# A HOT – Human, Organizational and Technological – Framework for a Software Engineering Course

Orit Hazzan
Department of Education in Technology and Science
Technion – Israel Institute of Technology
Haifa 32000, Israel
oritha@techunix.technion.ac.il

Yael Dubinsky
IBM Haifa Research Lab
Mount Carmel
Haifa 31905, Israel
dubinsky@il.ibm.com

## ABSTRACT
In this paper, we present a HOT – Human, Organizational and Technological – framework for software engineering and describe its application in a full one-semester software engineering course on agile software development. We suggest and illustrate how this framework has the potential to widen and deepen the students' understanding of software engineering processes.

## Categories and Subject Descriptors
K.3.2 [**Computing Milieux]:** Computers and Education - *Computer and Information Science Education*

## General Terms
Management, Measurement, Performance, Human Factors.

## Keywords
Software engineering education, HOT – Human, Organizational and Technological – framework, software engineering course, agile software development.

## 1. INTRODUCTION
The contents of many Software Engineering (SE) books highlight the activities performed during a software development process, e.g., requirements gathering, requirements analysis, design, coding, testing, deployments, etc. (e.g., Jalote, 2005; Hamlet and Maybee, 2001; Peters and Pedrycz, 2000; Sommerville, 2006). In the paper we present an alternative approach for an introductory SE course. This course examines the discipline of SE through a three-facet framework, namely, the HOT – Human, Organizational and Technological – framework.

We suggest that such a perspective at SE should be reflected in a SE course by a different set of topics than the activities performed in a typical software development process. These topics should highlight topics of concern in software development processes, such as time, quality, learning, and change – all are basic and central factors in any software development process; in addition, they are relevant for more than one SE activity.

In the rest of the paper we describe the HOT framework, present the structure of a full semester course that illustrates the appropriateness of this framework for an introductory SE course, and outline 14 main topics addressed in such a course.



*ICSE '10,* May 2-8, 2010, Cape Town, South Africa


## 2. HOT – HUMAN, ORGANIZATIONAL AND TECHNOLOGICAL – FRAMEWORK
It is widely accepted today that SE should not address only technological aspects, but rather, it should refer also to the work environment and to the professional framework. Accordingly, we suggest examining the field of SE through the following three perspectives:

- The **H**uman perspective, which includes cognitive and social aspects, and refers to learning and interpersonal (teammates, customers, management) processes;
- The **O**rganizational perspective, which includes managerial and cultural aspects, and refers to the workspace and issues that extend beyond the team;
- The **T**echnological perspective, which includes practical and technical aspects, and refers to how-to and code-related issues.

Figure 1 presents a schematic view of the HOT framework.

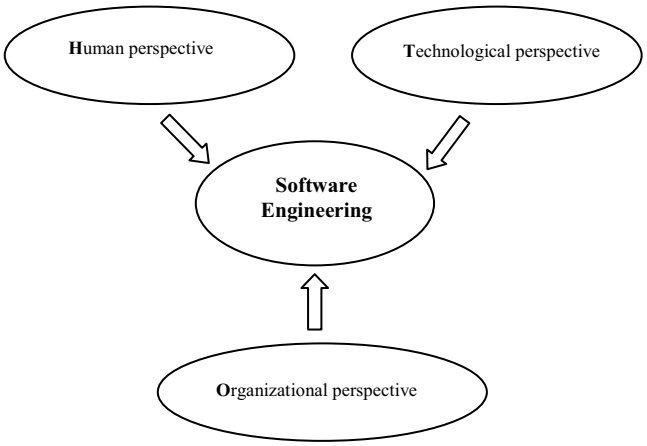


**Figure 1. The HOT Framework for Software Engineering**

## 3. COURSE DESCRIPTION
### 3.1 Course Structure
The SE course described in this paper, which employs the HOT framework, includes fourteen lessons, organized in three iterations. This structure enables to revisit the different subjects addressed in the course several times during the course as well as to guide the development of one-release software product (see below). Table 1 presents the course structure and the topic of each lesson.

**Table 1. Course Structure**

| Iteration | Lesson # | Topic |
|---|---|---|
| I | 1 | Introduction to Agile Software Development |
| | 2 | Teamwork |
| | 3 | Customers and Users |
| | 4 | Time |
| | 5 | Measures |
| | 6 | Quality |
| | 7 | Learning |
| II | 8 | Abstraction |
| | 9 | Trust |
| | 10 | Globalization |
| | 11 | Reflection |
| III | 12 | Change |
| | 13 | Leadership |
| | 14 | Delivery and Cyclicality |

It is recommended that the course is based on two main components that progress in parallel and are closely correlated to each other. The first component is theoretical and can be delivered in the lecture hall or the class; the second component is based on the development of a software project and it takes place in a physical learning environment that we call a studio or lab. Such a course structure enables learners become familiar with software development concepts both from a theoretical perspective (in the lectures) and from the practical perspective (in the studio).

In more details, the studio is a learning environment in which an intensive student-student and student–academic coach interaction accompanies the software development process (Kuhn, 1998; Tomayko, 1996; Hazzan 2002). It is the basic learning method used in architecture schools and has been central to architectural training for most of the twentieth century. In the architectural studio, a group of students meet their academic coach in the studio three times each week. Such an intensive schedule puts pressure on the students and, as a result, they devote themselves to the art of design: students learn the skills, the professional language and the discipline ways of thinking. In addition, the studio is the place where students reveal their personalities and can express their professional skills. Moreover, this learning environment exposes students to different kinds of social interactions such as, work on team projects, contribution to class discussions, and presentation of their products in front of their classmates.

Based on the suitability of the studio approach to architecture education, it is suggested applying this teaching approach to other disciplines, mainly to professions in which design considerations are inherent, there is more than one approach to tackle a given problem, and usually a solution to a given problem is not unique. It seems clearly that SE fits perfectly for the adoption of the studio as a learning environment. For a boarder discussion of this suitability see Hazzan, 2002.

## 3.2 Teaching Principles

The course teaching is based on principles which support gradual and meaningful learning of the course (See Table 2, based on Hazzan and Dubinsky, 2008). For example, Principle 4 - Elicit reflection on experience – leads learners to keep improving their understanding of the learned concepts; Principle 7 - Assign roles to team members – not only further the students' involvement in the course and their responsibility in the development process, but in addition, enables them to deepen their understanding of the topic on which their role concentrates; Principle 11 - Emphasize the software development approach in the context of the world of software development – specifically tells us to connect the taught material to software development environments in the real world.

**Table 2. Teaching Principles**

| Principle # | Description |
|---|---|
| 1 | Inspire the nature of the software development approach |
| 2 | Let the learners experience the software development approach |
| 3 | Explain while doing |
| 4 | Elicit reflection on experience |
| 5 | Elicit communication |
| 6 | Establish diverse teams |
| 7 | Assign roles to team members |
| 8 | Manage time |
| 9 | Be aware of abstraction levels |
| 10 | Use metaphors or "other world's concepts" |
| 11 | Emphasize the software development approach in the context of the world of software development |

## 3.3 Course Topics

The HOT framework is illustrated in what follows by an introductory SE course that emphasizes the agile approach (Hazzan and Dubinsky, 2008).

The rational for teaching agile software development is presented in our 2007 paper – Why Software Engineering Programs Should Teach Agile Software Development – published in *SIGSOFT/Software Engineering Notes* (Hazzan and Dubinsky, 2007). Specifically, we propose ten reasons why it is important, suitable and timely to introduce agile software development into SE programs (see Table 3). We mention them here to highlight the appropriateness of applying the HOT framework for a full SE course on agile software development.

For example, Reason #3 explicitly mentions the Human aspects of the HOT framework; Reason #7 explicitly addresses the Organizational aspects of the HOT framework; Reason #1 refers mainly to the Technological aspect of the HOT framework.

In what follows we briefly describe the contents of each lesson. In addition, for each lesson, we present one or two examples of tasks, presented to the students in the said lesson, in which the students analyze the topic of the lesson within the HOT framework, and explain how such an analysis may contribute to the students' understanding of SE concepts.

**Table 3. Reasons for Introducing Agile Software Development into SE programs**

| Reason # | |
|---|---|
| 1 | Agile development was evolved and is applied in the industry |
| 2 | Agile development educates for teamwork |
| 3 | Agile development deals with human aspects |
| 4 | Agile development encourages diversity |
| 5 | Agile development supports learning processes |
| 6 | Agile development improves habits of mind |
| 7 | Agile development emphasizes management skills |
| 8 | Agile development enhances ethical norms |
| 9 | Agile development highlights a comprehensive image of software engineering |
| 10 | Agile development provides one possible teachable framework |

## 3.4 The HOT Analysis Framework in Practice: A Full Semester Course Description

**Lesson 1: Introduction to Agile Software Development.** In this introductory lesson of the course, questions such as the following ones are addressed: What is agile software development? Why is an agile approach for software engineering needed? What are the main characteristics of agile software development? What can be achieved by agile software development processes? Does agile software development form a pleasant and professional software development environment? At the end of the lesson, in addition to several insights about agile software development in general, the students also conceive of the nature of agile software development and are able to clarify how it establishes a professional software development environment in which software engineers are able to express their skills and, at the same time, to produce software products on high quality.

*Examples of HOT tasks:*

1. *Describe the development process you used for the development of the last software project you developed. Analyze the process benefits and pitfalls according to the HOT framework.*
   In this task the students are asked first to describe the development process they employed in their last software project. At the second stage, they are asked to analyze it within the HOT framework. Such an analysis encourage them, first, to identify on which aspect(s) they focused in their description, and second, to complete their description by addressing also the aspect(s) they did not address in their first description. The motivation of this task is to let the students feel the relevance and applicability of the HOT framework from the very early stages of the course, and with respect to their own experience in project development.
2. *Specify how the HOT perspectives are expressed in each of the four principles of the Agile Manifesto and in the Manifesto as a whole.*
   In this task the students should examine the Agile Manifesto (see Table 4), which on the one hand, its formulation is very short and conceptual, and on the other hand, it encapsulates the essence of the agile paradigm for software development. Students' challenge is to elicit the multi-faceted essence of the Agile Manifesto by its examination within the HOT framework.

**Table 4. Manifesto for Agile Software Development**[1]

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

**Individuals and interactions** over processes and tools
**Working software** over comprehensive documentation
**Customer collaboration** over contract negotiation
**Responding to change** over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

**Lesson 2: Teamwork.** This lesson presents one of the basic elements of software projects – teamwork. It addresses how to build teams in a way that promotes team members' accountability and responsibility, as well as fosters communication between teammates. One of the basic ways to start team building is by assigning roles to team members. For this purpose a role scheme is presented in this lesson (Dubinsky and Hazzan, 2006), according to it each team member is in charge of a specific managerial aspect of the development process, such as design and continuous integration, in addition to his or her development tasks.

It is not always simple to reach fruitful teamwork and sometimes it raises dilemmas and conflicts between team members. This aspect of teamwork is not neglected in agile teams and, when a conflict emerges, it is addressed openly by all the team members. Agile mechanisms to overcome such conflicts are discussed in this lesson as well.

*Example of a HOT task:*

*How does the role scheme reflect the three aspects of the HOT framework?*

In this task, students should take the individual perspective and examine the role scheme by considering his or her involvement in the development process. The HOT framework encourages them to consider not only mere Technical work, but also the individual's place within the team (the Human aspect) and within the organization (the Organizational aspect).

**Lesson 3: Customers and Users.** Customers have a key role is agile software development processes. The Agile Manifesto statement 'customer collaboration over contract negotiation' (see Table 4) explicitly suggests alternative work relationships with the customer, in which the customer is part of the team and not an entity with which team members and management argue what should be developed and when, and what should not be developed at all. This lesson describes the customer role in agile development environments and how to enhance communication with the customer.

[1] The Agile Manifesto website http://www.agilemanifesto.org/

In many cases, it is expected to evaluate the software product also from the user perspective and the customer represents also the user group. This, however, turns out to be an ineffective mechanism in many cases. Accordingly, this lesson also describes the users' role and presents a mechanism which enables to increase the users' involvement and feedback during the development process. Specifically, data obtained from user evaluation enable to refine the user interface design, to increase product usability, and, consequently, to increase the product quality.

*Example of a HOT task:*

*What are the main characteristics of the Business Day? For each characteristic, specify which HOT – Human, Organizational and Technological – perspective you use.*

In agile software development processes, a Business Day takes place between each two consequent iterations of the release: at the end of each iteration and at the beginning of the next one. The rest of the iteration days are development days. In addition to the team and the customer, other project stake holders are invited to participate in the Business Day, including managers and external parties, such as customer associates and users. In the first part of the Business Day the previous iteration is summarized; in its second part, after a reflective session takes place, the next iteration planning starts.

This task, in which the students should examine the Business Day by the HOT framework, boarders their perspective at this event (especially if they participate in such an event in their development processes). It is, therefore, reasonable to assume that such a task increases the students' awareness to the centrality of the Business Day and to its crucial impact on the entire software development process.

**Lesson 4: Time.** Time plays a special role in SE: the project schedule should be met, the product should be delivered on time, teammates should complete their tasks on time, and so on. This lesson deals with how the time concept is expressed in SE in general and in agile software development environments in particular. Specifically, in agile software development time is boxed by short iterations of about 2-3 weeks each, and when needed, instead of 'moving' deadlines, the scope is changed accordingly to the customer priorities. This conception is supported by agile software development methods in different ways that not only enable to work in a sustainable pace, but also result in high quality products.

*Example of a HOT task:*

*Analyze the concept of time as it is manifested in agile software development processes from the HOT perspectives.*

Time is addressed differently by different people and cultures; for example, in western culture, time is mainly associated with financial profit, i.e., "Time is money". As mentioned in the above lesson description, this lesson highlights how time plays a special role also in SE. This perspective at time is further enforces in agile development environments. The aim of this task, which asks students to examine the concept of time within the HOT framework, is to lead students feel the special meaning of time as it is manifested in agile software development environments, to understand the source and reasons of this meaning, and to comprehend how the way time is considered in agile software development environments eventually improves the product quality.

**Lesson 5: Measures.** One way to improve the control and management of product development processes is by using measures to monitor the different aspects of such processes. In general, SE is a discipline that acknowledges product measures as well as the measuring of the development process; specifically, the agile approach adopts this approach by promoting a constant tracking during the entire software development process. For example, the team velocity explained in Lesson 4 – Time – is one of the important ways by which agile processes control the project progress; the tracker role, presented in Lesson 2 as one role of the role scheme, is responsible for the exact definition and refinement of the team measures, for the collection of the required data, and for the measure presentation in the Business Day.

In agile software development processes, some measures are presented on a daily basis, like the daily progress within the iteration; some measures are presented each iteration, like code coverage or iteration progress within the release; yet, other measures are presented every release, like customer satisfaction or level of product testability.

*Example of a HOT task:*

*Within the HOT framework, analyze the main ideas presented with respect to measures as they are manifested in agile software development environments.*

This general question about measure is presented after one third of the course is learned. It is expected that at this stage of the course students are familiar with the HOT framework, which is kept being presented in the course, and therefore, quite naturally, are able to apply it to any SE concept presented to them; in this case – the concept of measure as it is manifested in agile software development processes.

Specifically, the importance of measures in agile software development processes is highlighted by the HOT perspectives because when measures are taken on a regular basis, all teammates and stakeholders can view them, give feedback, and suggest measure refinements. Naturally, human, organizational and technical aspects can be elicited when the approach towards measures in agile software development processes is analyzed.

**Lesson 6: Quality.** High quality assurance is a fundamental element of every engineering process and is considered to be one of the difficult things to achieve and sustain. Since high quality is also a basic concern of SE, there are values and practices that support the assurance of high quality software products and processes. However, it is not sufficient and, in many cases, software products lack the required quality. In this lesson, among others issues, the reasons for this situation are analyzed.

In addition, it is also described how quality is perceived by the agile approach, by presenting agile values and practices that support and control the process quality, such as, customer collaboration, the planning activity of a typical agile software development process, refactoring, and the feedback gained by exhaustive testing and test automation. Finally, the focus is placed on the practice of Test Driven Development (TDD).

*Example of a HOT task:*

*Represent the analysis of the TDD activity presented in the course within the HOT framework.*

TDD is a programming technique that aims to provide clean and fault-free code (Beck, 2003). TDD means that, first, we write a test case that fails and then we write the simplest possible code that enables the test to pass successfully. TDD implies that new code is added to the software system only after an automated test has passed. In addition, in order to improve the code readability, design and maintenance, refactoring activities are performed (Fowler, 1999), among other reasons, to eliminate duplications.

TDD can help overcome some of the common problems associated with traditional testing as they are apparent in traditional software projects. Based on Dubinsky and Hazzan (2007), TDD is analyzed in the course, addressing cognitive, social, affective and managerial themes. The analysis is structured around arguments frequently offered to explain why, despite the benefits attributed to testing, in many cases, traditional testing is skipped. Such arguments are accompanied by an explanation of how TDD might help overcome these obstacles. To illustrate how the HOT aspects are reflected in this discussion on TDD, we present several such arguments: *Not enough time to test; Testing provides negative feedback*; *Responsibility for testing is transferred*; *Testing is a low-status job; Testing is hard to manage*; *Testing is hard*.

Based on the analysis presented in the lesson, the given task asks the students to specifically indicate HOT aspects expressed in the TDD activity.

**Lesson 7: Learning.** This lesson illuminates the perspective according to which software development process is a learning process. This statement can be explained both from the customers' and team members' perspectives. At the beginning of the development process, customers do not know explicitly and entirely the requirements of their developed product and, gradually, during the development process, they improve their understanding with respect to these requirements. The team members, in parallel, keep improving their understanding of the customer requirements as well as of the developed product.

From this perspective, if a software development process is a learning process, an appropriate learning environment should be provided to all project stake holders. Indeed, this is another characteristic of agile software development environments – they support leaning processes. This aspect is explored in this lesson by an examination of agile software development environments from the constructivist perspective (Piaget, 1977), which is a cognitive theory of learning processes.

*Example of a HOT task:*

*Analyze the practice of short releases within the HOT framework.*

In this task, the students are requested not only to view the practice of short releases as a work allocation mechanism, but also to consider all the project stakeholders whose participation in the development process is influenced by the practice of short releases. Specifically, the main stakeholders, whose working procedure and involvement in the development process are affected directly by and shaped according to this practice, are the managers, customer, developers, and end users. The specific consideration of these stakeholders naturally reveals the relevance of the HOT framework for the analysis of the practice of short releases. It is reasonable to assume, though, that since the task is presented after a lesson about learning, cognitive processes of the different project stakeholders will be mainly addressed by the students.

**Lesson 8: Abstraction.** This lesson continues the previous lesson, by exploring agile software development processes mainly from the cognitive perspective. Based on the working assumption that software development is a complex task also from the cognitive perspective, abstraction is introduced in this lesson as one means used for reducing the cognitive complexity involved in software product development processes.

Though abstraction is a useful tool, it is not always used. Sometimes it is just difficult to think abstractly; in other cases, abstraction is not utilized due to lack of awareness to its significance and to its potential contribution to cognitive processes. In this lesson, it is described how abstraction is expressed in agile software development environments. Specifically, software design and architecture are abstractions presented in this lesson to discuss the concepts of simple design and refactoring. In addition, subjects that have been introduced in earlier lessons of the course are revisited by analyzing them from the perspective of abstraction.

*Example of a HOT task:*

*Analyze the concept of abstraction and the practice of refactoring within the HOT analysis framework.*

The challenge in this task is to consider concepts, which are salient with respect to one aspects of the HOT framework, also from the other two perspectives. In other words, abstraction can be analyzed naturally from the cognitive perspective; students' efforts, however, to analyze it also from the technical and the organizational perspective further their understanding of SE processes. Similarly, while the practice of refactoring can be examined straight forward from the technical perspective, the aim of the task is to let the students realize that their understanding of additional aspects of this activity, can enhance their performances (at least) with respect to refactoring.

**Lesson 9: Trust.** Software is an intangible product. Therefore, it is difficult to sense its development processes by our regular senses, the exact status of the development process is not always clear, and misunderstandings may emerge. Consequently, it is sometimes difficult to establish trustful professional relationships and team members may tend not to trust each other. This lesson focuses on how agile software development fosters trust among team members. It first explains how agile software development processes turn the development process into a transparent process, making the process and the developed product more sensible. Then, within the prisoner's dilemma framework, taken from game theory, it is explained why and how a transparent process indeed fosters trust among team members. Based on this working assumption, i.e., that the transparent nature of agile software development process fosters trust among team members, the focus is placed on how agile software development enhances ethical behavior and diversity in a way that improves process and product quality.

*Examples of HOT tasks:*

1. *Within the HOT framework, suggest connections between the fact that software is an intangible product and its development process.*

This task expands the discussion of the topic presented in class, which focuses on the social (that is, human) aspects of the fact that software is an intangible product, by asking the students to address the topic also from the technical and organizational perspectives. Such an examination widens the students' perspective at software development processes with respect to one of the very basic facts with which any software engineer faces in any software development process – the fact that software is an intangible product. The important of such an examination cannot be neglected, of course.

2. *How is diversity reflected in the HOT aspects of software development processes in general and agile software development in particular?*

   Similarly to the previous task, this task asks the students to examine a topic whose human aspect is very clear, also from organizational and technical perspective. We emphasize this kind of expansion since in the analysis of SE situations it is more common to focus on the technical aspect.

**Lesson 10: Globalization.** In this lesson, the concept of agility in a wider (than software) context is addressed. One topic on which the focus is placed is globalization in terms of distributed teams; the second idea discussed in this lesson is the application of the agile approach for the management of non software projects.

Agile software development has evolved significantly during the last decade. In parallel to the evolvement of agile software development, globalization in software development has also emerged, and software is developed in many cases by teams which are spread across different geographical areas, cultures and nationalities. This reality, named global software development, has advantages as well as disadvantages. The most blatant advantage is the business aspect of cost reduction; the most problematic issues are communication and team synchronization. In this lesson the notion of global software development is described and it is explained how several agile practices help cope with the challenges involved in such situations. Specifically, it is illuminated how the fact that the agile approach guides towards a transparent global software development process increases information flow and project visibility and assists in solving communication and synchronization problems.

In addition, in this lesson, the notion of agility beyond the software world is also examined, and its usefulness in such projects is explored.

*Example of a HOT task:*

*Analyze how the aspects of the HOT framework are expressed in global software development environments.*

This task asks the students to specifically describe the expressions of human, organizational and technical factors in situations with which (most probably) they are not familiar at this early stage of their professional development. However, since many of them are familiar with other (mainly, on-line) distributed environments, they can bring this experience into this exploration. If the students use another world of concepts to explore the phenomenon of global software development, the cognitive idea of metaphor for the understanding of an unfamiliar topic can be highlighted, explained and connected to Lesson 3, in which its application to software development processes was discussed mainly with respect to the customer and users.

**Lesson 11: Reflection.** This lesson focuses on the notions of reflection and retrospective: reflection usually refers to the individual's thinking about what he or she has accomplished; retrospective is usually conducted in teams, and is partially based on the individuals' reflections performed before or during the retrospective sessions. This lesson, which closes the second iteration of the course, serves as an opportunity to understand the theory behind these concepts as well as to add practical details about their actual performance.

*Examples of HOT tasks:*

1. *What HOT aspects might elicit reflective thinking? Illustrate your answer.*
2. *Analyze the reflection and retrospective activities within the HOT framework.*

While the first task asks the students to share personal experiences with respect to reflective processes, the second task directs the students to apply more theoretical analysis. The rationale for the order of these two questions is similar to the rationale attributed to other topics discussed previously in the course (e.g., short releases); that is, it aims to lead the students gradually, from a concrete examination to a more global one, in other words, to let the students increase the level of abstraction step by step. Such guidance is important especially while working on complex topic, such as software development processes in general and reflective processes, discussed in this lesson, in particular.

**Lesson 12: Change.** Coping with change is one of the main challenges in SE processes. In this spirit, and as the fourth principle of the Agile Manifesto – Responding to change over following a plan – reflects (see Table 4), agile software development methods establish a development process that enables to cope successfully with changes introduced during the development process, while keeping the high quality of the developed product. This lesson widens the discussion on change in software development environments and discusses also on change introduction into organizations that plan to transit, or in the transition process, to agile software development. Specifically, an evolutionary framework for coping with change is presented, using it for the understanding of a transition process to agile development process.

*Example of a HOT task:*

*Analyze the notion of change in software requirements within the HOT framework.*

At this stage of the course, the students are familiar with the HOT framework and are trained to explore sophisticated topics through its lens. The examination that this question refers to, however, is not a simple one and it should be expected that the students would not know how to tackle it. In such a case, it can be suggested to the students to focus first on one instance in their development process in which a change has been introduced and to analyze this case first within the HOT framework. Then, gradually, the discussion can be elicited to the notion of change in software requirements in general, as the question asks for.

**Lesson 13: Leadership.** Leadership is the ability to influence people, leading them to behave in a certain way in order to achieve the group's goals. Leadership is independent of job titles and descriptions. Usually, however, in order to lead, leaders need the power derived from their organizational position. There are

different leadership styles, like task-oriented versus people-oriented, directive versus permissive, autocrat versus democrat. While leaders can shape their leadership style according to circumstances, followers might prefer a different leadership style pending on their situation and personality. In this lesson, the leadership style advocated by the agile approach is discussed, namely, a leadership style that empowers the people involved in the product development process. For example, instead of promoting the idea that 'Leaders should keep the power to themselves in order not to loose it', the agile approach fosters the idea that 'Leaders gains power from its sharing'. This idea is expressed, among other ways, by the transparency expressed in agile software development process that makes the information accessible to everyone and enables each team member to be accountable and fully involved in the development process.

*Example of a HOT task:*

*Analyze the coach role within the HOT analysis framework.*

The aim of this task is to let the students feel the meaning associated to team leaders (coaches) in agile software development environments. By focusing on this leadership role, its position, attitudes and specific responsibilities, the students become familiar with the idea that in agile software development environments coaching encompasses much more than technical activities and responsibilities; specifically, this task highlights the potential influences of the coaching style on human and organizational aspects of software development processes.

**Lesson 14: Delivery and Cyclicality.** This lesson inspires the cyclic nature of an agile software development process, which is composed of releases, each of which first, ends with a product delivery to the customer and a reflective process, and second, signals the beginning of the next release. Specifically, the focused is placed on the delivery of a product that the entire release has been dedicated for its development, describing what happens in agile software development processes at the end of the release – the delivery, as well as just before and after it. Specifically, prior to the delivery, the customer examines the product and checks its fitness to his or her expectations; then, the product release is celebrated; finally, after the delivery, a reflective session is facilitated to exploit the lessons learned during the release for the improvement of future developments.

*Examples of HOT tasks:*

1. *Analyze the Release Celebration event within the HOT analysis framework.*

   If the students develop a software product in parallel to the course, it is recommended to let them experience such a celebration in order to help them grasp its essence. Specifically, on the team level, during the last week of the semester, the students made the needed preparations for this celebration meeting. In this meeting, all students from all teams gather for a real celebration with light refreshments. Each team reports on its main achievements with respect to its project, and then a tour is conducted between the different studios to enable students to visit and examine the projects developed by the other teams. After the tour, each group meets its customer and academic coach for the presentation of the first release that was developed in three iterations during the fourteen weeks of the semester. The academic coach facilitates a feedback and reflective session and guides a summary session of the project.
   Table 5 presents an example for a timetable of such a release celebration in the academia for a course with 4 teams (Hazzan and Dubinsky, 2008).

**Table 5. Example for Celebration Timetable at the End of the Semester**

| Schedule | Activity |
|---|---|
| 14:30-15:30 | Project presentations (15 minute for each team) |
| 15:30-16:00 | Open tour in projects' studios |
| 16:00-17:00 | Presentation and discussion in the team studios. This includes presentation to the customer, feedback, reflection and summary session |

   Such a personal experience at the end-of-the-release celebration inspires, at least partially, the feelings of the end of the release as it happens in the real world of software development. The need to address human, organizational and technological aspects with respect to this period in the development process, as the question asks for, further highlights the richness of this event and its importance for the project success.

2. *Analyze the cyclic nature of agile software development processes within the HOT framework.*

   This task encapsulates many ideas explored so far in the course from the three HOT perspectives. This is because the cyclic nature of agile software development touches a variety of complicated topics, such as cognitive processes, feelings, organizational culture, change processes, leadership styles, and more. The focus placed in this task on one specific characteristic of agile software development processes, encourages the students to express many of the ideas that they previously learned, on the one hand, and one the other hand to specify the uniqueness of this particular characteristic of agile software development processes.

# 4. DISCUSSION

The above course agenda, with its special emphasis on the HOT – Human, Organizational and Technological – framework, illustrates several ideas about the nature of SE and of SE education.

First, different HOT aspects are highlighted with respect to different SE topics; nevertheless, in (almost) all SE topics we can find the three perspectives and, therefore, these three perspectives should not be neglected, even when their relevance is not immediately observed.

Second, the HOT analysis framework can be applied on different levels of abstraction with respect to different SE topics; since abstraction is considered as a main theme in SE processes and environments, this attributes of the HOT framework further highlights its suitability for the examination of the field of SE.

Third, students' awareness to the HOT perspectives should be elevated from a very early stage of their exposure to the discipline of SE; if students' awareness to this multi-faceted nature of SE starts to be constructed too late, the students may have already constructed a narrow image of the discipline. This assertion is especially implied to topics with respect to which one aspect is

dominant (for example, with respect to reflection, the human aspects is dominant; with respect to leadership, the organizational aspect is dominant; and with respect to testing, the technological aspect is dominant). If the three aspects are considered with respect to each topic, the students get the message that these aspects are interconnected and none of the aspects should be neglected.

Fourth, the HOT analysis framework serves here as a course framework on agile software development; it is suggested, however, that this framework can be used also for other SE courses. In practice, when a course is redesign within the HOT framework, the curse instructor should examine first what aspect(s) is/are highlighted with respect to each topic addressed in the course, and then to explore the expression of the other aspect(s) with respect to the said topic. It is suggested that such a presentation of SE topics, not only enriches students' understanding of SE, but also helps change the image of SE as it is widely convinced today.

Fifth, the same course can be taught both in academia (as it is described in this paper) and in industry. The focus will be, naturally, on different topics. The reason for these differences is the different personal experiences brought into the course by students (before graduation) and practitioner software engineers. Appropriate adjustments should be made accordingly for the two populations; nevertheless, the spirit of the HOT framework remains and it can keep guiding the course flow and main messages.

# 5. REFERENCES


[1] Beck, K. (2003). *Test-Driven Development by Example*, Addison Wesley.

[2] Dubinsky, Y. and Hazzan, O. (2006). Using a role scheme to derive software project quality, *Journal of System Architecture* **52**(11), 693-699.

[3] Dubinsky, Y. and Hazzan, O. (2007). Measured test-driven development: Using measures to monitor and control the unit development, *Journal of Computer Science*, Science Publication **3**(5), pp. 335-344.

[4] Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional.

[5] Hamelt, D. and Maybee, J. (2001). *The Engineering of Software*, Addison Wesley.

[6] Hazzan, O. (2002). The reflective practitioner perspective in software engineering education, *The Journal of Systems and Software* **63**(3), 161-171.

[7] Hazzan, O. and Dubinsky, Y. (2007). Why software engineering programs should teach agile software development, *SIGSOFT/Software Engineering Notes* **32**(2), 1-3.

[8] Hazzan, O. and Dubinsky, Y. (2008). *Agile Software Engineering*, Springer.

[9] Jalote, P. (2005). *An Integrated Approach to Software Engineering*, Springer.

[10] Kuhn, S. (1998). The software design studio: An exploration, *IEEE Software* **15** (2), 65-71.

[11] Peters, J. F. and Pedrycz, W. (2000). *Software Engineering – An Engineering Approach*, John Wiley & Sons, Inc.

[12] Piaget, J. (1977). Problems of equilibration. In Appel MH, Goldberg, L.S. *Topics in Cognitive Development*, Volume 1: Equilibration: Theory, Research and Application, Plenum Press, N, 3-13.

[13] Sommerville, I. (2006). *Software Engineering*, 8th Edition, Addison Wesley.

[14] Tomayko, J.E. (1996). Carnegie-Mellon's software development studio: A five-year retrospective, *SEI Conference on Software Engineering Education*. http://www.contrib.andrew.cmu.edu/usr/emile/studio/coach.htm.