The Concept of Change in Technology Transfer

Orit Hazzan
Department of Education in Technology and Science
Technion – Israel Institute of Technology
Haifa, Israel
(972 4) 829 3107
oritha@tx.technion.ac.il

Yael Dubinsky
Department of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel
yael@cs.technion.ac.il

ABSTRACT
In this paper we describe two case studies of technology transfer, both of which address the adoption of Extreme Programming, the most prevalent agile software development method. The first case takes place in an industrial setting; the second – in academia. For the analysis of these cases, we use two concepts: a software development method lifestyle and the concept of change. We propose that these concepts, as well as a combination thereof, can serve as a framework within which change processes in general, and technology transfer events in particular, can be analyzed.

Categories and Subject Descriptors
D.2.9 [Management]: Life cycle, Programming teams, Software process models.

General Terms
Management, Measurement, Human Factors.

Keywords
Software development methods, extreme programming, agile software development, change processes, technology transfer.

1. INTRODUCTION
In this paper, we examine technology transfer from the perspectives of both the academia and the industry. Specifically, we present two case studies of how Extreme Programming (Beck, 2000; Beck with Andres, 2005) was adopted within these two frameworks. The first case study describes the adoption of Extreme Programming (XP) by a military software development unit that was working on a large-scale software project. The second case study describes how XP was adopted in academia, specifically by the Department of Computer Science at the Technion – Israel Institute of Technology, as the software development method (SDM) used in the Operating Systems Lab.

We limit our discussion of technology transfer to the adoption of SDMs. To convey our ideas, we first introduce the concept of an SDM-lifestyle, which is a meta-method that guides the actual adoption of an SDM (whether in academia or in industry). Specifically, an SDM-lifestyle guides both the process of the adoption of an SDM and its maintenance by the organization.

Second, we present Henry Plotkin’s perspective on change, as it is presented in his book “Darwin Machines and the Nature of Knowledge” as part of the chapter dealing with the evolution of intelligence. Plotkin’s perspective is used for the analysis of the two case studies.

This position paper is based on our intensive experience in technology transfer in general, and in the adoption of SDMs in particular, both in academia and in industry (cf. for example, Hazzan and Dubinsky, 2003 and Dubinsky and Hazzan, 2005, as well as our website http://edu.technion.ac.il/Courses/cs_methods/eXtremeProgramming/XP_Technion.htm).

2. SDM-LIFESTYLE
This section introduces the concept of SDM-lifestyle, which is a meta-method that guides both the way in which the knowledge needed to use an SDM is gained, as well as the way the use of the SDM is sustained on a daily basis. To clarify the difference between an SDM and an SDM-lifestyle we will add that while an SDM is the way in which a software team works in order to provide a high-quality software product, an SDM-lifestyle is the way used (if at all) by software organizations in order to, first, integrate the SDM into the organization’s software development processes and, second, ensure its spirit is maintained on a daily basis by their software teams.

As a metaphor for this idea, let us look at real life. We are each aware of our own personal method of maintaining a good, healthy lifestyle; e.g., what to eat and what sport or exercise to do. This is our SDM. Still, there is yet another method, in fact a meta-method, that we use in order to achieve this lifestyle and to maintain it. This meta-method is the method that first guides our learning and integration of the said good habits into our life and second, helps us maintain these habits in our daily schedule. This is our SDM-lifestyle.

As in the case of a healthy lifestyle, we sometimes do not know exactly how to start working according to a specific SDM or how to form a team that will work according to it. Therefore, in most cases, in order to integrate a new SDM into our team or organization, we must learn it gradually and begin its implementation in small steps till we attain a level of expertise that enables us to maintain the SDM-lifestyle.
By way of analogy, we know what level we wish to attain in our gym workout, yet we cannot reach it immediately. We must learn first about the different gym equipment and the ways in which they affect our body. Only gradually do we reach our target level for keeping in shape. In the process of reaching the desired level, we constantly work on maintaining the healthy lifestyle, trying to improve our exercising routine according to the new healthy ideas, learning new workout technologies, etc.

Similarly, a software team that learns and reaches the expected level of SDM implementation, must start working on its maintenance and keep improving its development habits with respect to methods, advanced technologies and tools. In terms of technology transfer, an SDM-lifestyle means a mechanism that supports the technology transfer.

The concepts of SDM and SDM-lifestyle can be expressed with respect to Extreme Programming (XP) – the technology whose transfer is examined in this paper, as follows: The first book published on Extreme Programming (Beck, 2000) presents an (agile) SDM that describes very precisely the details of what a software team should do, on a daily basis, in the development process in order to produce a high-quality software product. In this paper we refer to the XP-lifestyle, which is a meta-method that guides both the acceptance of XP by the organization (which wishes to adopt it) as well as the maintenance of XP's strictness and spirit on a daily basis.

### 3. THE CONCEPT OF CHANGE

The concept of change is presented in Henry Plotkin’s book "Darwin Machines and the Nature of Knowledge" (1997) as part of the chapter on the evolution of intelligence:

Change is a universal condition of the world. If the world were unchanging, then evolution would have proceeded to some optimal point and then ceased. This has not happened. Nothing stands still, and the very occurrence of evolution is both a force for change itself and proof positive for its existence. (p. 139).

Changes are around us all. Specifically, in software development processes we witness many changes, including that with which we are most familiar, namely, changes in customer requirements. Still, this is not the only change in software development processes. Changes take place constantly in different areas, such as technology, economy, and society. The main question Plotkin addresses is how to cope with the uncertainty introduced by changes. He describes two main sets of solutions to deal with changes and explains how they enable to cope with changes in practice (Plotkin, 1997, pp. 145-152). Although Plotkin’s examples are taken mostly from the lives of animals, it is striking how his framework is suitable for coping with changes that take place in software development processes as well.

The first set of solutions is concerned with reducing the scope of change (see left-hand branch in Figure 1). One way to do this is by reducing the time period between conception and reproductive competence, i.e., maintaining a high reproductive output in a relatively short period of time (see branch T in Figure 1). The second way to reduce the scope of change, according to Plotkin, is to live in a relatively isolated and unpopulated place (see branch P in Figure 1). A variation of this idea is when parents attempt to protect their offspring by isolating them.

The second set of solutions for coping with changes takes the form of "if you can’t beat it, join it", i.e., create the phenotypes in such a way that will enable them to change together with, and remain compatible with, the changing features of the world (see right-hand branch in Figure 1). The first strategy used in order to accomplish this is to produce large numbers of different offspring in order to increase diversity so that the chance that at least some individuals will withstand the change increases (see branch D in Figure 1). The second strategy, named the "tracking option", is to produce phenotypes that change in response to changes in the world (see branch K in Figure 1). The tracking option is accomplished by using knowledge-gaining devices, one of which – the brain mechanism, known as rationality or intelligence – operates in the sphere of the physical world of events and objects.

![Figure 1. Solutions for Dealing with Change (Plotkin, 1997)](image)

### 4. CASE STUDIES

In this section we describe two case studies of technology transfer in general, and of Extreme Programming (XP) adoption, i.e. attaining an XP-Lifestyle, in particular. We analyze these cases in which an XP-Lifestyle was attained according to Plotkin's framework for coping with change, described in Section 3. The first case is characterized by the fact that the scope of change suited the organization and enabled gradual growth of the change (branch P in Figure 1). The second case is characterized by the knowledge-gaining mechanisms that were introduced into that process (branch K in Figure 1).

#### 4.1. Case Study 1 – Technology Transfer in Industry

This case takes place in a military software development unit. Following a very short conference presentation, attended by one of the unit's officers, we were invited to present XP to the entire
unit. We insisted that this presentation last at least three hours. In
the presentation, we adopted the format described by Dubinsky,
Hazzan and Keren (2005) which its main feature is experiencing
the SDM as much as possible. Following the presentation, one of
the officers, Ron, approached us. Ron was in charge of a software
project whose target users population consisted of thousands of
people and which was being developed by several dozen skilled
software developers and testers, who were organized in a
hierarchical structure of smaller groups.

Ron told us that he had been asked to change the current software
development process to a new process that will enable first, rapid
response to customers' requests and changes, and second, a
communication channel for feedback with respect to released
features. Ron had one year to implement this change.

Since Ron's unit was large, it was clear that such a change could
not be performed in one single step. Rather, it was obvious that
the transition must be a gradual process and should be planned
accordingly. Ron asked us to help him plan the transition to the
new development process (which had not yet been defined at that
time). More specifically, Ron asked us to help him plan and work
on a (non-software) project, which was to be, in fact, a new SDM-
lifestyle, aimed at planning the assimilation process of a new (yet
unknown) development process. From an organizational
perspective we can say that Ron's mission was to establish an
organizational change, the highlight of which would be expressed
by a new SDM-lifestyle, which would include the adoption of an
unknown software development process aimed at improving the
software development process and software quality.

On the face of it, it seems strange: Can't the military system
command its members to work according to a specific SDM? If
yes, then why must a transition process be planned? If no, then
why does this situation occur? As it turns out, commands cannot
be forced in software development projects, for several reasons.
First, such a process involves many unseen factors. It is therefore
impossible to define explicit rules for each of them and to enforce
their performance. Second, the accepted working habits within the
unit were so rooted that even if a change were to be forced, the
traditional development process might eventually return through
the back door. Third, human communication, which is such a vital
factor in software project, cannot be forced.

All these reasons explained the need for an SDM-lifestyle – a
process that would gradually lead the transition and the
assimilation of a new SDM by Ron's team. Such a process should
take into consideration the individual interests of people from
different parties, the possible resistance each party might raise,
and the harmony and synergy between the different changes that
are to take place in each process and role definition.

The actual work began when we met. Ron started by describing
the incremental model according to which the team was currently
working. The following basic problems were identified:

- Feedback processes were too long and, as a result, it
  sometimes could take 18 months to get feedback from users
  who had asked for a change. In many cases no feedback at
  all was received from the users. Yet, in other cases when
  feedback was received and addressed, it was no longer
  relevant.
- Only a few people communicated directly with the
  customer.
- Developers had no contact with the users.
- Intercommunication between team members was
  cumbersome.
- Each developer worked on only one abstraction level:
  Developer roles were very narrow and, therefore, they were
  forced to remain at one specific abstraction level (such as
  code level or design level) during the entire software
development process, and had no opportunity to examine
  the software product from another (more global or more
  local) perspective, if needed.
- Resistance to change in the current process: The source of
  this resistance was peoples' concern of losing some of their
  authority in the current organizational structure. At the same
time, those team members who did feel a need for change
  were positioned relatively low in the hierarchy and,
  therefore, their ideas could be easily blocked by their
  superiors.
- The ratio of "not developers" to "developers" was high (3:1)
  and only a small number of people actually wrote code,
  while other team members worked on requirement
  gathering, design activities, and testing. This structure is, in
  fact, one component of the "one level of abstraction"
  problem described above.
- There was no personal responsibility for any specific task; It
  was not clear who was in charge of what.
- There was a general lack of time to deal with all these
  problems and with the problem of lack of time itself, which
  partially contributes to all of the above-mentioned problems.

In preparation for the said transition, a task force of ten volunteers
was established. Its aim was to formulate an SDM-lifestyle that
would lead the process, at the end of which the entire unit would
be working according to the new desired SDM. Note that at that
preliminary stage, the scope of the change in the process was
small, both in terms of the number of people involved and the
activities undertook. This is the first key to the success of the
process, as will be explained later.

It is important to note that the entire unit leadership supported this
process. Furthermore, the leadership declared specifically that,
while a decrease in quantity might be acceptable, quality and
compliance with customer needs must not be compromised up. It
was realized that resistance might be expressed by the mid-level
officers.

As mentioned, the task force was composed of ten members,
representing all levels of the team. Meetings were held once every
two weeks. The task force worked according to the following
agile guideline. Note that agility is applied for the planning or the
SDM-lifestyle project and not for the software development
process itself.

- Small versions and iterations are defined.
- The customer's side (Ron) is constantly represented.
- Customers’ stories are gathered and divided into iterations.
- Customers’ stories are decomposed into specific tasks.
- Tasks are assigned at every meeting to each member of the
  task force. The members estimate the time needed to
perform each task and schedule its accomplishment within their (tight) schedule.

- Stand-up meetings take place at the beginning of each meeting.
- Special roles are assigned to team members in order to control the process.
- Tests are defined in order to check whether customer's stories were performed as required.

Thus, for example, the planning game was played in the first meeting, with Ron playing the role of the customer of the planning project, and different assignments (such as, reading and learning relevant material, mapping the project current organizational structure, etc.) were chosen by the members of the task force for the first two-week iteration.

The second meeting, which took place two weeks later, opened with a stand-up meeting. Then, the following roles were assigned: coach, tracker, on-site customer, acceptance testing coordinator, presentation coordinator, documentation coordinator, design coordinator, code effectiveness and correctness coordinator. Since the project involved planning an SDM-lifestyle project rather than a software project, roles were defined slightly differently than in software development processes, as described by Dubinsky and Hazzan (2004).

Next, the members of the task force presented their first iteration products on which they had worked during the past two weeks. Then, the planning game for the second iteration of the first released was played. The meeting ended with a retrospective session of 20 minutes in which the following two main topics were discussed: a) the increased ability to cope with many tasks if they are planned, allocated and carried out properly, and b) the suitability of this work mode to the work of the task force.

At present, the SDM-lifestyle project is proceeding. XP is the SDM whose lifestyle was adopted. So far, three teams have participated in our workshops and have started to work according to XP. Scalability issues are discussed now at the unit level. Two more teams are scheduled to participate in our workshops.

The three teams that are working according to XP are guided by a software process into an organization. This case study also shows that in order to initiate such a process successfully, attention should first be given to implementing a change of small scope and only gradually should the process be expanded. Such an approach enables to establish a controlled process consisting of many checkpoints for the evaluation and the assimilation of the SDM-lifestyle in the organization.

Specifically, since Ron was aware of the fact that there was a significant gap between his perception and that of the other team members with respect to the culture needed for the successful accomplishment of the software project, Ron adopted an approach that reduces the scope of change during the initial stages. In other words, instead of starting the cultural change in parallel to the introduction of a new SDM, on which he would have had to decide himself, he started by preparing the human infrastructure needed for that change. He knew that during this period he would be able to improve his understanding with respect to people's concerns, significant problems in the current process, relationships between different roles and people in the project, and other similar (mainly human-related) issues. He realized that an improved understanding of all these factors would eventually help him, together with the task force members, formulate a SDM-lifestyle process that, on the one hand, would meet the system's needs and addresses the important problems of the current process, and on the other hand, would take into the consideration the interests and viewpoints of the people involved.

Clearly, such a model can decrease resistance towards the new SDM when actually introduced and applied.

In short, examination of the case within the change framework reveals that Ron simply reduced the scope of change, a fact that enabled him to cope with the needed change gradually, whereby each step was examined carefully and the lessons learned inspired subsequent steps in the change process.

4.2. Case Study 2 – Technology Transfer in Academia

One of the advanced courses offered by the Department of Computer Science at the Technion, and taught by the second author for the past ten years, is the project-based capstone course "Projects in Operating Systems". Since the summer semester of 2002, the course has been taught in a studio-oriented format. The “studio” is the basic learning method used in architecture schools. In such studios, closely guided by an academic coach, students develop projects while performing on-going reflection both on what is created and on the creation process itself (Kuhn, 1998; Tomayko, 1996). Hazzan (2002) analyzed the suitability and implementation of the studio-based teaching approach for software engineering education. Extreme Programming (Beck, 2000; Beck with Andres, 2005) was the development method chosen to be introduced into the studio in the said course.

Since the summer semester of 2002, the above course format was applied in the Operating Systems Projects course to about 40 projects. The initiative was and is highly supported by the Department of Computer Science, which, among other things, provides the required resources and equipment: Each team of 10-12 students works on a specific operating system project during the entire semester. Each team has its own studio, equipped with computers, tables and white boards. Each team also has an academic coach who guides the development process (but does not manage it!).

Attendance of all students at all of the weekly 2-4 hour sessions is compulsory. In these meetings, XP is taught and employed. In between sessions, communication is conducted via an electronic forum, which is part of the course web-based environment. This implementation of XP into the course enables the development of
large-scale operating systems projects. Furthermore, it enriches the development environment with respect to topics such as customer needs and process management (Dubinsky and Hazzan, 2005).

**Case Study 2 - Analysis**

In what follows we present several principles of the XP-lifestyle that guides the adoption and maintenance of XP by the Operating Systems Projects course. In general, we characterize this case by the fact that the SDM-lifestyle contains knowledge-gaining devices. Specifically, we present two such tools.

**a. A team of coaches guides the SDM-lifestyle:** Software development is a complex process. Therefore, one of the concepts of the agile software development approach in general, and of XP in particular is the Whole Team, whereby interaction among team members helps sustain the development pace. In the same spirit, in the case of the adoption of XP as described here, a team of supervisors began supervising the student teams, each coach supervising one team. In a similar manner to agile development, this helps maintain the spirit.

Specifically, a training program was implemented prior to the actual supervising of the SDM-lifestyle (Dubinsky and Hazzan, 2003). The training program was based on four meetings: In the first, the general ideas of XP were introduced; in the second, the structure of the studio was discussed; in the third, the supervisor’s role in the lab and the teaching of XP practices were addressed; and in the fourth meeting, an evaluation scheme was agreed upon.

During the actual supervision process, the team guiding the SDM-lifestyle makes decisions together, yet each supervisor can make specific decisions related to his or her team, such as selecting the students’ project according to his or her areas of expertise.

**b. Self expression and reflection:** In order to learn about the feelings of students, their work habits and their social interactions with respect to the project development during the course, the students were offered tools and means for self-expression. The ability to express their impressions gives students the feeling that their thoughts and feelings are of interest to the course instructors and supervisors. In addition, if students wish to complain, this is a good way to encourage them to do so. In this spirit, students should be encouraged to express not only positive ideas, but also negative feelings and suggestions for improvement.

Furthermore, we believe that such means of expression enhance students’ reflective skills (Schön, 1983, 1987), whose importance in software development processes is acknowledged and which are encouraged by the studio approach. As it is well known in the software industry, a reflective person, who learns both from the successes and failures of previous software projects, is more likely to improve his or her own performance in the field.

**5. Conclusion**

In the two cases described in this paper, as with other SDM-lifestyle changing processes, an initial decision to start the change had to be made. Therefore, from the concept of change perspective, special attention should be given to that initial step. In the first case study described, we saw how a task force was established and the scope of the change has been reduced; in the second case, a team of supervisors began a process which served as a knowledge-gaining device.

These two cases, as well as additional cases from our experience in technology transfer guided by the concept SDM-lifestyles, fit very well into Plotkin’s framework of dealing with change. We will share these cases in the workshop.

**6. REFERENCES**


