5]. In this process, the work situation of the single developer is in the focus of attention. Another example is the Extreme Programming approach [1], which focuses on developers performing in a group and in tight cooperation with representants of the customer.

The idea is to put control over the software process into the hands of the developers; their managers are more consultants which find solutions for problems of work organization than than bosses which tell them just how they have to work. This means that developers get the responsibility to organize their work. This fits a principle known from occupational and organizational psychology: To fix reponsibility and competences for some task on the most basic level of an organization which is possible, in order both to recognize problems and to take effective countermeasures quickly. In software development, the members of development teams are the first who might recognize problems, and this is true both for project and product problems.

This is especially relevant since software development often takes place in a quickly changing environment. Different customers have different requirements, and even the same customer can have different priorities in different projects, or even in different project phases; innovation is fast with respect to operating systems, programming language versions, libraries, and compilers; developer groups are formed and dissolve. A software development process which works nicely in one case might stop working in the next environment. The developers are the first who might notice that a particular feature of a software development process does not work under the current circumstances.

Assigning this responsibility to a developer, without him or her having the capability to deal with the requirements, sometimes leads to the person acquiring the necessary competences, but more often it leads to failing to cope with the assigned responsibility, to frustration on the side of the developer, and finally to a desaster in the project. Before developers are made responsible for adjusting their work process to the requirements of a changing environment, they have to be made competent to do this. This competence has two facets. First, they need specific practical *knowledge*. They must learn how to organize their work process, how to recognize problems early, and how to reorganize their work process when some problem has been recognized. The second facet of this competence

is the *freedom of choice* about the decisions to take, to use resources for analyzing one's situation and to take countermeasures if a problem has been identified. It is the responsibility of the management to provide the developers both the necessary knowledge and the necessary freedom.

## 3 Software Development Organization as Controlled Learning Process

### 3.1 The concept of a controlled learning process

Software development process organization should be laid into the hands of the developers to be accepted by them. The general framework for conceptualizing software development processes with this concequence is to see them as controlled learning processes:

- It is conceptualized as a *learning* process, because the right way to perform a software development project often can not be fixed beforehand. The changing environment conditions, with respect to customers, task, tools, individual developer personalities and their behaviour in group work, make it impossible to know a priori which methods are the most appropriate in a newly started project. Thus, we assume that nobody in the software development organization knows which is the optimal process to use in a given case, i.e. neither management, nor a software process or QA group, nor the developers themselves.

  An area which might illustrate this nicely is the question how one should deal with change requests from the customer. If they are rare, sensible, and reliable, in the sense that requirements are not changed back and forth, it might not be necessary to follow a formally fixed written protocol. On the other hand, if the customer develops the tendency to require changes in an initial specification only to change these changes back again, it makes sense to follow some formal way in dealing with change requests, if only to document the reasons for some initially unexpected amount of work. The way how one has to deal with change requests adequately might not be known until after some time into the project; thus, the project team has to be able to learn so as to adjust their way of dealing with change requests.

  It is one of the main features of our approach to convince all relevant parties at the start of a new project that the optimal process is not yet known, i.e. that there is some learning need, and that it is thus sensible to organize the software development process as a learning process.

- The learning process should be *controlled*. This means that valid and objective data about critical process properties are collected systematically, that these data are reflected regularly, that the changes of the process, i.e. the learning, are based on such data (if possible), and that after some time, process changes are evaluated, if possible on the basis of such data. In particular, it means that

  – the current development process should be fixed in some accessible description,

- the adequacy of the current software development process to the current conditions should be assessed by some software process measures,
- that the reasons for process changes should be written down, if possible (but not necessarily) with reference to some measurements which make the change plausible,
- and the effects of the process change should be checked after some time, if possible via the comparison of some measurements values taken before the change and some measurement values taken after the change.

Thus, to conceptualize a software development process as a controlled learning process does *not* mean that developers may just do as they like. Even if management should not directly control the work processes used, they have to check whether developers know what they are doing, whether they strive to learn, and whether they control the effects of their interventions.

Software development as a controlled learning process is similar to research. It is about finding the proper process for the situation at hand by way of stepwise exploration, where process elements are changed in an experimental fashion in order to see which combination of process elements is best for the situation at hand. Thus, it is important to explore, but this activity should be controlled, i.e. experiences should be collected, interpreted, and consequences should be drawn from them.

Both the PSP and the Extreme Programming approach contain elements which help the developers to assess their own work processes. Performance measures, e.g., are built into both the PSP and the Extreme Programming approach, thus there is ample opportunity to investigate into the appropriateness of some specific work process under the current conditions. A software development process which is seen as controlled learning emphasizes the use of performance measures by the developers themselves, not by their managers.

### 3.2 Creating an organizational culture for learning

In order to install the conceptualization of software development as controlled learning, a change in the typical organizational culture is necessary. From the typical situation in which participants emphasize how much they are in control of a situation (even if they are not), a way has to be found to create a situation in which learning needs are acknowledged and it is generally accepted to spend ressources for learning. The elements of this culture can partly be described individually, and partly by reference to features of the organization.

**Some organizational features.** Much has been written on learning organizations (cf. [2] for an approach). We give only some small examples for a primarily organizational feature of a learning culture: Management has to accept that learning needs ressources, that reflection on ones experiences takes time; it has to show by its question and interest in some project that it *expects* the developers to learn; and it has to present a role example to the developers, by showing that learning needs are also acknowledged by managers and how they deal with them.

**An individual feature.** We used the approach of conceptualizing a software development project as a controlled learning process in several student projects in the software

engineering education at the BTU Cottbus. Several groups of about four students had to do a software development project which was scheduled to take the winter term.

In the first set of such projects, we quickly learned that the main problem was to make the students understand their own learning needs. This is a feature of learning processes which stresses some property of an individual; it is less a feature of the organization. But it is a first step on the way to creating an organizational culture which is friendly to learning.

It proved sensible to let the students experience their learning needs not just theoretically. For example, it did not suffice to ask them how long they estimated it would take them to complete some software development task, and then to demonstrate to them how different the numbers were, and how few data they had on which to base their estimation, and that their numbers were thus highly speculative; while all this was helpful, it was not sufficient. The most motivating idea proved to be the following: We took three weeks of the projects total time and let the students define a partial project which they should finish after the first three weeks. We required that the partial project had to go through all the steps of their chosen process, which should include at least specification, design, implementation, test and a documentation phase, and that they should select a partial project from the total project small enough so that they could complete all tasks in three weeks calendar time, and that each student should write down the times needed during this partial project for different types of activities. After three weeks, we let each of the groups present their results, and we emphasized that they had especially to deal with process problems they encountered. All the groups missed their target date. In the presentation, the groups were not only forced to explain their problem with completing the partial project in time, but they also saw that schedule problems were universal. - On the basis of these experiences, it was not too difficult to convince most students that they had some learning needs, and to help them make more specific what they wanted to learn in the course.

In order to make learning needs explicit and to make developers sensitive with respect to them, the first step in the approach of the psychologist Kurt Lewin to practical learning seems to be appropriate. Lewin proposed to cut the learning process of behavioural change into the three phases unfreezing, change, and re-freezing. “Unfreezing” means to break up existing conceptions in order to open the person for a behavioural change. The “change” phase contains the learning of another kind of behaviour, and in the “re-freeze” phase, the changed behavioural pattern is fixed. Our initial 3-week phase corresponds to the unfreezing phase in a Lewinian behavioural change process, by changing the self-conception of the participants from a person not needing to learn much to a software developer with at least some quite specific learning needs.

### 3.3 Important features of a learning friendly software development process organization

We could identify the following important features of a software development process seen as a controlled learning process:

- The development process should place many activities into *social* situations.

  The social situation will help learning by making otherwise implicit assumptions

and violations of rules explicit. For example, when experiences are interpreted, a social situation will help to make different conceptions explicit and to raise different relevant points, which leads to more differentiated learning opportunities. During coding, a social situation (like pair programming) will not allow rule violations to happen without being at least mentioned. During design, a social situation can help to discuss more alternatives or analyze them in more respects (cf. [6]).

- The development process should progress *cyclically*.

  Our concept of learning is based on experiences made in practice, not so much on deep thoughts or on reading about experiences made by other persons. To learn from practical experiences, situations have to repeat, such that what has been learned in some instance can be applied in some later instance. Because of this, it is important that similar situations occur repeatedly.

  Thus, a pure waterfall model is the wrong approach to organize software development projects for learning. Boehm's spiral model [3] is an approach to organizing software development processes which stresses this feature.

  Both the PSP and the Extreme Programming approach include a cyclical process model. Humphrey proposes to develop software in increments. The total functionality is split up into parts which are designed, implemented, and tested as units. In the Extreme Programming approach, the time frames for increments are fixed: Each increment takes about three calendar weeks. This slotted organization helps in the synchronization of the micro-teams the whole development team consists of.

- A *base process* hast to be established.

  The base process is used in the first iterations of a development process. As such, it has several functions:

  - It is the collection of the experiences made in the organization so far. Thus, it is some kind of an organizational memory. It can be used for learning/teaching purposes.
  - The process elements collected in the base process provide a common vocabulary for developers to talk about their processes.
  - The base process provides a horizon of standard process elements. They provide a minimal set of methods which emphasize some features of the development process.

  The base process should be rather generic and, at the same time, minimal.

  The base process is the starting point of a series of successive changes which are meant to lead to an ever increasing fitness of the work process to the current situation. Only small changes are possible in each process change experiment – otherwise the changed process would be too difficult to follow. Since each process change experiment takes some time, it depends heavily on the adequacy of the starting point, i.e. of the base process, how good the team performs on average during the whole process.

The definition of this base process is similar to the customary tasks of process definition and standards groups in software developing organizations. The main difference to standard ways of using process descriptions by developers is that in our approach, it is the responsibility of the developers to decide about the appropriateness of process changes, not the responsibility of a process group.

- *Measurement data* have to be collected about the process.

  Measurements are a method to objectify experience. Recently, they receive much attention in computer science (cf. [4]). Process measures are used for describing features of the work process itself. The most interesting process measures for controlling the learning process are the time needed to complete some task or the defects found in some work phase. These are also used by Humphrey in his PSP.

  Product measures are used to characterize the results of the work process. These, too, are important for the learning process. The most often used software product measure is the number of code lines. There are many variants with respect how to count empty lines, comment lines, lines with only scope delimiters or lines with several statements on them. No matter what line counting standard is established, the most important feature is that it should correlate linearly with the time used for developing the software.

  Both process and product measures are helpful for learning. They provide a more easily handled abstraction of what has happened during a process, and they provide an abstraction of what has been produced. In addition, composite measures like productivity for some task can be defined by dividing the time used for the task (a process measure) by some measure describing the size of the work product (a product measure).

- The development process must include *reflexion phases*.

  Data collection alone will not lead to controlled learning. The collected data have to be interpreted, and based on the interpretation, consequences have to be taken. It is the interpretation which is supported by the reflexion phases. Results of reflexion phases should be documented in the form of process change recommendations.

- A collection of problem descriptions and possible solutions should be collected in a *catalogue*.

  The catalogue of problem descriptions and solutions best has the form of a collection of descriptions of concrete cases. It complements the base process described earlier. While the base process provides a platform from which any learning can start and should be generic and minimal, this catalogue contains project specific cases, in the form of problem descriptions and solutions found.

  This catalogue also forms part of the organizational memory. It has to be maintained by a special instance in the organization which collects relevant organization, but it may and should be used by everyone. The case description approach should make it an easier reading than if a pseudo-scientific format was chosen for it.

- *Learning consultants* help the development groups to organize their project.

  Organizing a software development project as a controlled learning process is a difficult task. Thus, developers need assistance, at least when they start using the new approach to organizing a project.

  The main task of these consultants is to make the developers sensitive for learning needs, and to point to solutions for the problems which have been developed in other development groups, in order to make the developers try them out.

- The approach is *teachable* in a school environment, and it is an appropriately fundamental subject.

  The approach to conceptualize the software development process as a controlled learning process stresses less concretely given current methods and tools – even if they are also a component of the approach. But these are themselves merely tools to learn a quite fundamental and general approach to organizing one's work: To use some orderly process, collect data about it, reflect on them, and draw consequences about process changes to try out. This fundamental approach is appropriate in many situations, and it can be tried out in a school or training environment, which makes it an adequate approach to letting developers taking responsibility for their work processes.

## 4 Costs and Risks of the Approach

The general concept of organizing software development projects as controlled learning projects also carries some costs and risks with it. In this section, we will just list some of them, without mentioning possible methods to reduce them.

- Developers have to be made sensitive to their learning needs.
- Developers have to be educated about how they can collect data describing relevant features of their work process.
- Data collection takes some time, and some discipline.
- Reflexion needs time. The collected data have to be interpreted.
- Changing the process in a multiperson project means that a consensus has to be found regarding the interpretation of the data and the consequences to be drawn from them.
- Process change experiments sometimes fail, and before this is noticed by a team, the team might perform worse than if they had adhered strictly to a standard process.

  This problem is relevant especially if the environment of the team and the team structure are not really turbulent. In this case, an optimal process might be definable before a process starts.

- A base process and a problems/solutions catalogue have to be maintained, and consultancy has to be provided.

  This is not an additional cost to an approach using fixed processes, since also there, a standards department has to exists which maintains the descriptions of processes to be followed. In the controlled learning approach, the tasks of the standards and process department change.

- Permanently changing processes cost more effort. A well exercised process will be executed easier than a constantly changing process.

  Because of this problem, it is important not to change too many process elements at a time.

- Consultants have to help the developers to recognize their learning needs and to help them find appropriate ways to learn.

  It costs additional efforts to demonstrate a learning need to someone; it would be easier just to prescribe some process to use, since it takes time to convince somebody. The risk exists that some developer or some developer group might not be convinced about some existing learning need.

- It might be too much for developers to focus at the same time on their concrete development task and on changes needed in their software development process. This might imply non-use or even mis-use of the discretion.

- Often, development teams change in the course of a project. If there is a standard process to which developers are accustomed, it is easy for them to work smoothly in another group, since the procedures to follow are the known ones. But if the used process is changing, it takes more effort to learn about the differences to the base process used in an organization.

- The total additional effort spent for the process optimization might outweigh the gains from this optimization. But this difference between effort and gains can not be measured, since the unchanging process is not enacted in parallel to the changing learning-based process.

## 5 Summary

It has been shown how a conceptualization of a software development process as a controlled learning process can put the control over their work process into the hands of the developers. Consequences to be expected by the adoption of a process based on the development team's learning to perform in changing environments is that they learn to value process definitions and data collection as tools for their learning and performing, instead of rejecting them as instruments which they are controlled through.

Additionally, developers enhance their capabilities to reflect on what they are doing. The ability to reflect one's practice (cf. [7]) is an important precondition for progress in one's professional life.

The experiences made by the developers in their specific projects can be made accessible to other projects by collecting relevant case stories about problems and solutions found for them in a catalogue.

Each project has its pecularities with respect to composition of the group of developers, the interaction with the customer, the task to be solved by the development team and the infrastructure which it can use, and from these pecularities follow specific requirements for the process organization. In changing environments it is appropriate to put responsibility for adequate reactions on environment changes into the hands of people who recognize such changes as the first ones.

It should not be ignored that many costs and dangers are involved with such an approach. The most important is the one that the freedom they are provided by the approach is not used appropriately by some developers.

Currently, there is only a small number of developer-centered approaches to software development processes. Humphrey's PSP and the Extreme Programming approach propagated by Kent Beck are known today as two quite dissimilar but both fairly encompassing activities in this respect. Both contain some elements of the approach described here, and our approach owes much to either of them. But neither puts the research component into the focus.

## References


[1] Ann Anderson, Ralph Beattie, Kent Beck, David Bryand, Marie DeArment, Martin Fowler, Margaret Fronczak, Rich Garzaniti, Dennis Gore, Brian Hacker, Chet Hendrickson, Ron Jeffries, Doug Joppie, David Kim, Paul Kowalsky, Debbie Mueller, Tom Murasky, Richard Nutter, Adrian Pantea, and Don Thomas. Chrysler goes to "extremes". *Distributed Computing*, pages 24–28, October 1998.

[2] Chris Argyris and Donald A. Schön. *Organizational Learning II: Theory, Method, and Practice*. Addison Wesley, Reading/Massachusetts, 1996.

[3] B. W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 5(21):61–72, 1988.

[4] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. International Thomson Computer Press, London et al.,[2] 1996.

[5] Watts S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, Reading/Massachusetts, 1995.

[6] Jürgen Pasch. *Softwareentwicklung im Team*. Springer-Verlag, Berlin, 1994.

[7] Donald A. Schön. *The Reflective Practitioner*. Basic Books, New York, 1982.