CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test - II

Sub:	Agile Technologies							Code:	**14SCS423**
Date:	13/05/2017	Duration:	90 mins	Max Marks:	50	Sem:	IV	Branch:	Mtech(CSE)
Answer Any FIVE FULL Questions									

		Marks	OBE	
			CO	RBT
1	Describe four different team strategies for generating trust among the team members.	[10]	CO4	L1
2.	Write short notes on a. Outsourced Custom Development b. Vertical-Market Software c. Horizontal-Market Software	[10]	CO4	L2
3	What is production ready software? Also explain how to achieve Done Done.	[10]	CO4	L3
4	Describe any three ingredients for achieving nearly zero bugs.	[10]	CO4	L2
5	Describe ten minute build briefly.	[10]	CO4	L2
6	How can you manage project specific risks? Explain briefly. .	[10]	CO4	L3
7	Explain any two techniques to improve the process followed in the organization	[10]	CO3	L3

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Describe the agile software development methodology.	0	0	0	0	0	0	0	0	0	0	2	0
CO2:	Describe the XP Lifecycle, XP Concepts, Adopting XP	0	0	0	0	0	0	0	1	0	0	1	0
CO3:	Explain the different aspects of Mastering agility.	0	0	0	0	0	1	0	1	0	0	0	0
CO4:	Apply the various practices of Thinking, Collaborating, Releasing, Planning and Development in organization.	2	0	0	0	3	3	0	2	0	0	0	0

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 – *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*

#132, AECS Layout, IT Park Road, Kundalahalli, Bangalore – 560 037
T:+9180 28524466 / 77

CMR INSTITUTE
OF TECHNOLOGY

Scheme and Solution – (IAT 2)

Department of Computer Science and Engineering

Agile Technologies (14SCS423)

Scheme

Q.1 Explanation on Customer-Programmer Empathy(3 marks)

Explanation on Programmer-Tester Empathy (3 marks)

Explanation on Eat together (2 marks)

Explanation on Team Continuity (2 marks)

Q.2Explanation on Outsourced Custom Development(4 marks)

Explanation on Vertical-Market Software (3 marks)

Explanation on Horizontal-Market Software (3 marks)

Q.3Explain production ready software(4 marks)

Explain how to achieve done done (6 marks)

Q.4Explanation on any three ingredients for achieving nearly zero bugs (10 marks).

Q.5Explanation on Automating Build(2 marks)

Explanation on how to automate (2 marks)

Explanation on when to automate (2 marks)

Explanation on Automating Legacy Projects (2 marks)

Explanation on Ten minutes or less (2 marks)

Q.6 Detail explanation on managing project risk(10 marks)

Q.7 Explanation on any two techniques for improving the process where each technique carries 5 marks.

<u>Solution</u>

1. **Describe four different team strategies for generating trust among the team members.**

The team must take joint responsibility for their work. Team members need to think of the rest of the team as “us,” not “them.”Trust is essential for the team to perform this well. You need to trust that taking time to help others won’t make you look unproductive. You need to trust that you’ll be treated with respect when you ask for help or disagree with someone. The organization needs to trust the team, too.

Team Strategy #1: Customer-Programmer Empathy

- Customers often feel that programmers don’t care enough about their needs and deadlines, some of which, if missed, could cost them their jobs. Programmers often feel forced into commitments they can’t meet, hurting their health and relationships.
- Sometimes the acrimony is so intense that the groups actually start doing what the others fear: programmers react by inflating estimates and focusing on technical toys at the expense of necessary features; customers react by ignoring programmer estimates and applying schedule pressure. This sometimes happens even when there’s no overt face-to-face hostility.
- Sitting together is the most effective way to build empathy. Each group gets to see that the others are working just as hard. Retrospectives also help, if your team can avoid placing blame. Programmers can help by being respectful of customer goals, and customers can help by being respectful of programmer estimates and technical recommendations. All of this is easier with energized work.

Team Strategy #2: Programmer-Tester Empathy

- There is “us versus them” attitudes between programmers and testers, although it isn’t quite as prevalent as customer-programmer discord.
- When it occurs, programmers tend not to show respect for the testers’ abilities, and testers see their mission as shooting down the programmers’ work.
- Programmers, remember that testing takes skill and careful work, just as programming does.
- Take advantage of testers’ abilities to find mistakes you would never consider, and thank them for helping prevent embarrassing problems from reaching stakeholders and users.
- Testers, focus on the team’s joint goal: releasing a great product.

Team Strategy #3: Eat Together

- Another good way to improve team cohesiveness isto eat together. Something about sharing meals breaks down barriers and fosters team cohesiveness. Try providing a free meal once per week.

- If you have the meal brought into the office, set a table and serve the food family-style to prevent people from taking the food back to their desks.

Team Strategy #4: Team Continuity

- After a project ends, the team typically breaks up. All the wonderful trust and cohesivenessthat the team has formed is lost. The next project starts with a brand-new team, and they haveto struggle through the four phases of team formation all over again.
- You can avoid this waste by keeping productive teams together. Most organizations think ofpeople as the basic "resource" in the company. Instead, think of the team as the resource.
- Rather than assigning people to projects, assign a team to a project. Have people join teamsand stick together for multiple projects.
- Some teams will be more effective than others. Take advantage of this by using the mosteffective teams as a training ground for other teams.

2. **Write short notes on**
   a. **Outsourced Custom Development**
   b. **Vertical-Market Software**
   c. **Horizontal-Market Software**

Outsourced Custom Development

- Outsourced custom development is similar to in-house development, but you may not have the connections that an in-house team does. As a result, you may not be able to recruit realcustomers to act as the team's on-site customers.
- One way to recruit real customers is to move your team to your customer'soffices rather than asking them to join you at yours.
- If you can't bring real customers onto the team, make an extra effort to involvethem. Meet in person with your real customers for the first week or two ofthe project so you can discuss the project vision and initial release plan.
- Ifyou're located near each other, meet again for each iteration demo retrospective, and planning session.If you're far enough apart that regular visits aren't feasible, stay in touch via instant messagingand phone conferences.

Vertical Market Software

- Vertical-market software is developed for many organizations.Like custom development, however, it's built for a particular industry and it's often customizedfor each customer.
- Because vertical-market software has multiplecustomers, each with customized needs, you haveto be careful about giving real customers too muchcontrol over

the direction of the product. You couldend up making a product that, while fitting your onsitecustomer’s needs perfectly, alienates yourremaining customers.

- Instead, your organization should appoint a product manager who understands the needs of your real customers impeccably.
- Rather than involving real customers as members of the team, createopportunities to solicit their feedback. Some companies create a customer review board filledwith their most important customers. They share their release plans with these customers and—on a rotating basis—provide installable iteration demo releases for customers to try.

Horizontal-Market Software

- Horizontal-market software is the visible tip of the software development iceberg: softwarethat’s intended to be used across a wide range of industries.
- As with vertical-market software, it’s probably better to set limits on the control that realcustomers have over the direction of horizontal-market software. Horizontal-market softwareneeds to appeal to a wide audience, and real customers aren’t likely to have that perspective.
- Again, an in-house product manager who creates a compelling vision based on all customers’needs is a better choice.
- As a horizontal-market developer, your organization may not have the close ties withcustomers that vertical-market developers do. Thus, a customer review board may not be agood option for you. Instead, find other ways to involve customers: focus groups, userexperience testing, community previews, beta releases, and so forth.

3. What is production ready software? Also explain how to achieve Done Done.

- Done-Done means the story or feature should be completely finished with integration, testing and ready to be deployed at the end of iteration.
- Partially finished stories result in hidden costs and destabilizes the release planning.

Story or feature is done-done only when it satisfies this criteria

- Tested (all unit, integration, and customer tests finished)
- Coded (all code written)
- Designed (code refactored to the team’s satisfaction)
- Integrated (the story works from end to end—typically, UI to database—and fits into therest of the software)
- Builds (the build script includes any new modules)

- Installs (the build script includes the story in the automated installer)
- Migrates (the build script updates database schema if necessary; the installer migrates datawhen appropriate)
- Reviewed (customers have reviewed the story and confirmed that it meets theirexpectations)
- Fixed (all known bugs have been fixed or scheduled as their own stories)
- Accepted (customers agree that the story is finished)

How to be Done-Done

- Make a little progress on every aspect of your work every day.
- Use TDD.
- Integrate the code with rest of the system(CI)
- Use ten minute build and have a separate team for updating the installer.
- Include the onsite customers and show them the look of UI and get feedback.
- Integrate various pieces and run the software to make sure the pieces all work together.
- Conduct exploratory testing.
- Fix the mistakes, bugs and errors.
- Improve work habits.
- When you believe the story is "done done," show it to your customers for final acceptance review.

4. **Describe any three ingredients for achieving nearly zero bugs.**

Ingredient # 1: Write fewer bugs

- Start with TDD- to reduce the number of defects.
- Work sensible hours(i.e. energized work)
- Use pair programming- Program all production code in pairs. This improves your brainpower which helps you make less mistakes and find them quickly.
- Allow onsite customers to sit together in your team.

- Use customer tests to help communicate complicated domain rules.
- Testers work with customer to find and fix gaps in their approach to requirements.
- Demonstrate software to stakeholders every week and act on their feedback.
- Good coding standards, done-done checklist will help to avoid common mistakes.

Ingredient # 2: Eliminate bug breeding ground

- **Technical debt-breeds bugs.**Hence pay down your technical debt to generate few defects or bugs
- Hence to avoid or eliminate the bug breeding groundsPay down the technical debt of old code in iteration slack.
- Create clean code by following simple design.Both code and design should be refactored.

Ingredient # 3: Fix bugs now

- Fix bugs quickly- To reduce hidden cost, future problems, defects in future.
- After fixing the bug identifyWhy did that bug occur?Is there a design flaw that made this bug possible?Is there some way to refactor the code that would make this kind of bug less likely?If you identify systemic problem-please discuss it in stand up meeting or retrospective.
- Fixing bug requires all team members to sit together.
- Use collective code ownership so that any pair can fix a buggy module.
- Bug is too large
  - ➢ Write it in story card.
  - ➢ Identify tasks for this story card
  - ➢ If there is enough slack fix the bug
  - ➢ Else ask your product manager to decide whether to fix it in this release or next iteration.

5. Describe ten minute build briefly.

Automate Build

- Building the whole project is often a frustrating and lengthy experience. This frustration can spill over to the rest of your work.

- Hence automating your build is one of the easiest ways to improve morale and increase productivity.

How to automate

- **Using build tools**
  - Java-Ant Tool
  - .Net- Nant and MSBuild
  - C and C++- Make
- **The build tool should be able to do following without human intervention.**
  - Compile code and run tests
  - configure registry settings
  - initialize database schemas
  - Set up web servers.
  - ability to create a production release
- **Local Build:**You should be able to build even when disconnected from the network. A local build will allow you to build and test at any time without worrying about other people’s activities.
- Instead of using the generic script generated by IDE **create completely new script.**

When to automate

- At the start of first iteration
- First iteration: involves delivering the working model of simple product.
- Because the product is so small and simple at this stage, creating a high-quality automated build is easy.
- In every iteration, as you add features that require more out of your build, improve your build script.
- Use the build script to configure the integration machine. Don’t configure it manually.

Automating Legacy Projects

- Creating automated build script for existing project requires some time , but surely it improves your life.
- Existing project has many components. Choose the most error-prone or frustrating component for creating the automated build first.
- First the build script should be able to compile the component.
- Next, add the ability to run unit tests and make sure they pass.
- Next, add the ability to handle some configuration settings automatically.
- Next, add the ability to either create a release package such as an installer or deploying directly to the production servers

Ten minutes or less

- Build is two-step process
  - On local machine
  - On integration machine
- A 10-minute build leads to a 20-minute integration cycle. That's a pretty long delay.
- The easiest way to keep the build under 5 minutes (with a 10-minute maximum) is to keep the build times down from the beginning.
- Slow tests are the most common cause of slow builds.
- You should be able to run about 100 unit tests per second. Unit tests should comprise the majority of your tests. A fraction (less than 10 percent) should be integration tests, which checks that two components synchronize properly.
- if compilation speed becomes a problem, consider optimizing code layout or using a compilation cache or incremental.
- You could also use a distributed compilation system or take the best machine available for use as the build master.

6. How can you manage project specific risks? Explain briefly.

- Create a risk census—that is, a list of the risks your project faces that focuses on your project's unique risks. Gather the whole team and hand out index cards. Remind team members that during this exercise, negative thinking is not only OK, it's necessary. Ask them to consider ways in which the project could fail. Write several questions on the board.

- What about the project keeps you up at night?
- Imagine it's a year after the project's disastrous failure and you're being interviewed about what went wrong. What happened?
- Imagine your best dreams for the project, then write down the opposite.
- How could the project fail without anyone being at fault?
- How could the project fail if it were the stakeholders' faults? The customers' faults? Testers? Programmers? Management? Your fault? Etc.
- How could the project succeed but leave one specific stakeholder unsatisfied or angry?

- Once you have your list of catastrophes, brainstorm scenarios that could lead to those catastrophes. From those scenarios, imagine possible root causes. These root causes are your risks: the causes of scenarios that will lead to catastrophic results.
- After you've finished brainstorming risks, let the rest of the team return to their iteration while you consider the risks within a smaller group. (Include a cross-section of the team.) For each risk, determine: Estimated probability—I prefer "high," "medium," and "low."Specific impact to project if it occurs—dollars lost, days delayed, and project cancellation are common possibilities.
- For the risks you decide to handle, determine transition indicators, mitigation and contingency activities, and your risk exposure:
- Transition indicators tell you when the risk will come true. It's human nature to downplay upcoming risks, so choose indicators that are objective rather than subjective. For example, if your risk is "unexpected popularity causes extended downtime," then your transition indicator might be "server utilization trend shows upcoming utilization over 80percent.
- Mitigation activities reduce the impact of the risk. Mitigation happens in advance, regardless of whether the risk comes to pass. Create stories for them and add them to your release plan. To continue the example, possible stories include "support horizontal scalability" and "prepare load balancer."
- Contingency activities also reduce the impact of the risk, but they are only necessary if the risk occurs. They often depend on mitigation activities that you perform in advance. For example, "purchase more bandwidth from ISP," "install load balancer," and "purchase and prepare additional frontend servers."
- Risk exposure reflects how much time or money you should set aside to contain the risk. To calculate this, first estimate the numerical probability of the risk and then multiply that by the impact. When considering your impact, remember that you will have already paid for mitigation activities, but contingency activities are part of the impact. For example, you might believe that downtime due to popularity is 35 percent likely, and the impact is three days of additional programmer time and $20,000 for bandwidth, collocation fees, and new equipment. Your total risk exposure is $7,000 and one day.

7. Explain any two techniques to improve the process followed in the organization.

Understand your project

- To improve the process understands how it affects your project.
- What works well and what doesn't- Find it through the feedback.
- Always ask why
    - Why do we follow this practice?
    - Why is this practice working?
    - Why isn't this practice working?
    - Encourage open discussion among team members.

In practice

- Feedback Loops: Helps to improve work
- Root-cause analysis, retrospectives: improve the team's understanding.
- Stand-up meetings, informative workspace: contribute to an information-rich environment.
- Energized work and slack: reduce work pressure.
- Pair programming: gives one person in each pair time to think about strategy.

Tune and adapt

- Whenever some change is needed modify the process.
- Changes require tuning. Tuning is adopted by doing small, isolated changes (i.e. experiment).
- Observe the results according to your expectation. Iterate until you're satisfied with the results.
- Team should be flexible, adaptive and needs to have the courage to experiment and occasionally fail.
- New agile teams lack experience and hence should be cautious about changing their process.
- The feedback about your changes helps to improve your process and your understanding of agility.

In practice

- Tuning and Adapting is achieved with the help of retrospective. Retrospective is an explicit practice.
- But the Courage to Adapt any time is important principle and is based on XP's value Courage.