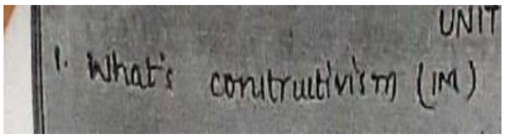
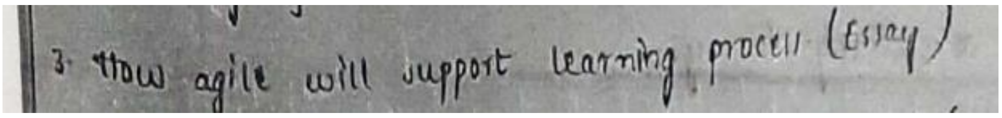
UNIT – 4


UNIT

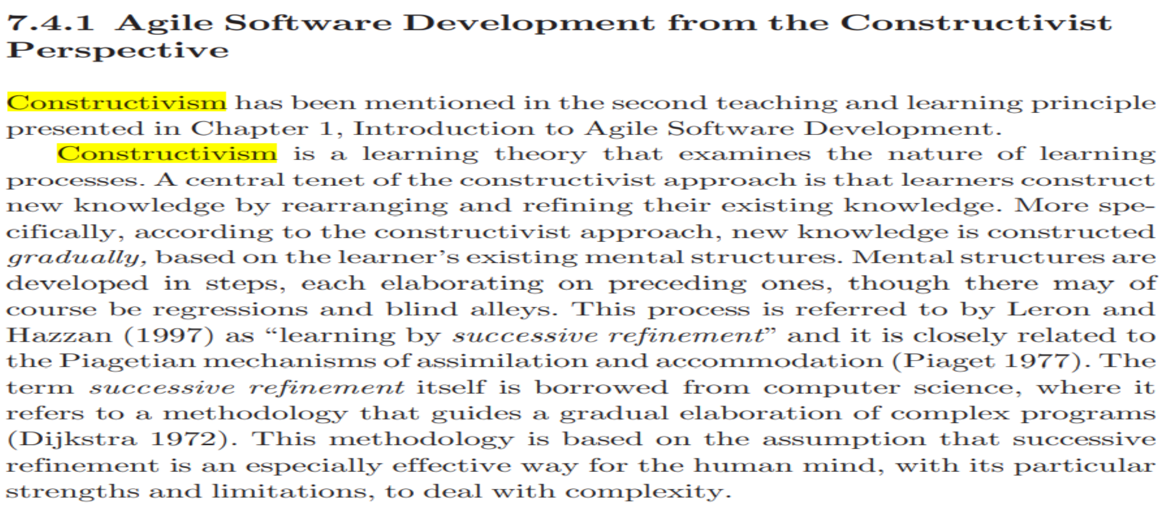
1. What's construtivism (IM)

(Or) 2) what is learning by successive refinement ?

Ans : Constructivism is a learning theory that examines the nature of learning processes. A central tenet of the constructivist approach is that learners construct new knowledge by rearranging and refining their existing knowledge. More specifically, according to the constructivist approach, new knowledge is constructed gradually, based on the learner's existing mental structures. Mental structures are developed in steps, each elaborating on preceding ones, though there may of course be regressions and blind alleys. This process is referred as learning by successive refinement.

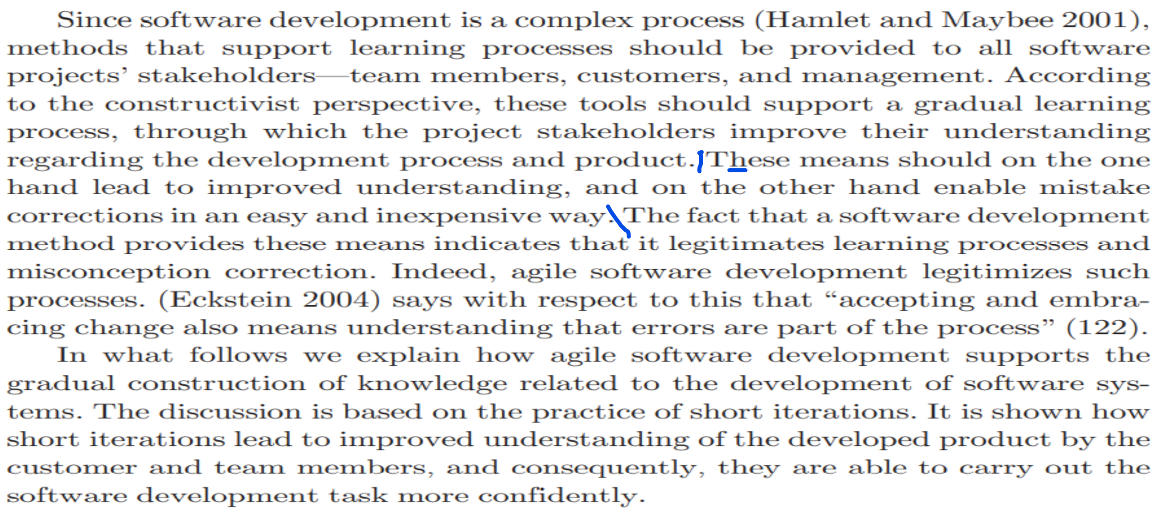
3) 
3. How agile will support learning process (Essay)

ANS :


7.4.1 Agile Software Development from the Constructivist Perspective

Constructivism has been mentioned in the second teaching and learning principle presented in Chapter 1, Introduction to Agile Software Development.

Constructivism is a learning theory that examines the nature of learning processes. A central tenet of the constructivist approach is that learners construct new knowledge by rearranging and refining their existing knowledge. More specifically, according to the constructivist approach, new knowledge is constructed *gradually*, based on the learner's existing mental structures. Mental structures are developed in steps, each elaborating on preceding ones, though there may of course be regressions and blind alleys. This process is referred to by Leron and Hazzan (1997) as "learning by *successive refinement*" and it is closely related to the Piagetian mechanisms of assimilation and accommodation (Piaget 1977). The term *successive refinement* itself is borrowed from computer science, where it refers to a methodology that guides a gradual elaboration of complex programs (Dijkstra 1972). This methodology is based on the assumption that successive refinement is an especially effective way for the human mind, with its particular strengths and limitations, to deal with complexity.


Since software development is a complex process (Hamlet and Maybee 2001), methods that support learning processes should be provided to all software projects' stakeholders—team members, customers, and management. According to the constructivist perspective, these tools should support a gradual learning process, through which the project stakeholders improve their understanding regarding the development process and product. These means should on the one hand lead to improved understanding, and on the other hand enable mistake corrections in an easy and inexpensive way. The fact that a software development method provides these means indicates that it legitimates learning processes and misconception correction. Indeed, agile software development legitimizes such processes. (Eckstein 2004) says with respect to this that "accepting and embracing change also means understanding that errors are part of the process" (122).

In what follows we explain how agile software development supports the gradual construction of knowledge related to the development of software systems. The discussion is based on the practice of short iterations. It is shown how short iterations lead to improved understanding of the developed product by the customer and team members, and consequently, they are able to carry out the software development task more confidently.

7.4.2 The Role of Short Releases and Iterations in Learning Processes

One of the main practices of agile software development processes is short releases and iterations. An iteration of an agile software development process includes all the phases and activities involved in software development processes; they are applied, however, only on a portion of the developed software product. The part on which an iteration focuses is determined by the customer, who prioritizes the development tasks according to his or her preferences.

This practice of agile software development has been mentioned several times in this book so far, especially in Chapter 3, Customers and Users. Different aspects of this practice have been highlighted in these discussions. In the context of this chapter we highlight the idea that the fact that the software is developed in short iterations and releases guides the customer, as well as the team members, in a gradual process of knowledge construction with respect to the developed software.

It is a known fact that customers face difficulties in determining in advance all of the required features of the software. Still, some software developers request customers to define their software requirements in detail at the beginning of the development process. Agile software development, however, takes a different approach: iteration and release Business Days—which include planning and reflective sessions—are conducted frequently in accordance with the practice of

From the constructivist perspective, a development process that is based on short iterations has several benefits which are directly connected to learning processes.

First, it allows both the customer and the software team members to focus on a relatively small part of the iteration.

Second, it allows the customer and the software team members to gradually improve their understanding with respect to the developed software. This is because short iterations do not require dealing with future developments that are unknown at a specific stage, and that will probably be clarified later when the development proceeds.

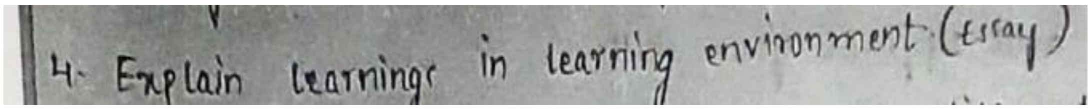
Third, short iterations improves communication between among the project stakeholders in general and between the customer and the team in particular. The Business Day, which takes place after each short iteration, and in which the customer, the team, and management participate, enables all the project stakeholders to gather together, communicate, become familiar with the others' perspectives on the project, express their concerns with respect to the development

process and the product, and reflect on previous developments. All these activities improve the understanding of the development process and the product by all the project stakeholders.

Fourth, short iterations define very clearly the time for feedback and reflective sessions. Feedback is provided by the customer at the end of each iteration; reflective sessions also take place at the end of each iteration.

Fifth, in addition to the lessons learned during such reflective sessions, such a break enables the developers to rest and detach for a while from the demanding, complex, and tight process of software development. It enables the team members to exploit their capabilities in a better way when they return to the development tasks of the next iteration.

Sixth, short iterations foster courage to raise problems and to attempt to solve them together. At the end of each iteration the team presents to the customer what has been accomplished during the last iteration, and if needed, shares with the customer any misunderstandings and/or problems in the development process.


4. Explain learnings in learning environment (Essay)

Ans : TEXT BOOK PAGE NO : 144

ELICITE COMMUNICATION

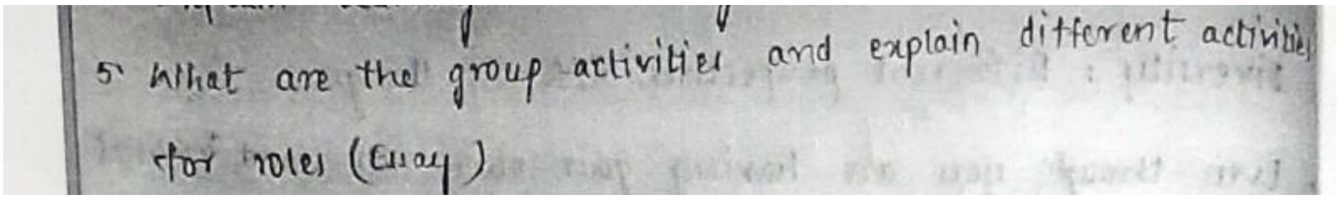
Gradual Learning Process of Agile Software Engineering

Learning and Teaching Principle:

- Elicit Reflection on Experience

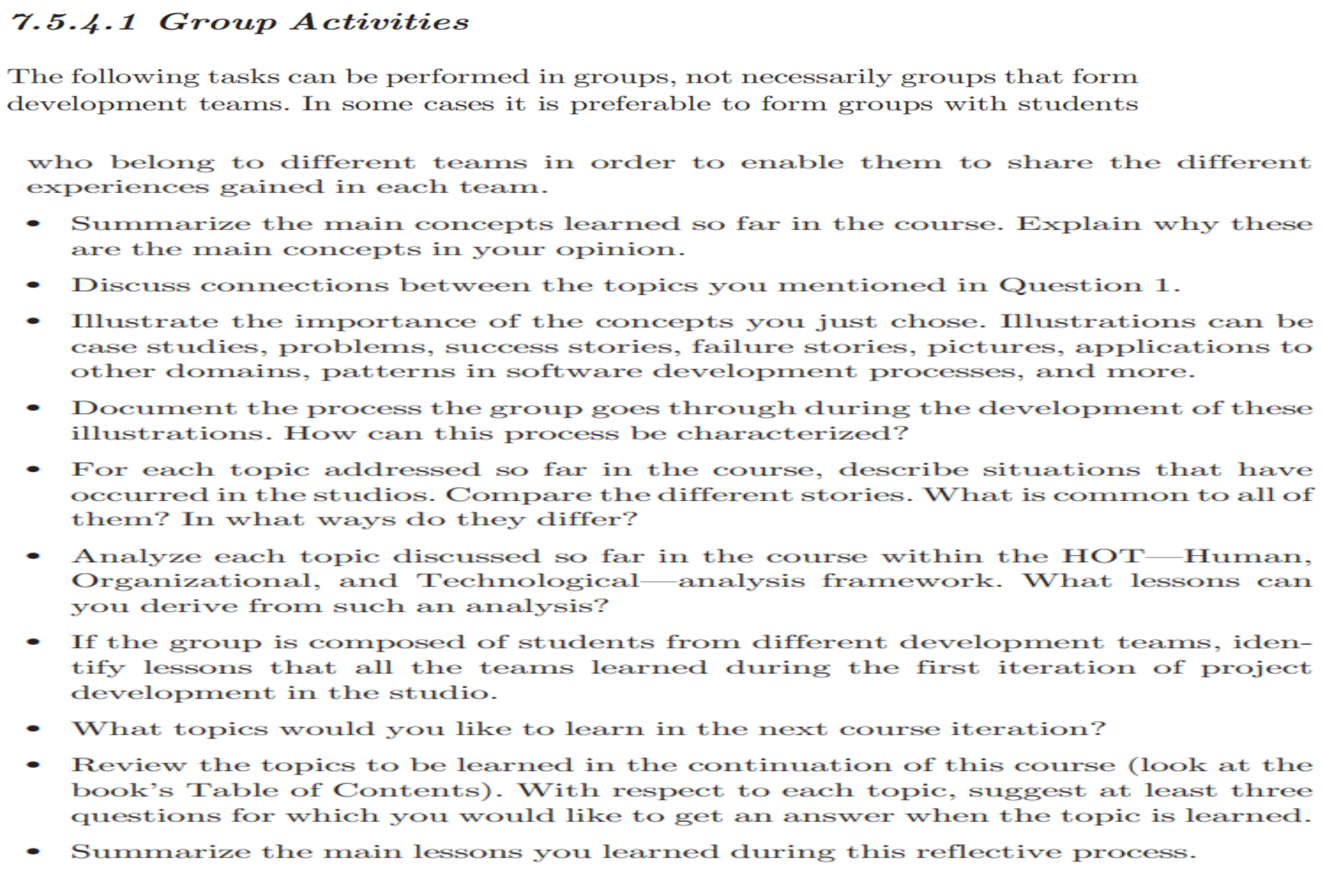
The Studio Meeting—End of the First Iteration

Intermediate Course Review and Reflection


5. What are the group activities and explain different activitie

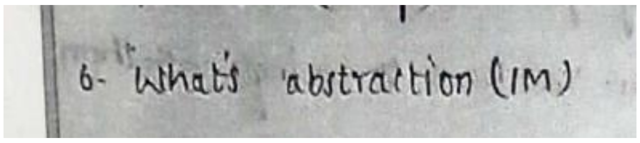
for roles (Essay)

Ans : (not much important)


7.5.4.1 Group Activities

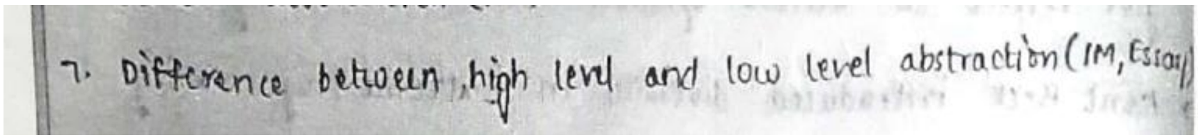
The following tasks can be performed in groups, not necessarily groups that form development teams. In some cases it is preferable to form groups with students who belong to different teams in order to enable them to share the different experiences gained in each team.

- Summarize the main concepts learned so far in the course. Explain why these are the main concepts in your opinion.
- Discuss connections between the topics you mentioned in Question 1.
- Illustrate the importance of the concepts you just chose. Illustrations can be case studies, problems, success stories, failure stories, pictures, applications to other domains, patterns in software development processes, and more.
- Document the process the group goes through during the development of these illustrations. How can this process be characterized?
- For each topic addressed so far in the course, describe situations that have occurred in the studios. Compare the different stories. What is common to all of them? In what ways do they differ?
- Analyze each topic discussed so far in the course within the HOT—Human, Organizational, and Technological—analysis framework. What lessons can you derive from such an analysis?
- If the group is composed of students from different development teams, identify lessons that all the teams learned during the first iteration of project development in the studio.
- What topics would you like to learn in the next course iteration?
- Review the topics to be learned in the continuation of this course (look at the book's Table of Contents). With respect to each topic, suggest at least three questions for which you would like to get an answer when the topic is learned.
- Summarize the main lessons you learned during this reflective process.


6. What's abstraction (1M)

Ans :

Abstraction is one means used for reducing the complexity involved in software product development. One way by which abstraction is expressed is by removing details in order to simplify and capture a concept, finding a common denominator for generalization.

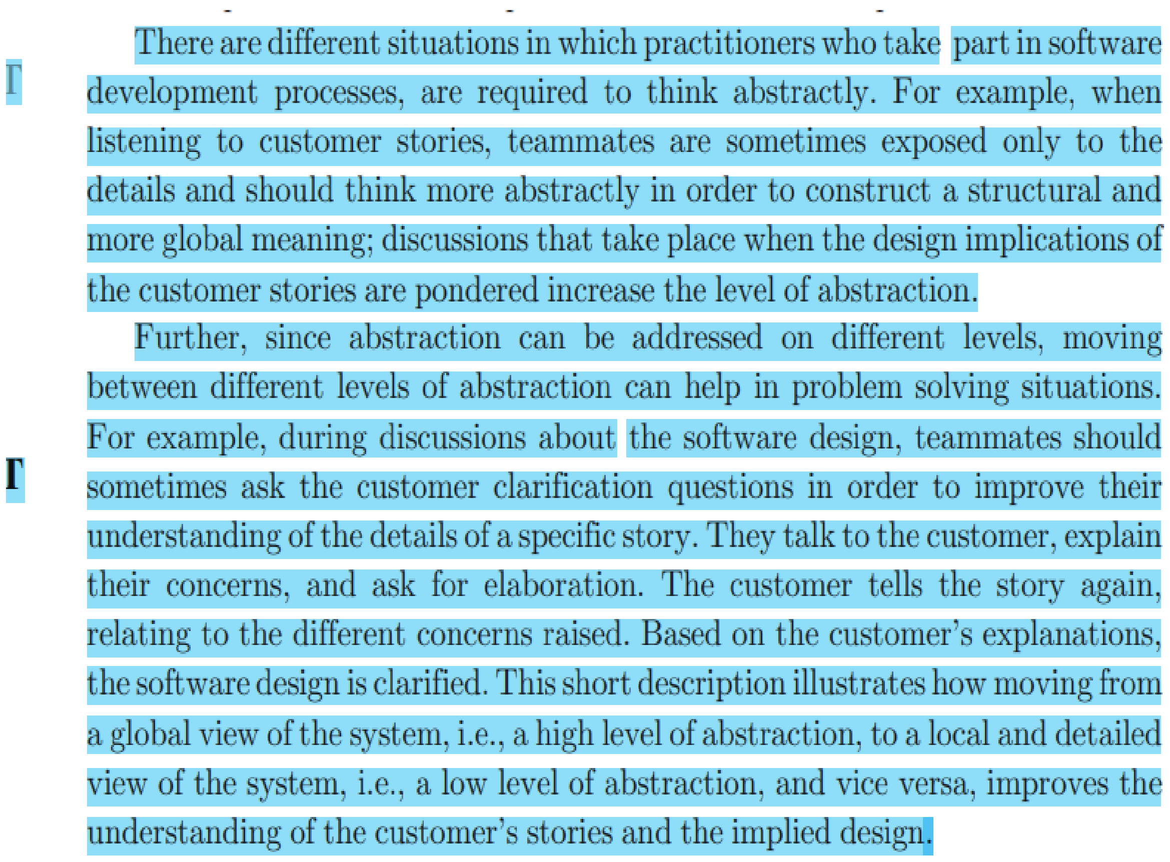

7. Difference between high level and low level abstraction (1M, Essay)

Ans :

The high level of abstraction describes about the design of the software.

The low level of abstraction describes about the refactoring of the code .

For high level


There are different situations in which practitioners who take part in software development processes, are required to think abstractly. For example, when listening to customer stories, teammates are sometimes exposed only to the details and should think more abstractly in order to construct a structural and more global meaning; discussions that take place when the design implications of the customer stories are pondered increase the level of abstraction.

Further, since abstraction can be addressed on different levels, moving between different levels of abstraction can help in problem solving situations. For example, during discussions about the software design, teammates should sometimes ask the customer clarification questions in order to improve their understanding of the details of a specific story. They talk to the customer, explain their concerns, and ask for elaboration. The customer tells the story again, relating to the different concerns raised. Based on the customer's explanations, the software design is clarified. This short description illustrates how moving from a global view of the system, i.e., a high level of abstraction, to a local and detailed view of the system, i.e., a low level of abstraction, and vice versa, improves the understanding of the customer's stories and the implied design.

Software design is carried out on a higher level of abstraction than code development. In other words, the development of a specific part of the software includes details which are eliminated when the design is constructed. Design is used to simplify communication with respect to the software and to represent the structure of the software components in an organized manner that can be understood by the different developers involved. Design can be sketched by some visual model and should be kept simple and clear.

FOR LOW LEVEL :

Refactoring (Beck 2000, Fowler 1999, Highsmith 2002) or redesign means that we improve the software design without adding functionality. Refactoring is based on the current design and it attempts to simplify it and ease future changes. This activity requires thinking at a higher abstraction level than the level of abstraction needed when dealing with the design itself.

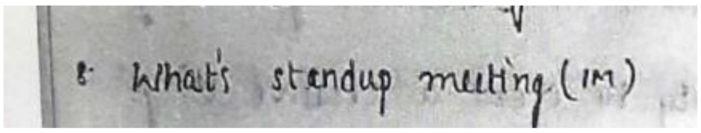
Reaching a simple design is not a simple task, and therefore refactoring is one of the practices that people find hard to accomplish. This difficulty can be explained by the need to think about the code at a higher level of abstraction than the level of abstraction on which the code was written.

Since the practice of refactoring encourages programmers to keep improving code structure and readability without adding functionality to the code, the essence of refactoring is a gradual process of code improvement. More specifically, and especially for cases in which the final "correct" structure of the code and design cannot be predicted in advance, refactoring serves as a tool that leads and supports the team members in a gradual process of code and design improvement.

The inclusion and legitimization of refactoring as part of the development method delivers a clear message—that it is acceptable to stop the development of new tasks from time to time and to allocate time for code improvement. This improvement, in turn, eases future development.

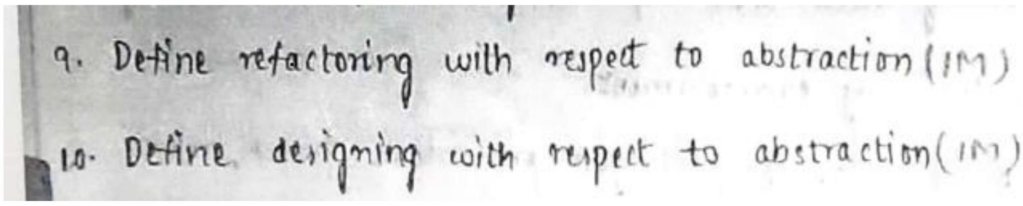
Further, in practice, when a need for an extensive refactoring is acknowledged and is agreed to by the customer in the planning session, time is allocated for refactoring, and the same activities conducted with respect to code development, such as breaking down the refactoring activity into small parts and time estimation, are conducted with respect to code refactoring.

Ongoing refactoring takes time. The return on this investment, however, is expressed in maintenance activities, in which changes are inserted in clear and easy-to-change code.



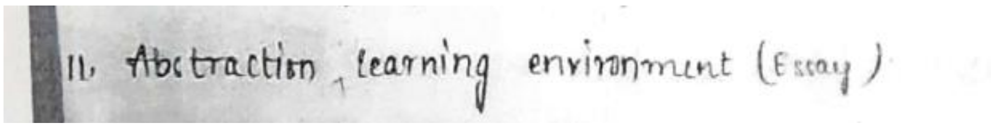
8. What's standup meeting (1M)

The stand-up meeting is conducted at the beginning of every development day (see Chapter 4, Time). It takes about ten minutes, HOT in which each teammate describes in up to one minute what he or she accomplished the day before with respect to project development, what he or she is going to perform today, and the main problems encountered, if any. The meeting goal is to share relevant infor mation about the project and to launch the development day. The coach who listens to the main problems can take care of them briefly during the stand-up meeting or, if needed, during the development day. The teammates stand during the meeting to make it short and concise.


9. Define refactoring with respect to abstraction (1M)

10. Define designing with respect to abstraction (1M)

ANS : WRITE 7 QUESTION ANSWER IN SHORT FOR 1 MARK.


11. Abstraction, learning environment (Essay)

Ans :

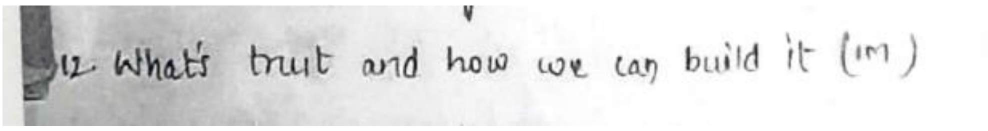
TEXT BOOK PAGE NO : 166

Elicit Communication

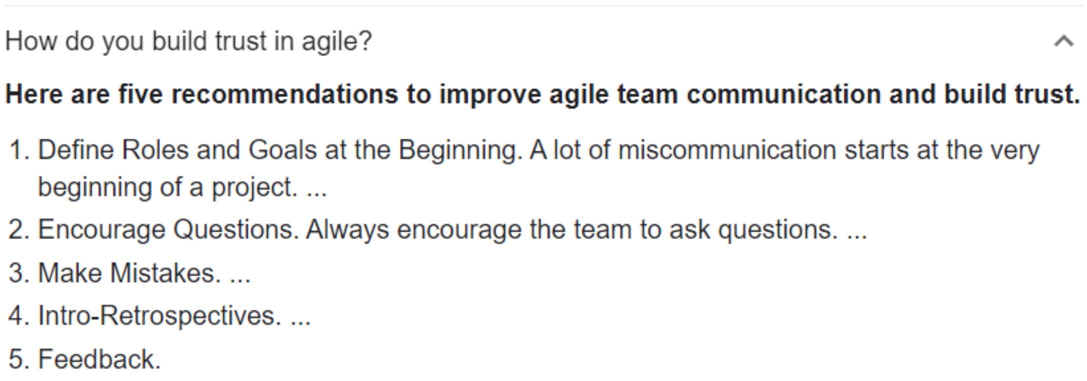
Teaching and Learning Principles :

- Be Aware of Abstraction Levels

Case Study 8.2. RefactoringActivity

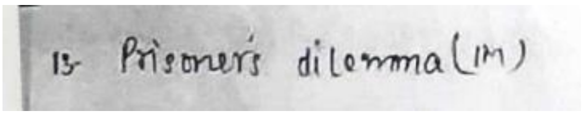

12. What's trust and how we can build it (1M)

ANS : Trust is **the bedrock of self-organizing agile teams**. Trust allows agile teams to communicate quickly and respond rapidly to changes as they emerge. Without sufficient trust, team members can waste effort and energy hoarding information, forming cliques, dodging blame, and covering their tracks.


How do you build trust in agile?

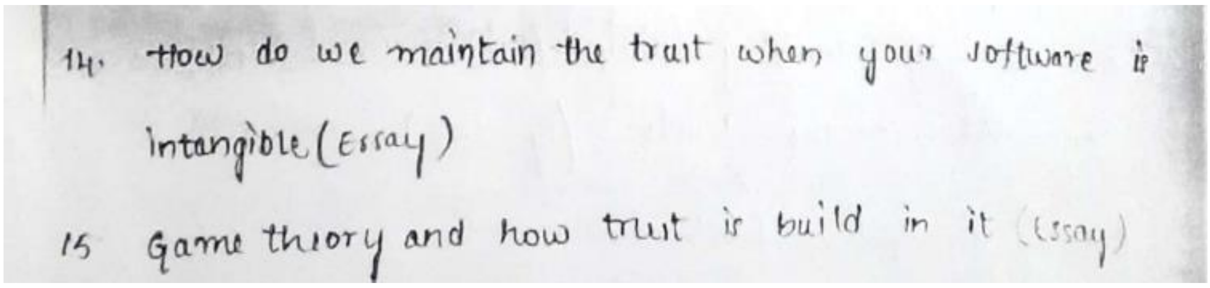
Here are five recommendations to improve agile team communication and build trust.

1. Define Roles and Goals at the Beginning. A lot of miscommunication starts at the very beginning of a project. ...
2. Encourage Questions. Always encourage the team to ask questions. ...
3. Make Mistakes. ...
4. Intro-Retrospectives. ...
5. Feedback.


13 Prisoner's dilemma (1M)

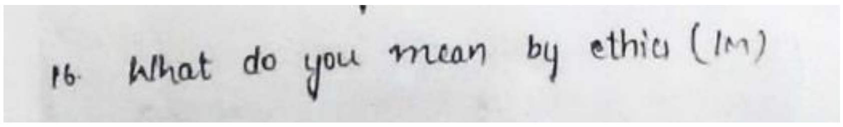
Ans :

- It is a game theory framework which illustrates how lack of trust leads people to compete with one another even in a situation where they might gain more profit by team cooperation.
- Eg: Chakde India (where Shah Rukh leads a women hockey team and figures out that there's no trust among the team and tries to teach them what it means to build a trustworthy team and how one can achieve success being in a team)


14. How do we maintain the trust when your software is intangible (Essay)

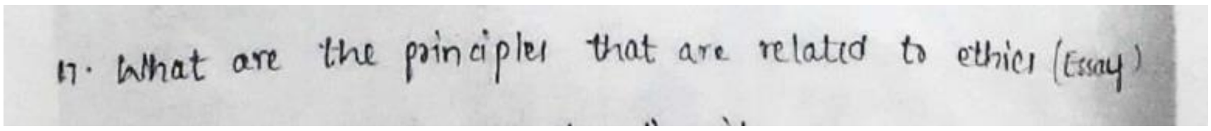
15 Game theory and how trust is build in it (Essay)

Ans : In game theory pdf


16. What do you mean by ethics (1M)

Ans :

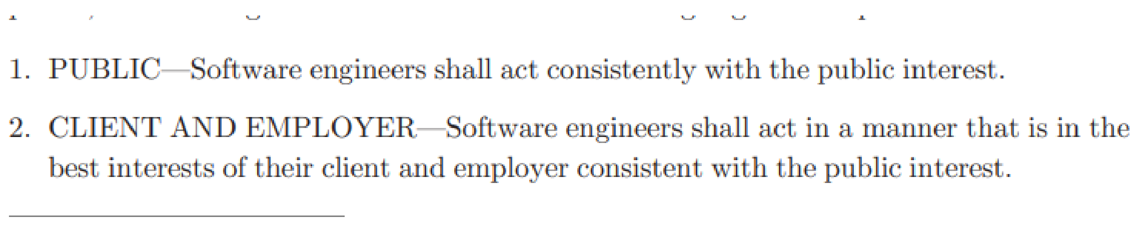
ethics guide professionals how to behave in vague situations when it is not clear what is right and what is wrong. The need for a code of ethics arises from the fact that any profession generates situations that can neither be predicted nor answered uniformly by all members of the relevant professional community.


17. What are the principles that are related to ethics (Essay)

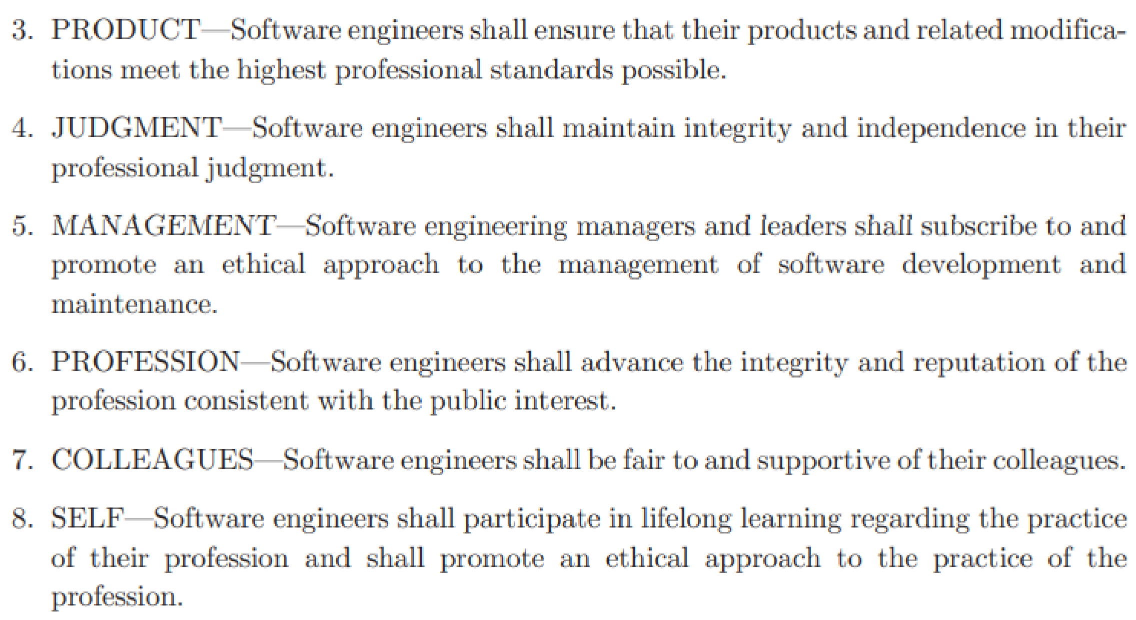
Ans :

Eight Principles:

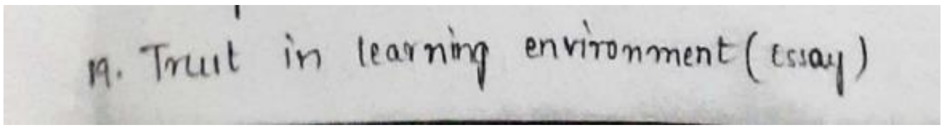
1) PUBLIC
2) CLIENT AND EMPLOYER
3) PRODUCT
4) JUDGMENT
5) MANAGEMENT|
6) PROFESSION
7) COLLEAGUES
8) SELF



1. PUBLIC—Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER—Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.



3. PRODUCT—Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT—Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT—Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION—Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES—Software engineers shall be fair to and supportive of their colleagues.
8. SELF—Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.



19. Trust in learning environment (Essay)

Ans :

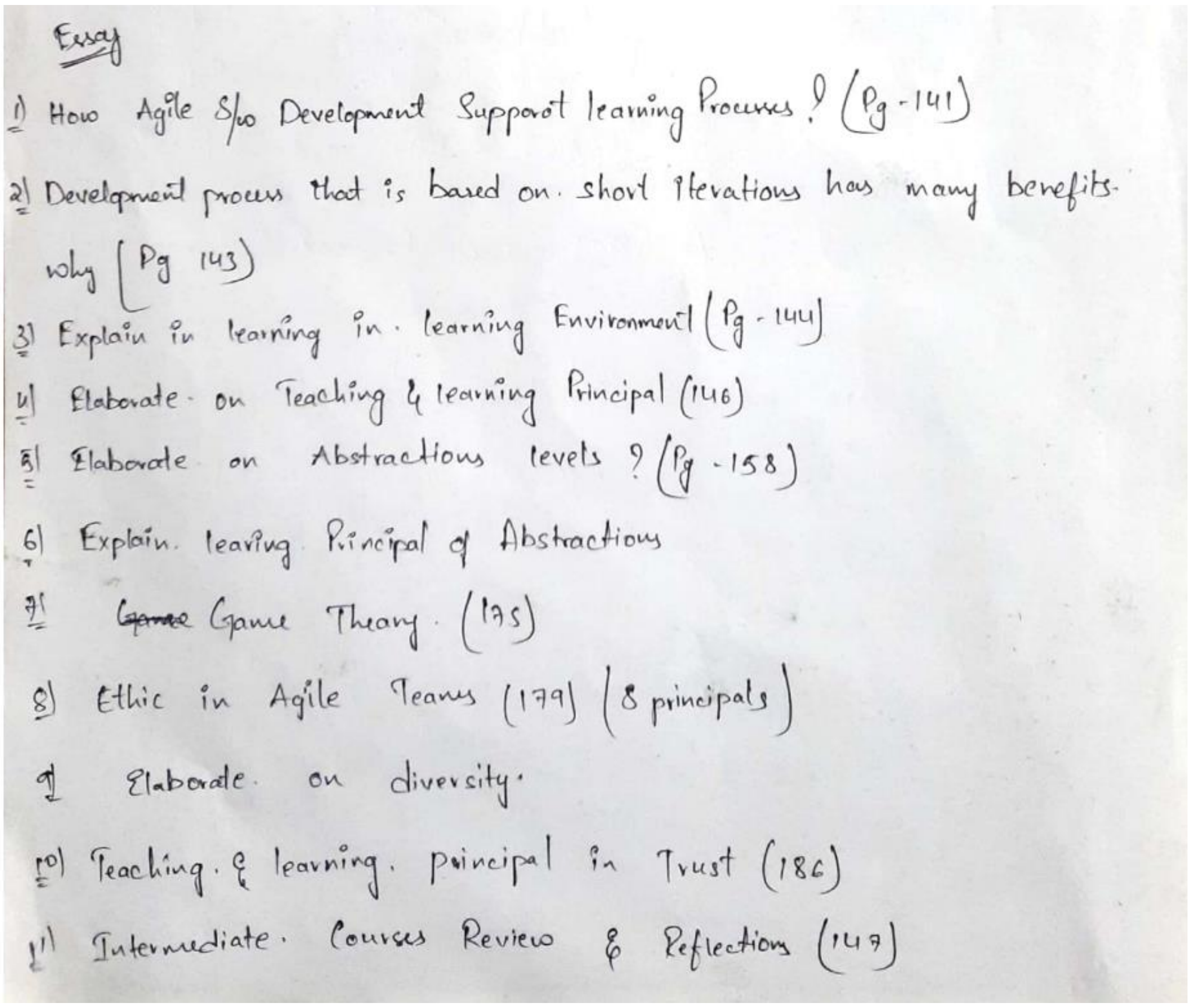
TEXT BOOK PAGE NO : 186

Teaching and Learning Principle :

Teaching and Learning Principle 6: Establish Diverse Teams

WHAT IS DIVERSITY ?

Ans :
Diversity can be expressed in different contexts, such as nationalities, HOT? worldviews, genders, minorities, cultures, and life styles. Diversity can also be expressed with respect to internal characteristics, such as hobbies, skills, thinking styles, and interests.



Essay

1) How Agile S/w Development Support learning Processes ? (Pg -141)

2) Development process that is based on. short Iterations has many benefits. why (Pg 143)

3) Explain in learning in. learning Environment (Pg - 144)

4) Elaborate. on Teaching & learning Principal (146)

5) Elaborate on Abstractions levels ? (Pg - 158)

6) Explain. leaving. Principal of Abstractions

7) ~~Game~~ Game Theary. (175)

8) Ethic in Agile Teams (179) (8 principals)

9) Elaborate. on diversity.

10) Teaching. & learning. principal in Trust (186)

11) Intermediate. Courses Review & Reflections (147)