COURSE FILE

MODERN SOFTWARE ENGINEERING

(Subject Codes (R 16 B.Tech): 138DK)

IV Year - II Sem B.TECH. (IT)

Submitted to

DEPARTMENT OF INFORMATION TECHNOLOGY

BY

Mr. Qazi Basheer, Associate Professor

NAWAB SHAH ALAM KHAN COLLEGE OF ENGINEERING AND TECHNOLOGY
New Malakpet, Hyderabad, Telangana- 500024
(Affiliated to JNTUH, Approved by AICTE, NEWDELHI) http://www.nsakcet.ac.in/
2020 – 2021

NAWAB SHAH ALAM KHAN COLLEGE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

(Name of the Subject/Lab Course): MODERN SOFTWARE ENGINEERING

(JNTU CODE: 138DK **Program: UG**

Branch: IT **Version No: 1**

Year: I **Document Number : NSAKCET/IT/MSE/01**

Semester: I **No. of Pages:**

Classification status (Unrestricted/Restricted) : Distribution List:

Prepared by :

1) Name : Qazi. M. A. Basheer
2) Sign :___________________
3) Design :- Associate Professor
4) Date :__________________________

Updated by :

1) Name :
2) Sign :
3) Design :
4) Date :

Verified by:

1) Name :
2) Sign :
3) Designation :
4) Date :

***For Q.C only**

1)Name :
2) Sign :
3) Design :
4) Date :

Approved by (HOD) :

1) Name: Dr. G. S. S. Rao
2) Sign :
3) Date :

Contents

2. Syllabus Copy

MODERN SOFTWARE ENGINEERING

B.Tech. IV Year II Sem.

	L T P C
Course Code: CS854PE	3 0 0 3

UNIT - I

Introduction Extreme Programming (XP) - Agile Development

Why Agile - Understanding Success, Beyond Deadlines, Importance of Organizational Success, Introduction to Agility, How to Be Agile - Agile methods, Don't make your own method, Road to mastery, Understanding XP (Extreme Programming) - XP life cycle, XP team, XP Concepts, Adopting XP - Knowing whether XP is suitable, Implementing XP, assessing Agility, Practicing XP - Thinking - Pair Programming, Energized work, Informative Workspace, Root cause Analysis, Retrospectives

UNIT - II

Collaborating: Trust, Sit together, Real customer involvement, Ubiquitous language, meetings, coding standards, Iteration demo, Reporting

UNIT – III

Releasing: Bugfree Release, Version Control, fast build, continuous integration, Collective ownership, Documentation

UNIT - IV

Planing: Version, Release Plan, Risk Management, Iteration Planning, Slack, Stories, Estimating

UNIT - V

Developing: Incremental requirements, Customer tests, Test driven development, Refactoring, Incremental design and architecture, spike solutions, Performance optimization, Exploratory testing

TEXT BOOK:

1. The art of Agile Development, James Shore and Shane Warden, 11th Indian Reprint,
O'Reilly, 2018

REFERENCES:

1. Learning Agile, Andrew Stellman and Jennifer Greene, O’Reilly, 4th Indian Reprint, 2018
2. Practices of an Agile Developer, Venkat Subramaniam and Andy Hunt, SPD, 5th Indian Reprint, 2015
3. Agile Project Management - Jim Highsmith, Pearson Low price Edition 2004

3. Vision & Mission of the Institute

3. Vision and Missions of the Institution

Vision of the Institution:
To impart quality technical education with strong ethics, producing technically sound engineers capable of serving the society and the nation in a responsible manner.

Mission of the Institution:
M1: To provide adequate knowledge encompassing strong technical concepts and soft skills thereby inculcating sound ethics.

M2: To provide a conducive environment to nurture creativity in teaching- learning process.

M3: To identify and provide facilities which create opportunities for deserving students of all communities to excel in their chosen fields.

M4: To strive and contribute to the needs of the society and the nation by applying advanced engineering and technical concepts.

4. Vision and Mission of the Department

4. Vision and Missions of the Department

Vision of the department: To produce quality IT professionals, with an ability to adapt to ever changing IT needs of local, national and international arena, through effective teaching & learning, interactions with alumni and industry.

Mission No.	Mission Statements
M1	To provide a holistic learning environment for students through ethical practices.
M2	To provide quality infrastructure through practical exposure to the latest technology requirements.
M3	To train the students in soft skills to excel in placements and competitive exams at higher level the industry ready.
M4	To have a healthy Industry - Institute interaction through faculty development programs, student internships, guest lectures and using latest teaching learning methodologies.
M5	To provide effective platform to meet the industrial requirement and provide research oriented environment for the faculty to meet the continuous societal needs.

5. Pos and PSOs

5. Program Outcomes and Program Specific Outcomes

List of Program Outcomes	
PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
PO5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
PO6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities
PO9	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary fields
PO12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological
PSO1	Develop efficient information management systems using latest development tools catering to the globally changing requirements in multi-disciplinary domains.
PSO2	Manage real time IT projects with consideration of human, financial, ethical and environmental factors and an understanding of policy implications.

6. Course Objectives and Course Outcomes

6. Course Objectives and Course Outcomes

Course Objectives:

1. Learn and identify the theoretical and methodological issues involved in modern software engineering
2. To be able to understand Extreme Programming (XP) basics and program design with functions using XP.
3. To understand a range of adopting agile programming development, as well as the study and development techniques.
4. To understand the high-performance XP designed to strengthen the practical expertise.
5. Develop software projects based on current technologies, by managing resources economically and keeping ethical values.

Course Outcomes:

After completing this course, the student must demonstrate the knowledge and ability to:

1. Examine the importance of agile development and the basics of XP.
2. Analyze and apply the collaborating methods of Agile Software Development.
3. Analyze and use the Bug Free Development of the Software and Release.
4. Illustrate the mechanisms of adopting and implementing the Agile Software.
5. Develop the software according to the customer requirements and expectations by managing resources economically and keeping ethical values.

7. CO PO Mapping with Venn Diagrams

7. PO and CO Mappings

After completing this course the student must demonstrate the knowledge and ability to:	
CO1	examine the importance of agile development and the basics of XP,
CO2	analyze and apply the collaborating methods of Agile Software Development.
CO3	Analyze and use the Bug Free Development of the Software and Release.
CO4	illustrate the mechanisms of adopting and implementing the Agile Software.
CO5	develop the software according to the customer requirements and expectations by managing resources economically and keeping ethical values.

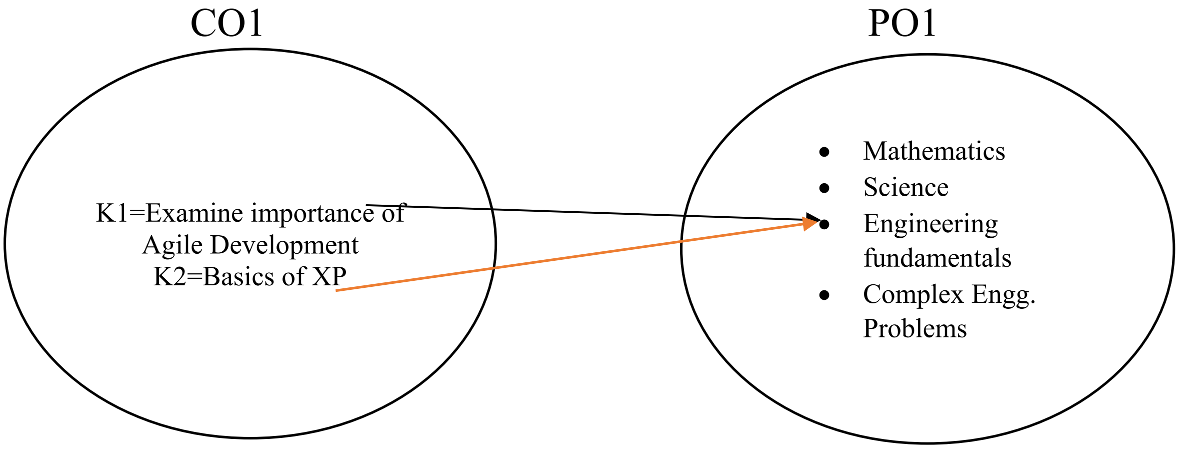
COs AND POs Mapping

Cos/POs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	3													
CO2	3	3	3		3				3	3		3		
CO3	3	3	1		3				3	3		3		
CO4			3		3				3	3		3	3	3

Probability (CO# to PO#) =

< 0.25	- No Correlation
> 0.25 and <= 0.50	- 1
>0.50 and <= 0.75	- 2
>0.75 and <= 1.00	- 3

8. CO and PO Venn Diagrams



CO1

K1=Examine importance of Agile Development

K2=Basics of XP

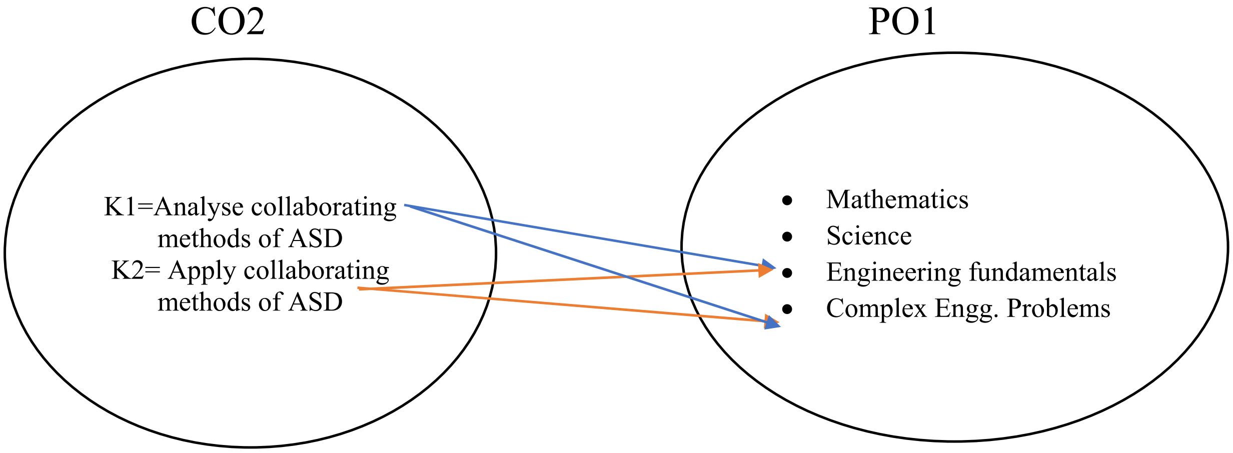
PO1

- Mathematics
- Science
- Engineering fundamentals
- Complex Engg. Problems

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

Correlation - CO1 to PO1 = 3



CO2

K1=Analyse collaborating methods of ASD

K2= Apply collaborating methods of ASD

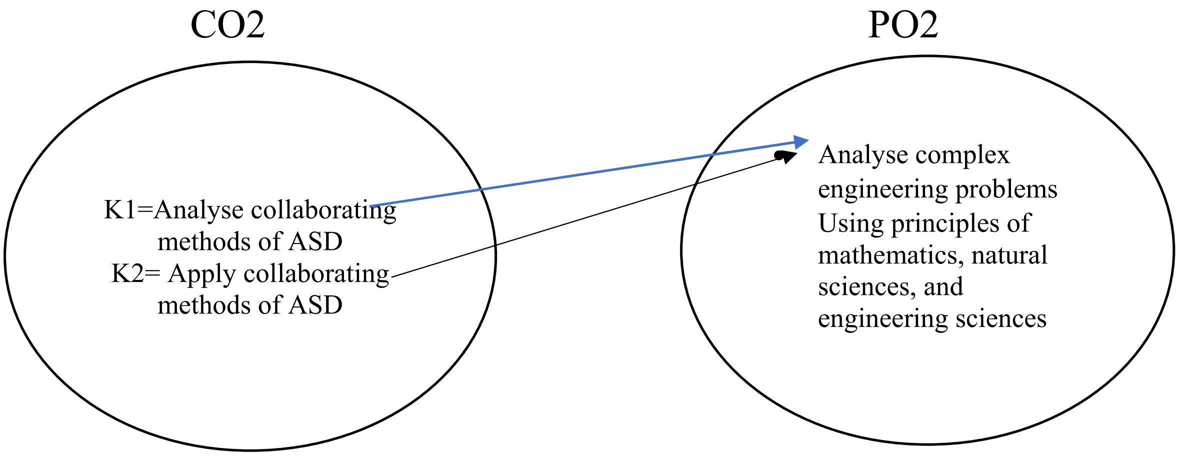
PO1

- Mathematics
- Science
- Engineering fundamentals
- Complex Engg. Problems

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

Correlation – CO2 to PO1 = 3



CO2

PO2

K1=Analyse collaborating methods of ASD

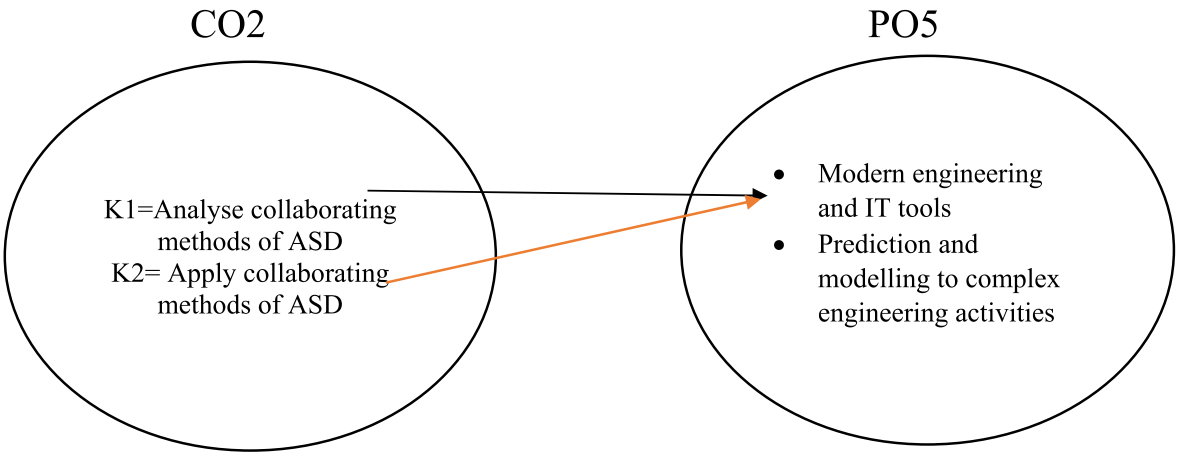
K2= Apply collaborating methods of ASD

Analyse complex engineering problems Using principles of mathematics, natural sciences, and engineering sciences

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 1/2=1$

Correlation – CO2 to PO2 = 3



CO2

PO5

K1=Analyse collaborating methods of ASD

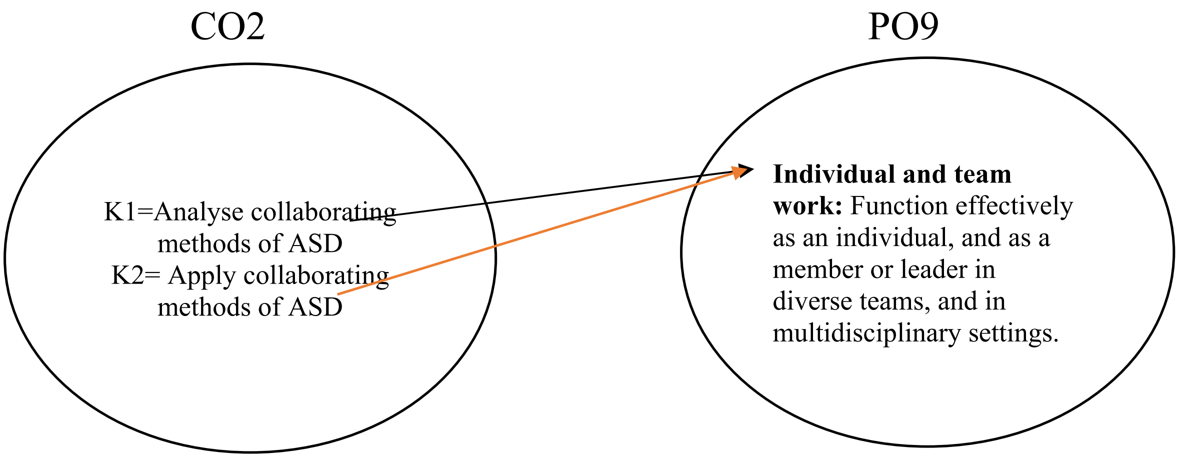
K2= Apply collaborating methods of ASD

- Modern engineering and IT tools
- Prediction and modelling to complex engineering activities

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

Correlation – CO2 to PO5 = 3



CO2

PO9

K1=Analyse collaborating methods of ASD

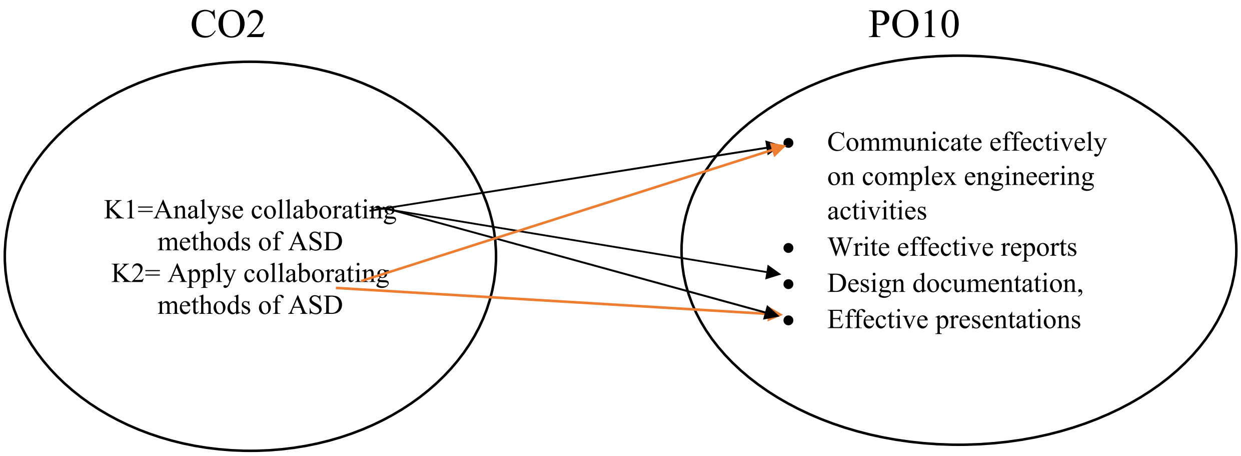
K2= Apply collaborating methods of ASD

Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

Correlation – CO2 to PO9 = 3



CO2

PO10

K1=Analyse collaborating methods of ASD

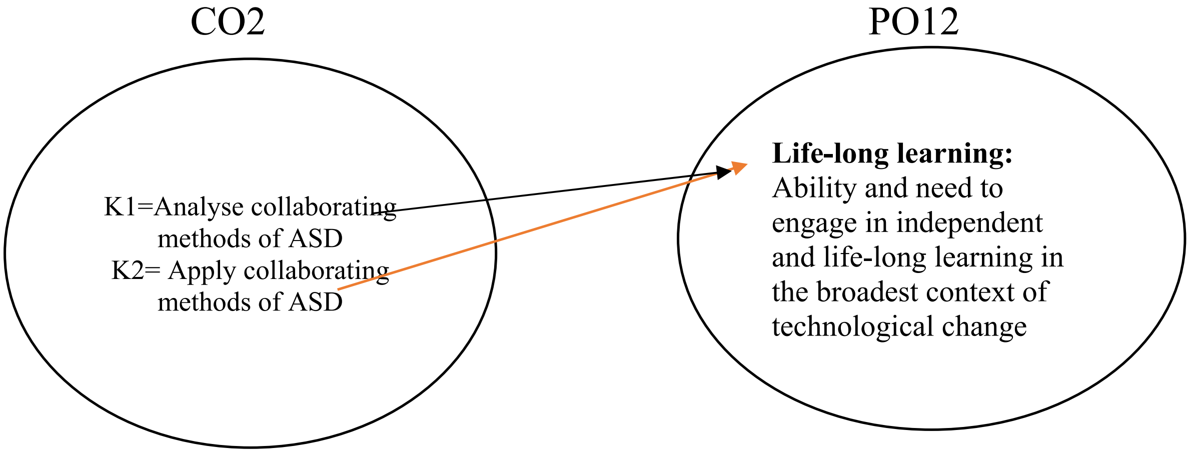
K2= Apply collaborating methods of ASD

- Communicate effectively on complex engineering activities
- Write effective reports
- Design documentation,
- Effective presentations

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

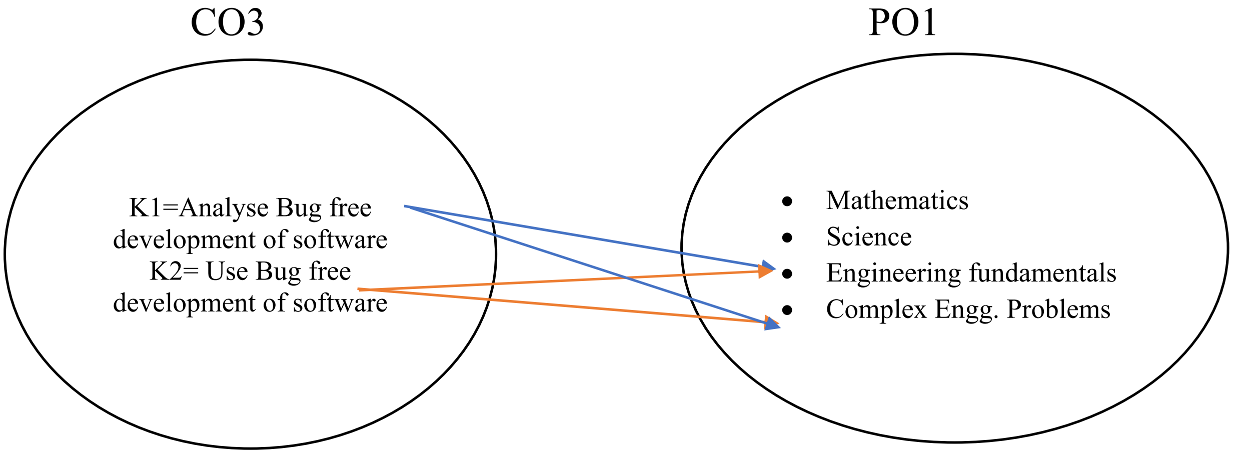
Correlation - CO1 to PO1 = 3


CO2
PO12
K1=Analyse collaborating methods of ASD
K2= Apply collaborating methods of ASD
Life-long learning: Ability and need to engage in independent and life-long learning in the broadest context of technological change

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

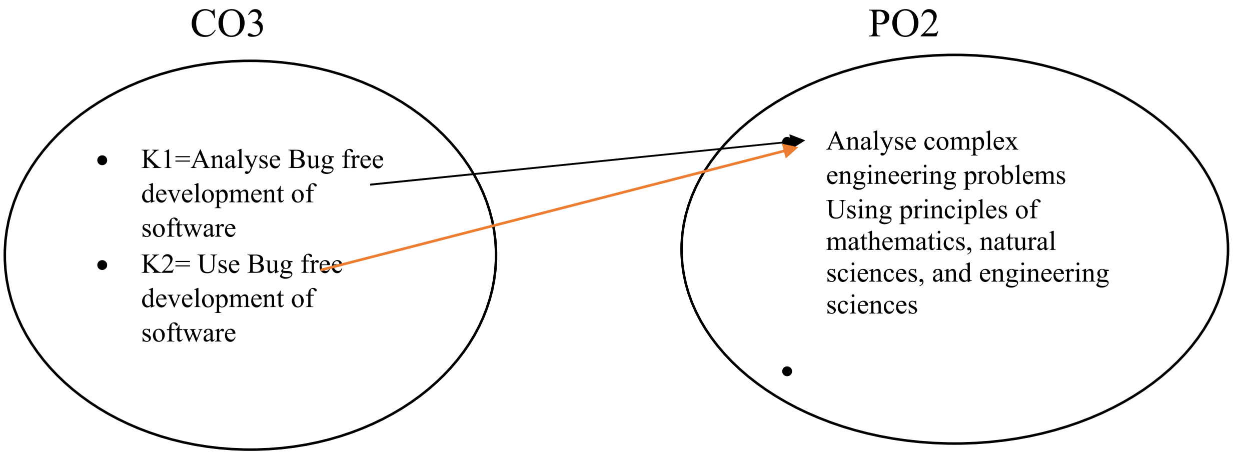
Correlation – CO2 to PO12 = 3


CO3
PO1
K1=Analyse Bug free development of software
K2= Use Bug free development of software
Mathematics
Science
Engineering fundamentals
Complex Engg. Problems

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

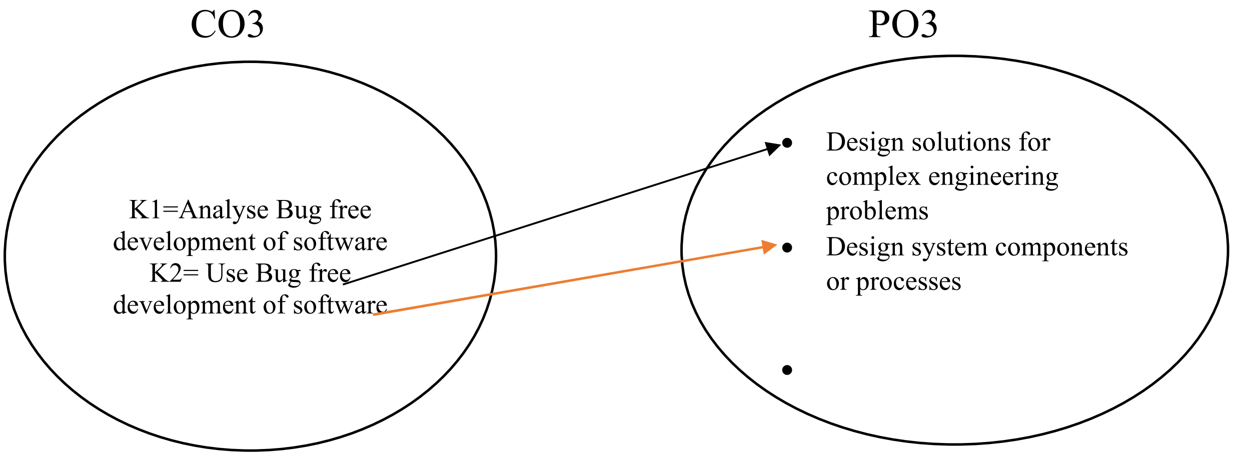
Correlation – CO3 to PO1 = 3


CO3
PO2
K1=Analyse Bug free development of software
K2= Use Bug free development of software
Analyse complex engineering problems Using principles of mathematics, natural sciences, and engineering sciences

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

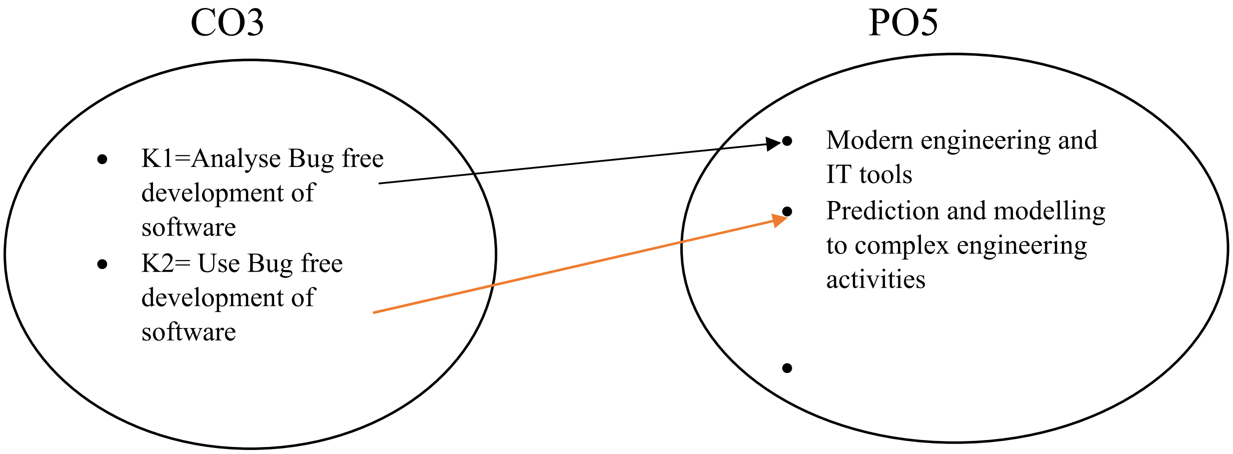
Correlation – CO3 to PO2 = 3


CO3
PO3
K1=Analyse Bug free development of software
K2= Use Bug free development of software
Design solutions for complex engineering problems
Design system components or processes

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+0/2 = 1/2=0.5$

Correlation – CO3 to PO3 = 1



CO3

- K1=Analyse Bug free development of software
- K2= Use Bug free development of software

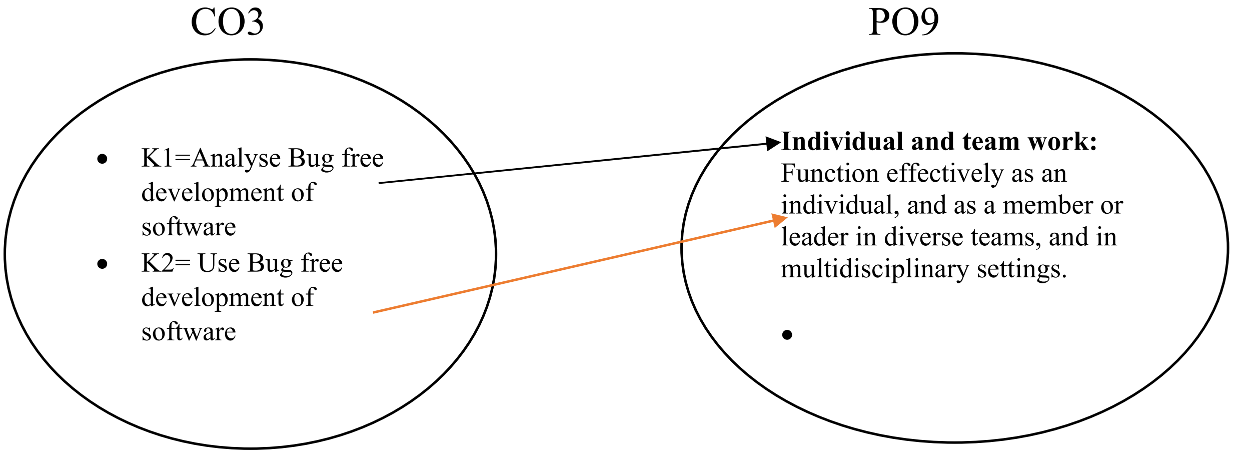
PO5

- Modern engineering and IT tools
- Prediction and modelling to complex engineering activities
-

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

Correlation – CO3 to PO5 = 3



CO3

- K1=Analyse Bug free development of software
- K2= Use Bug free development of software

PO9

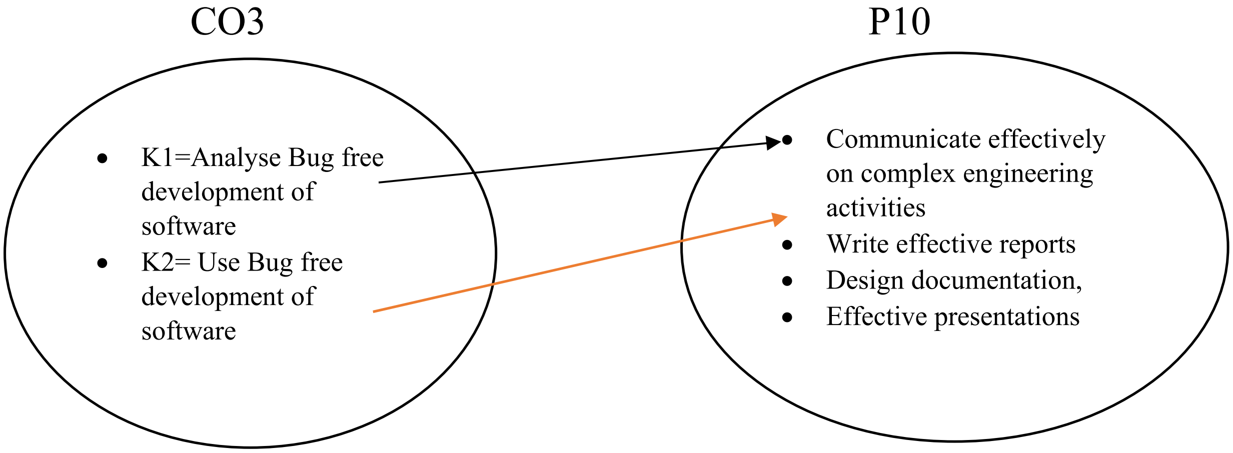
Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

-

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

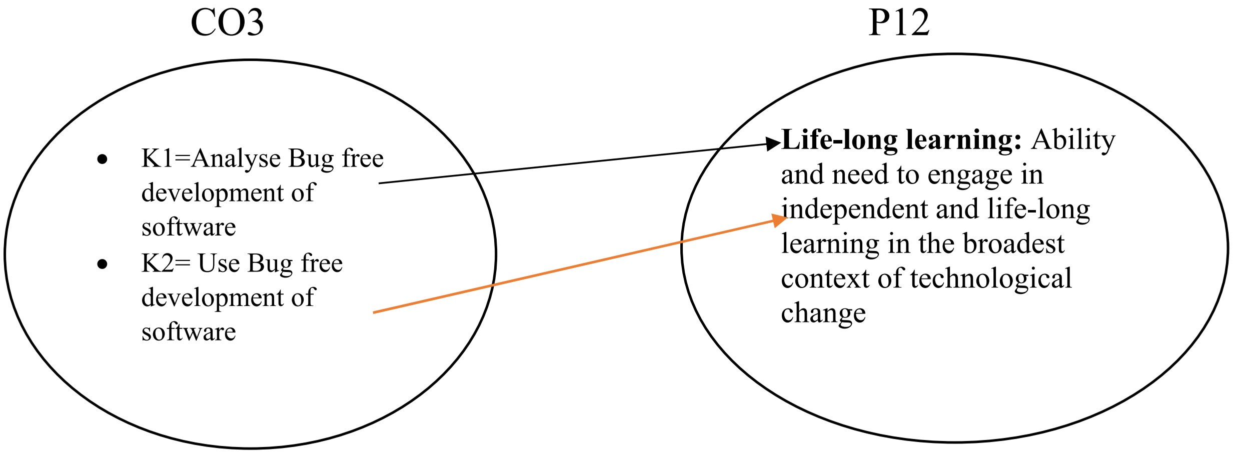
Correlation – CO3 to PO9 = 3


CO3
P10
K1=Analyse Bug free development of software
K2= Use Bug free development of software
Communicate effectively on complex engineering activities
Write effective reports
Design documentation,
Effective presentations

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

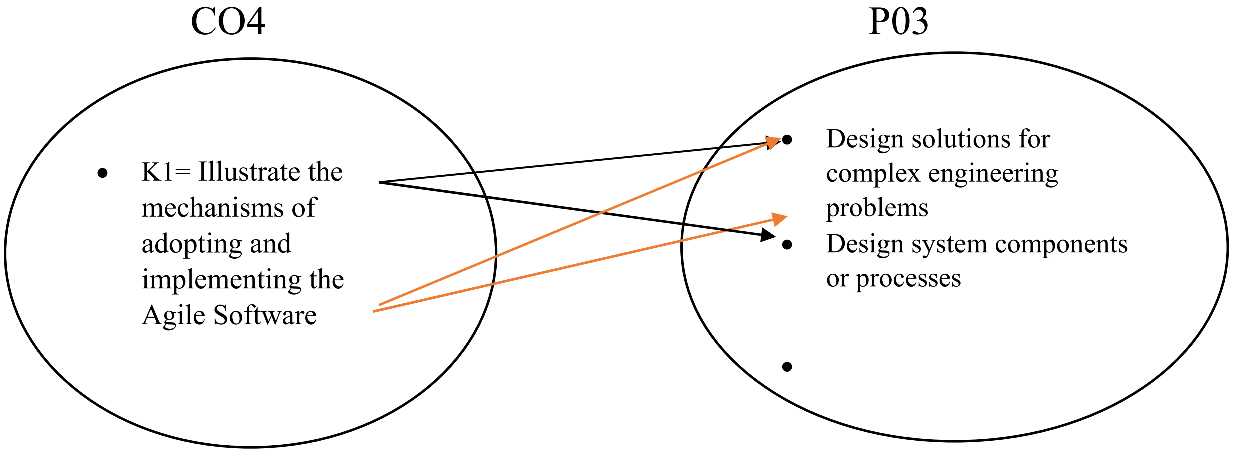
Correlation – CO3 to PO10 = 3


CO3
P12
K1=Analyse Bug free development of software
K2= Use Bug free development of software
Life-long learning: Ability and need to engage in independent and life-long learning in the broadest context of technological change

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

Correlation – CO3 to PO12 = 3



CO4

- K1= Illustrate the mechanisms of adopting and implementing the Agile Software

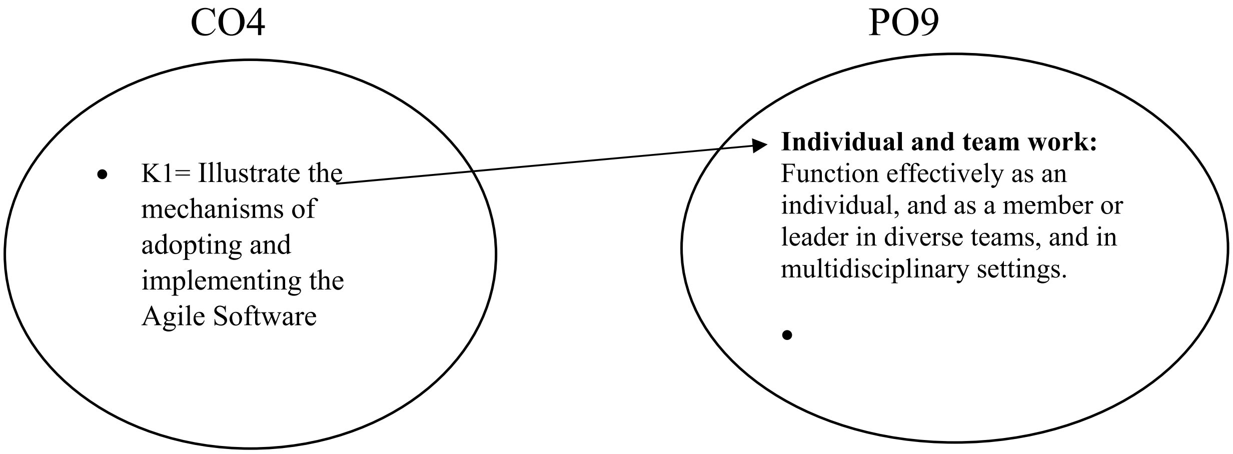
P03

- Design solutions for complex engineering problems
- Design system components or processes
-

Probability of CO# to PO# = $P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

Correlation – CO4 to PO3 = 3



CO4

- K1= Illustrate the mechanisms of adopting and implementing the Agile Software

PO9

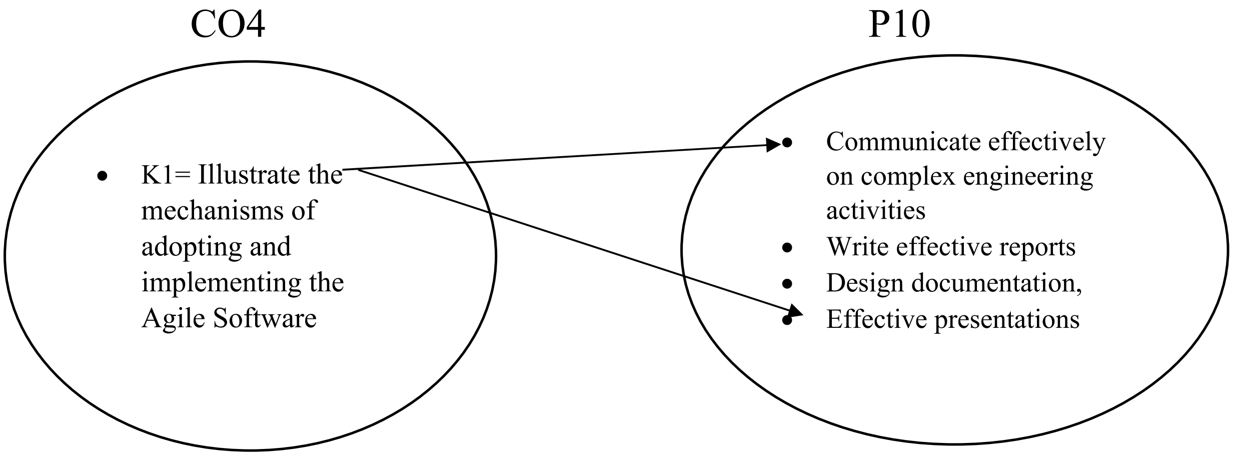
Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

-

Probability of CO# to PO# = $P(K_1) + P(K_2)$

$= 1/1 =1$

Correlation – CO4 to PO9 = 3



CO4

- K1= Illustrate the mechanisms of adopting and implementing the Agile Software

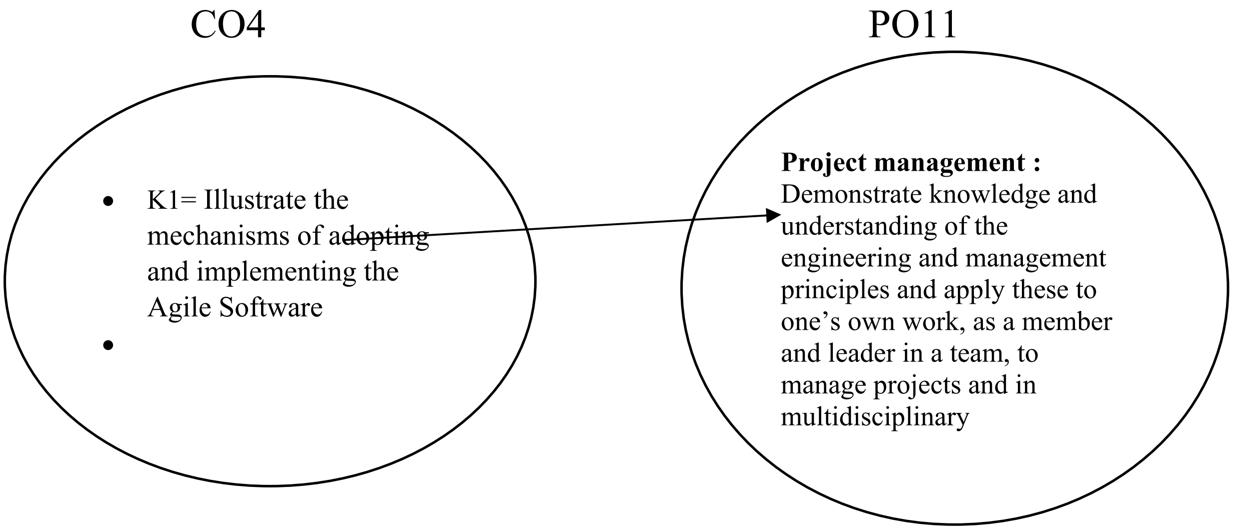
P10

- Communicate effectively on complex engineering activities
- Write effective reports
- Design documentation,
- Effective presentations

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/2+1/2 = 2/2=1$

Correlation – CO4 to PO10 = 3



CO4

- K1= Illustrate the mechanisms of adopting and implementing the Agile Software
-

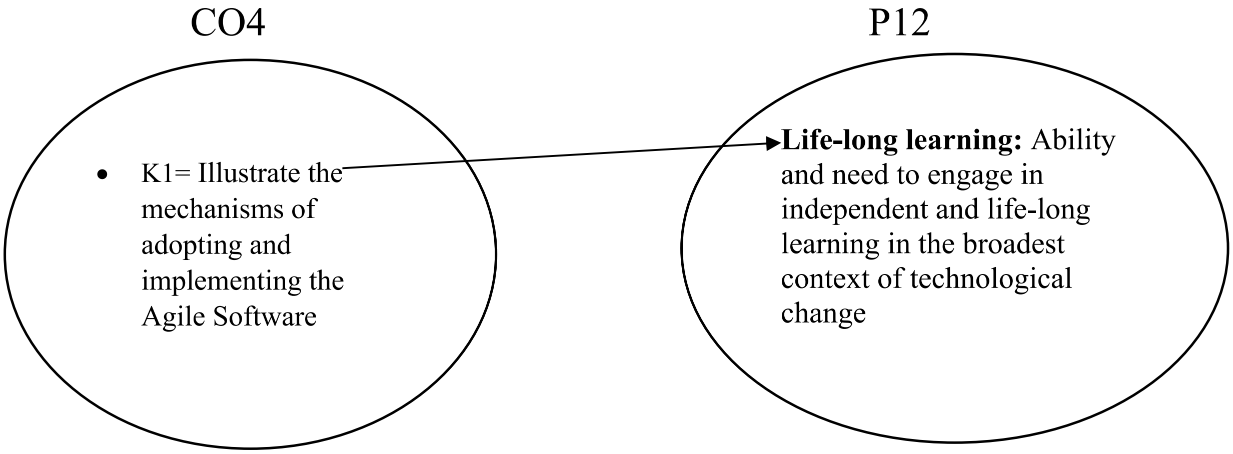
PO11

Project management : Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary

Probability of CO# to PO# $= P(K_1)$

$= 1/1 = 1$

Correlation – CO4 to PO11 = 3



CO4

- K1= Illustrate the mechanisms of adopting and implementing the Agile Software

P12

Life-long learning: Ability and need to engage in independent and life-long learning in the broadest context of technological change

Probability of CO# to PO# $= P(K_1) + P(K_2)$

$= 1/1=1$

Correlation – CO4 to PO12 = 3

8. Prerequisites

8. Prerequisites

1. Software Engineering.
2. Software Testing Tools

9. Class Timetable

Nawab Shah Alam Khan
COLLEGE OF ENGINEERING & TECHNOLOGY
BE: CE,ME,EEE,ECE,CSE,IT - ME: CSE, Embedded Sys, Structural, HVAC - Polytechnic: CE,ME,EEE,ECE
Approved by AICTE | Affiliated to OU | Accredited to NAAC | Permitted by Govt. of TS | Included in 2F UGC

IV B.TECH II SEM (A:Y 2020-21) REG (R16) BRANCH :- IT

TIME TABLE FOR ONLINE CLASSES W.EF :07-04-2021

DAY/ TIME	09:30AM - 10:15 AM	10:20AM- 11:00 AM	11:30 AM- 12:10 PM
	1	2	3
MONDAY	EIA	EIA	HCI
TUESDAY	EIA	MSE	HCI
WEDNESDAY	HCI	MSE	MSE
THURSDAY	PROJECT WORK		
FRIDAY	PROJECT WORK		
SATURDAY	PROJECT WORK		

S.NO	NAME OF SUBJECT	NAME OF FACULTY
1	MODERN SOFTWARE ENGINEERING(PE-V) (MSE)	MR QAZI M. A. BASHEER
2	HUMAN COMPUTER INTERACTION (PE-VI) (HCI)	DR. G.S.S.RAO
3	ENVIRONMENTAL IMPACT ASSESSMENT (OE-III) (EIA)	MR AMER KHUSRO
4	PROJECT CO-ORDINATOR	MS TAHERA ABID

CLASS

COORDINATOR HOD PRINCIPAL
MS. TAHERA ABID DR. G.S. RAO DR. SYED ABDUL SATTAR

10. Individual Timetable

Faculty Name: Qazi M A Basheer

Day/Time	10:30-11:30	11-12:40	12:40-1:30P.M	1:30-2:30	2:30-3:30	03:30/4:30
Mon		DS	Prayer & Lunch Time			
Tue		MSE				
Wed	MSE					
Thurs	DS					
Fri		DS				
Sat						

DS	Data Science
MSE	Modern Software Engg.

11. Lecture Schedule with methodology been used

S. No.	Period No.	Topic	Regular/ Additional	Teaching aids used PPT/ OHP/ BB	Remarks
		UNIT-I			
1	1	Introduction Extreme Programming (XP) - Agile Development Why Agile - Understanding Success	Regular	PPT	Online classes on MS Teams Online class
2	2	, Beyond Deadlines, Importance of Organizational Success, Introduction to Agility	Regular	PPT	
3	3	How to Be Agile - Agile methods	Regular	PPT	
4	4	Don't make your own method, Road to mastery	Regular	PPT	
5	5	, Understanding XP (Extreme Programming) - XP life cycle	Regular	PPT	
6	6	XP team, XP Concepts, Adopting XP	Regular	PPT	
7	7	Knowing whether XP is suitable, Implementing XP	Regular	,PPT	
		assessing Agility, Practicing XP			
8	8	Thinking - Pair Programming, Energized work	Regular	PPT	
9	9	Informative Workspace, Root cause Analysis, Retrospectives	Regular	PPT	
		UNIT-II	Regular	PPT	
10	10	Collaborating: Trust, Sit together	Regular	PPT	
11	11	Real customer involvement	Regular	PPT	
		Ubiquitous language			
12	12	meetings, coding standards	Regular	PPT	
13	13	Iteration demo	Regular	PPT	

14	14	Reporting	Regular	PPT	Online class on MS Teams Online class
		UNIT-III	Regular	PPT	
15	15	Releasing: Bugfree Release	Regular	PPT	
16	16	Version Control, fast build	Regular	PPT	
17	17	continuous integration		PPT	
18	18	Collective ownership	Regular	PPT	
19	19	Documentation	Regular	PPT	
		UNIT-IV	Regular	PPT	
20	20	Planing: Version, Release Plan	Regular	PPT	
		Risk Management, Iteration Planning		PPT	
21	21	Slack, Stories:	Regular	PPT	
22	22	Estimating	Regular	PPT	
		UNIT-V	Regular	PPT	
23	23	Developing: Incremental requirements	Regular	PPT	
24	24	Customer tests, Test driven development.		PPT	
25	25	Refactoring, Incremental design and architecture		PPT	
26	26	Spike solutions, Performance optimization, Exploratory testing		PPT	

12. Lesson Plan & Schedule

Lesson Plan & Schedule

<table>
<tr><td colspan="4"><b>NAWAB SHAH ALAM KHAN COLLEGE OF ENGG & TECH</b></td></tr>
<tr><td colspan="4">NEW MALAKPET HYDERABAD-24</td></tr>
<tr><td colspan="4">Department of Information Technology</td></tr>
<tr><td colspan="4">B.Tech(CSE) IVth Year Semester-II</td></tr>
<tr><td colspan="4">TEACHING PLAN</td></tr>
<tr><td colspan="4">Subject: Modern Software Engineering Faculty Name: Q.M.A.Basheer</td></tr>
<tr><th>S. No.</th><th>Date</th><th>Topic</th><th>Total No. of Periods</th></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td><b>UNIT-I</b></td><td></td></tr>
<tr><td>2</td><td>22/3/2021</td><td>Introduction Extreme Programming (XP) - Agile Development Why Agile - Understanding Success</td><td>1</td></tr>
<tr><td>3</td><td>23/3/2021</td><td>, Beyond Deadlines, Importance of Organizational Success, Introduction to Agility</td><td>1</td></tr>
<tr><td>4</td><td>24/3/2021</td><td>How to Be Agile - Agile methods</td><td>1</td></tr>
<tr><td>5</td><td>29/3/2021</td><td>Don't make your own method, Road to mastery</td><td>1</td></tr>
<tr><td>6</td><td>30/3/2021</td><td>, Understanding XP (Extreme Programming) - XP life cycle</td><td>1</td></tr>
<tr><td>7</td><td>31/3/2021</td><td>XP team, XP Concepts, Adopting XP</td><td>1</td></tr>
<tr><td>8</td><td>5/4/2021</td><td>Knowing whether XP is suitable, Implementing XP</td><td>1</td></tr>
<tr><td>9</td><td>6/4/2021</td><td>assessing Agility, Practicing XP</td><td>1</td></tr>
<tr><td>10</td><td>7/4/2021</td><td>Thinking - Pair Programming, Energized work</td><td>1</td></tr>
<tr><td>11</td><td>13/4/2021</td><td>Informative Workspace, Root cause Analysis, Retrospectives</td><td>1</td></tr>
<tr><td></td><td></td><td><b>UNIT-II</b></td><td></td></tr>
</table>

12	14/4/2021	Collaborating: Trust, Sit together	1
13	20/4/2021	Real customer involvement	1
14	27/4/2021	Ubiquitous language	1
15	28/4/2021	meetings, coding standards	1
16	03/5/2021	Iteration demo	1
17	04/5/2021	Reporting	1
		UNIT-III	1
19	10/5/2021	Releasing: Bugfree Release	1
20	11/5/2021	Version Control, fast build	1
21	12/5/2021	continuous integration	1
22	17/5/2021	Collective ownership	1
23	18/5/2021	Documentation	1
		UNIT-IV	1
24	01/6/2021	Planing: Version, Release Plan	1
25	02/6/2021	Risk Management, Iteration Planning	1
26	07/6/2021	Slack, Stories:	1
27	08/6/2021	Estimating	1
		UNIT-V	1
28	22/6/2021	Developing: Incremental requirements	1
29	23/6/2021	Customer tests, Test driven development.	1
30	24/6/2021	Refactoring, Incremental design and architecture	1
31	25/6/2021	Spike solutions, Performance optimization, Exploratory testing	1

13. Detailed Notes

Unit 1

Why Agile?

Agile development is popular, but that's no reason to use it.

The real question: will agile development make your team more successful?

Understanding Success
The Traditional Idea of Success is usually defined as delivering on time, under budget, and as specified. That's a flawed definition.

Many late projects are huge successes for their organizations, and many on-time projects don't deliver any value.

Instead, think in terms of organizational, technical, and personal success.
Agile development is no silver bullet, but it is useful.

Organizationally, agile delivers value and reduces costs;

technically, it highlights excellence and minimal bugs; personally, many find it their preferred way to work.

Definitions of Successful, Challenged, Impaired

Successful

“Completed On Time, on Budget, With all features and Functions as originally specified”

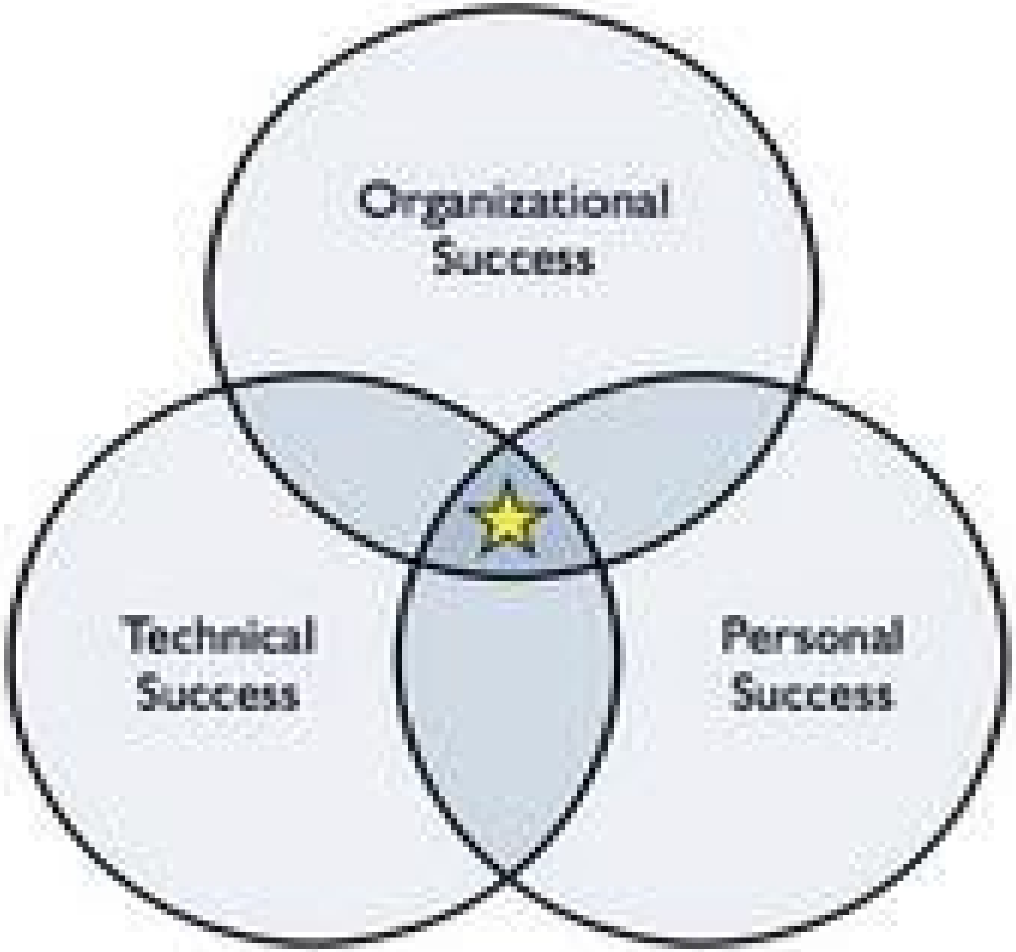
Challenged

“Completed and operational but over budget, over the time estimate, fewer features and functions than originally specified”

Impaired

“Cancelled at some point during the development cycle”

Aspect of Success or Type of Success



All the above mentioned successes are important

Without personal success, you will have trouble to motivating yourself and employees.

Without technical success, your source code will eventually collapse under its own weight.

Without organizational success, your team may find they are no longer wanted in the company.

WHAT DO ORGANIZATIONS VALUE?

Aside from revenue and cost savings, sources of value include:

✓ Competitive Differentiation

✓ Brand Projection

✓ Enhanced Customer loyalty

✓ Satisfying regulatory requirements

✓ Original Research

✓ Strategic Information

Will Agile Development help you more successful?

It might.

Why because, agile development focuses on achieving personal, technical, and organizational successes.

Organizational Success

Agile methods achieve organizational successes by focusing on delivering value and decreasing costs. This directly translates to increased return on investment.

Agile method also sets expectations early in the project, so if your project won't be an organizational success, you will find out early enough to cancel it before your organization spent much money on the project.

Technical Success

XP programmers work together, which helps them keep track of the nitpicky details necessary for great work and ensures that at least two people review every piece of code.

Programmers continuously integrate their code, which enables the team to release the software whenever it makes business sense.

The whole team focuses on finishing each feature completely before starting the next, which prevents unexpected delays before release and allows the team to change direction at will.

Personal Success

Personal success is, well, personal. Agile development may not satisfy all of your requirements for personal success. However, once you get used to it, you'll probably find a lot to like about it, no matter who you are:

Executives and senior management

They will appreciate the team's focus on providing a solid return on investment and the software's longevity.

Users, stakeholders, domain experts, and product managers

They will appreciate their ability to influence the direction of software development, the team's focus on delivering useful and valuable software, and increased delivery frequency.

Project and product managers

They will appreciate their ability to change direction as business needs change, the team's ability to make and meet commitments, and improved stakeholder satisfaction.

Developers

They will appreciate their improved quality of life resulting from increased technical quality, greater influence over estimates and schedules, and team autonomy.

Testers

They will appreciate their integration as first-class members of the team, their ability to influence quality at all stages of the project, and more challenging, less repetitious work.

How to Be Agile?

What does it mean to "be agile"?

Agile development isn't a specific process you can follow.

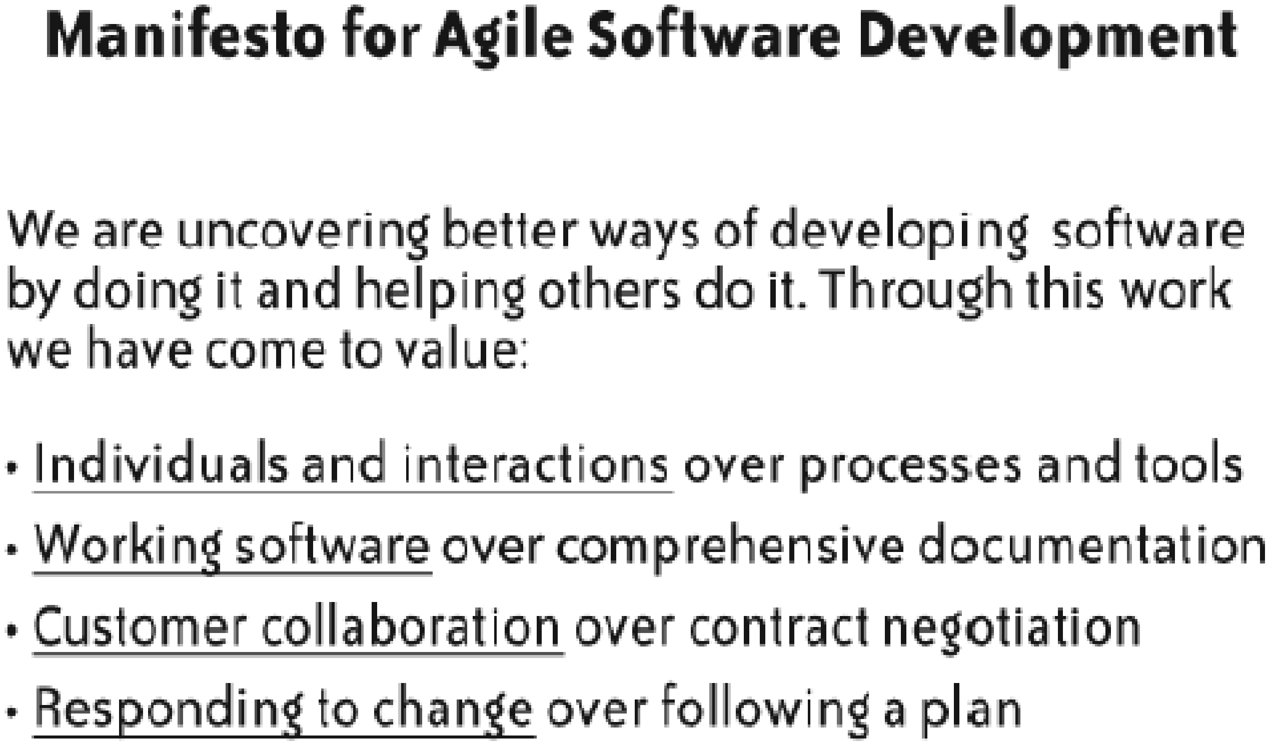
No team practices the Agile method. There's no such thing.

Agile development is a philosophy.

The canonical description of this way of thinking is the Agile Manifesto, a collection of 4 values and 12 principles.

To “be agile,” you need to put the agile values and principles into practice.

4 Values of Agile

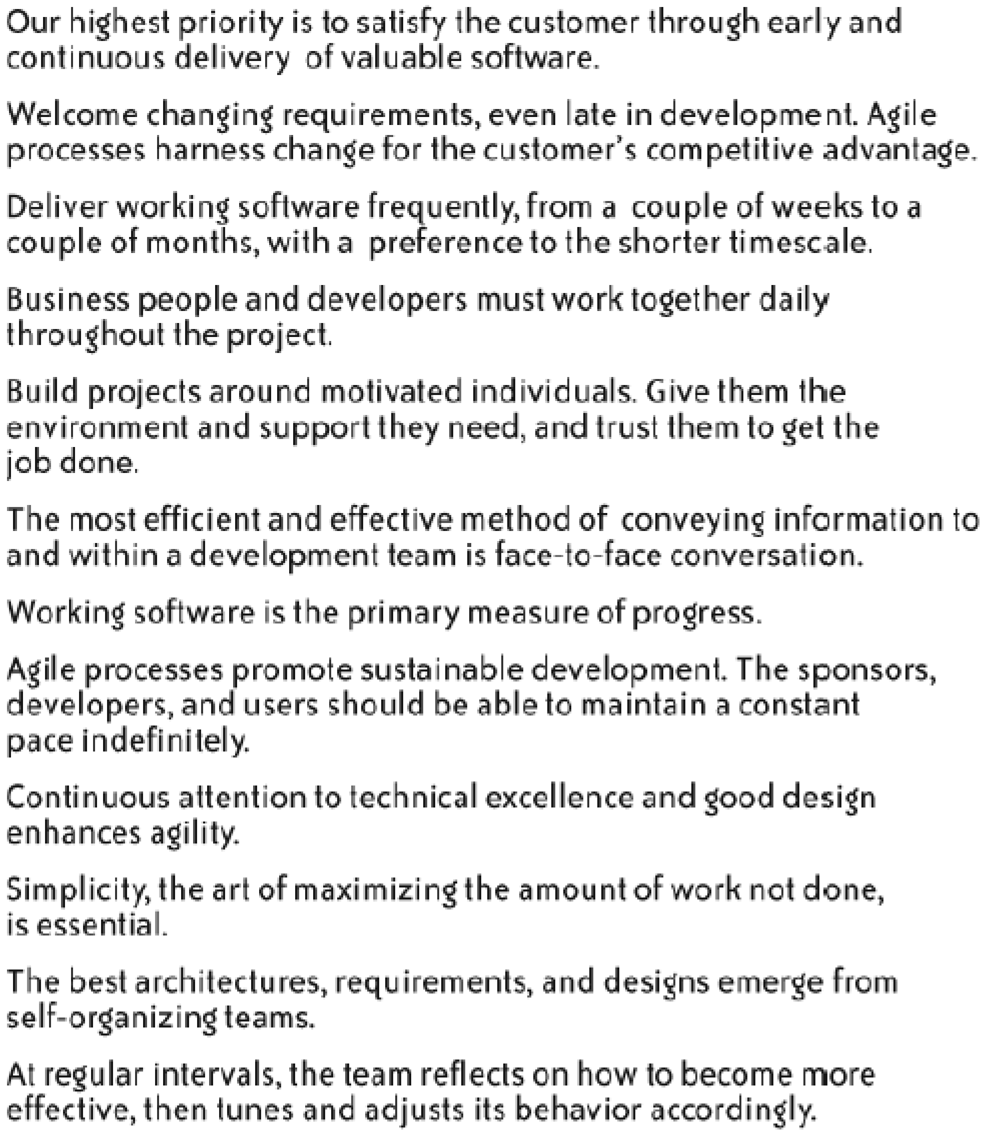


Manifesto for Agile Software Development

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

12 Principles of Agile



Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity, the art of maximizing the amount of work not done, is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Methods

A method, or process, is a way of working.

Whenever you do something, you’re following a process.

Some processes are written, as when assembling a piece of furniture; others are ad-hoc and informal, as when I clean my house.

Agile methods are processes that support the agile philosophy.

For Example Include Extreme Programming and Scrum.

Don’t Make Your Own Method

Just as established agile methods combine existing practices, you might want to create your own agile method by mixing together practices from various agile methods.

At first glance, this doesn’t seem too hard. There are scores of good agile practices to choose from.

However, creating a brand-new agile method is a bad idea if you’ve never used agile development before.

The Road to Mastery

Mastering the art of agile development requires real-world experience using a specific, well-defined agile method.

For Example: Extreme Programming for this purpose.

It has several advantages:

- Of all the agile methods, XP is the most complete. It places a strong emphasis on technical practices in addition to the more common teamwork and structural practices.
- XP has undergone intense scrutiny. There are thousands of pages of explanations, experience reports, and critiques out there. Its capabilities and limitations are very well understood.
- I have a lot of experience with XP,

which allows me to share insights and practical tips that will help you apply XP more easily.

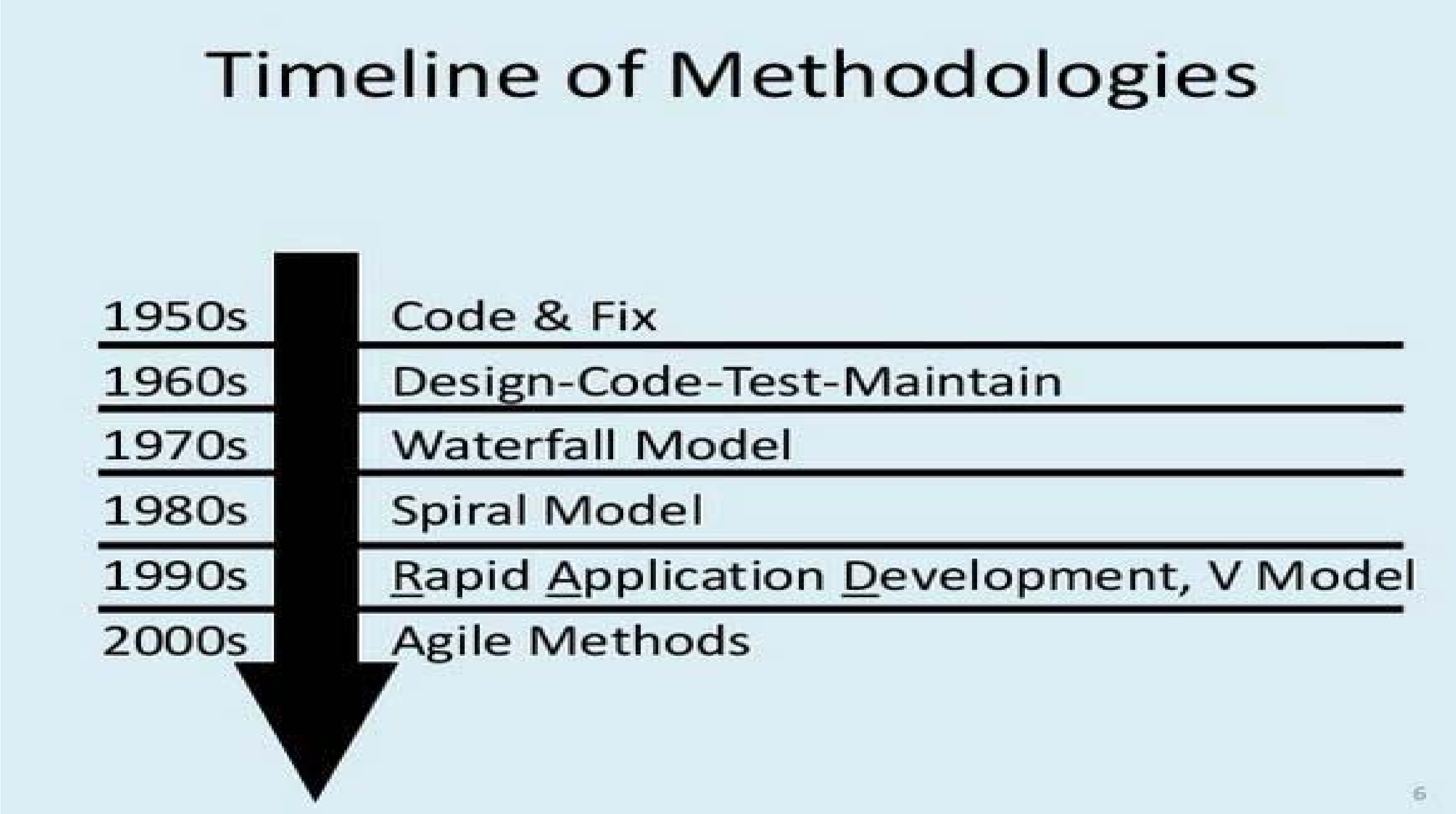
To master the art of agile development
or
simply to use XP to be more successful—follow these steps:

1. Decide why you want to use agile development. Will it make your team and organization more successful? How?
2. Determine whether this book’s approach will work for your team.
3. Adopt as many of XP’s practices as you can.
4. Follow the XP practices rigorously and consistently.

As you become confident that you are practicing XP correctly—again, give it several months—start experimenting with changes that aren’t “

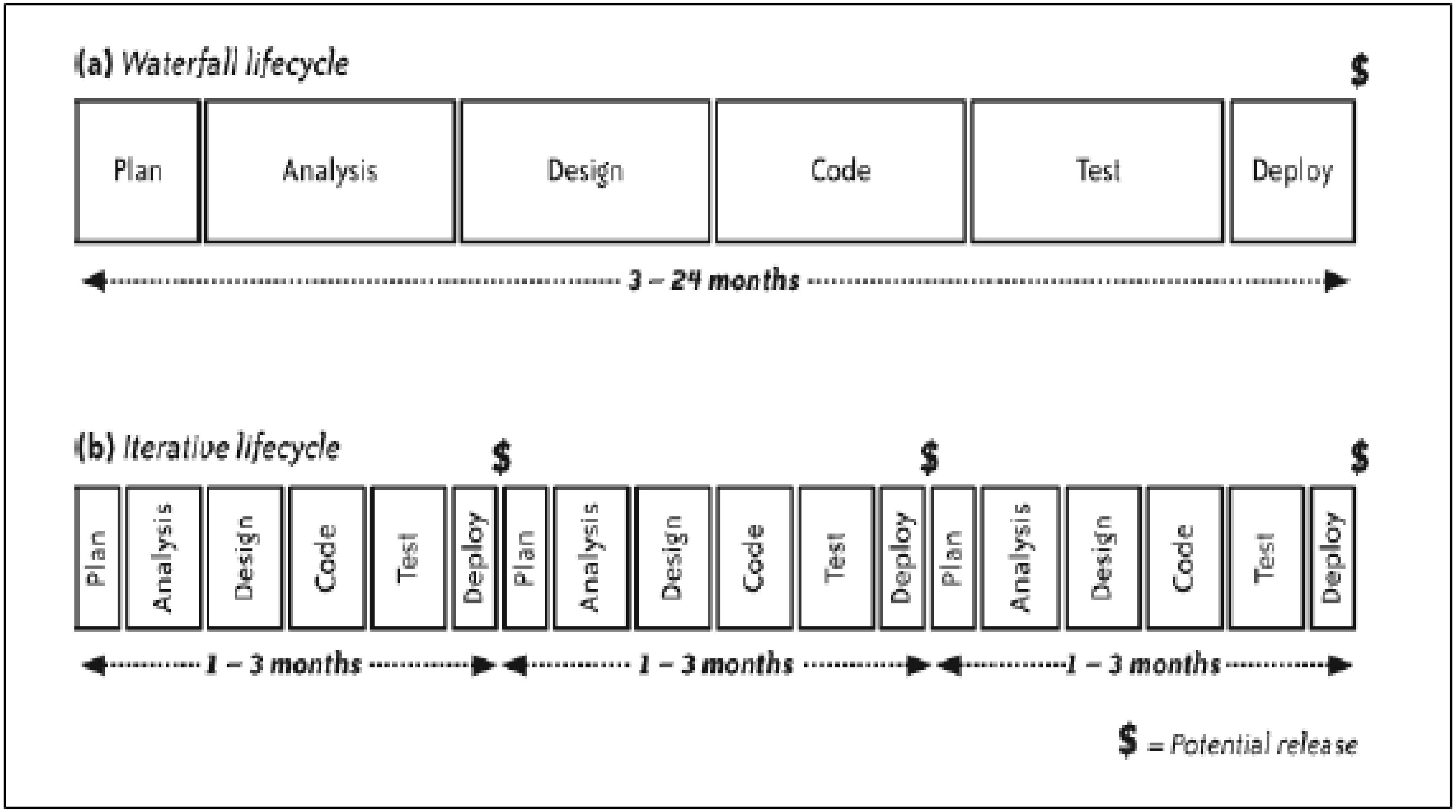
Understanding XP

Its is a Model that that represent one method as to how software can be developed.

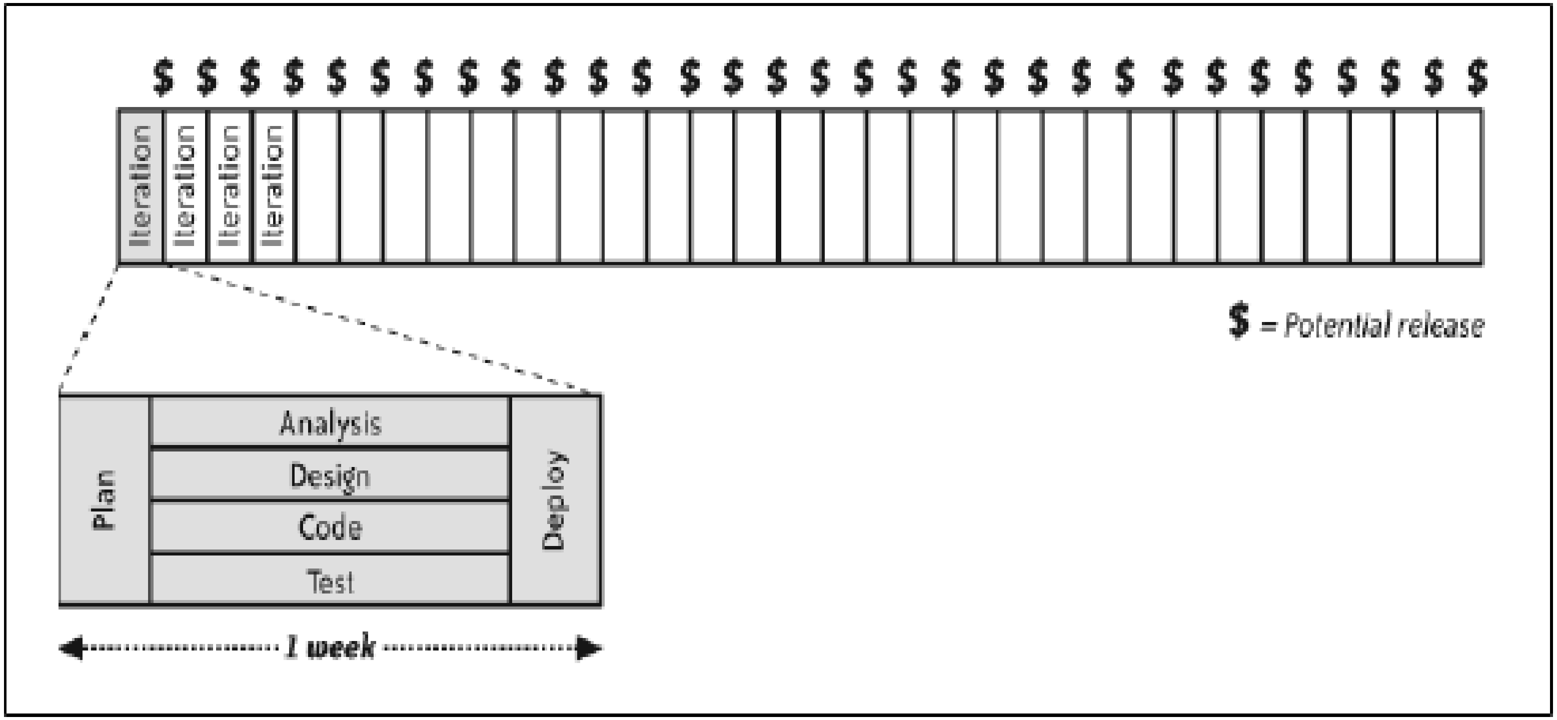


Timeline of Methodologies

1950s	Code & Fix
1960s	Design-Code-Test-Maintain
1970s	Waterfall Model
1980s	Spiral Model
1990s	Rapid Application Development, V Model
2000s	Agile Methods


(a) Waterfall lifecycle
$
Plan
Analysis
Design
Code
Test
Deploy
3 – 24 months
(b) Iterative lifecycle
$
$
$
Plan
Analysis
Design
Code
Test
Deploy
Plan
Analysis
Design
Code
Test
Deploy
Plan
Analysis
Design
Code
Test
Deploy
1 – 3 months
1 – 3 months
1 – 3 months
$ = Potential release

XP Life Cycle



Using simultaneous phases, an XP team produces deployable software every week.

In each iteration, the team analyzes, designs, codes, tests, and deploys a subset of features.

XP teams perform nearly every software development activity simultaneously. Analysis, design, coding, testing, and even deployment occur with rapid frequency.

Planning

Every XP team includes several business experts—the on-site customers—who are responsible for making business decisions.

The on-site customers point the project in the right direction by clarifying the project vision, creating stories, constructing a release plan, and managing risks.

Programmers provide estimates and suggestions, which are blended with customer priorities in a process called the planning game.

Together, the team strives to create small, frequent releases that maximize value.

Analysis

Rather than using an upfront analysis phase to define requirements, on-site customers sit with the team full-time.

On-site customers may or may not be real customers depending on the type of project, but they are the people best qualified to determine what the software should do.

Design and coding

XP uses incremental design and architecture to continuously create and improve the design in small steps.

This work is driven by Test-driven development (TDD), an activity that inextricably weaves together testing, coding, design, and architecture.

To support this process, programmers work in pairs, which increases the amount of brainpower brought to bear on each task and ensures that one person in each pair always has time to think about larger design issues.

Programmers are also responsible for managing their development environment.

They use a version control system for configuration management and maintain their own automated build.

Programmers integrate their code every few hours and ensure that every integration is technically capable of deployment.

To support this effort, programmers also maintain coding standards and share ownership of the code. The team shares a joint aesthetic for the code, and everyone is expected to fix problems in the code regardless of who wrote it.

Testing

XP includes a sophisticated suite of testing practices.

testers help the team understand whether their efforts are in fact producing high quality code.

They use exploratory testing to look for surprises and gaps in the software.

When the testers find a bug, the team conducts root-cause analysis and considers how to improve their process to prevent similar bugs from occurring in the future.

Testers also explore the software’s nonfunctional characteristics, such as performance and stability.

Customers then use this information to decide whether to create additional stories.

Deployment

XP teams keep their software ready to deploy at the end of any iteration. They deploy the software to internal stakeholders every week in preparation for the weekly iteration demo.

Deployment to real customers is scheduled according to business needs.

The XP Team

Team software development is different. The same information is spread out among many members of the team.

Different people know:

- How to design and program the software (programmers, designers, and architects)
- Why the software is important (product manager)
- The rules the software should follow (domain experts)
- How the software should behave (interaction designers)
- How the user interface should look (graphic designers)
- Where defects are likely to hide (testers)
- How to interact with the rest of the company (project manager)
- Where to improve work habits (coach)

All of this knowledge is necessary for success. XP acknowledges this reality by creating cross functional teams composed of diverse people who can fulfill all the team’s roles.

On-Site Customers

On-site customers—often just called customers—are responsible for defining the software the team builds.

→ The rest of the team can and should contribute suggestions and ideas, but the customers are ultimately responsible for determining what stakeholders find valuable.

→ Customers' most important activity is release planning. This is a multifaceted activity.

→ Customers need to evangelize the project's vision; identify features and stories; determine how to group features into small, frequent releases; manage risks; and create an achievable plan by coordinating with programmers and playing the planning game.

→On-site customers may or may not be real customers, depending on the type of project.

→ Regardless, customers are responsible for refining their plans by soliciting feedback from real customers and other stakeholders. One of the venues for this feedback is the weekly iteration demo, which customers lead.

The product manager (aka product owner)

The product manager has only one job on an XP project, but it’s a doozy.

Domain experts (aka subject matter experts)

→ Most software operates in a particular industry, such as finance, that has its own specialized rules for doing business.

→ To succeed in that industry, the software must implement those rules faithfully and exactly.

→ These rules are domain rules, and knowledge of these rules is domain knowledge.

→ Most programmers have gaps in their domain knowledge, even if they’ve worked in an industry for years. In many cases, the industry itself doesn’t clearly define all its rules.

→ The basics may be clear, but there are nitpicky details where domain rules are implicit or even contradictory.

Interaction designers

The user interface is the public face of the product. For many users, the UI is the product. They judge the product’s quality solely on their perception of the UI.

Business analysts

On non agile teams, business analysts typically act as liaisons between the customers and developers, by clarifying and refining customer needs into a functional requirements specification.

Programmers

A great product vision requires solid execution. The bulk of the XP team consists of software developers in a variety of specialties. Each of these developers contributes directly to creating working code. To emphasize this, XP calls all developers programmers.

Designers and architects

Everybody codes on an XP team, and everybody designs. Test-driven development combines design, tests, and coding into a single, ongoing activity.

Expert designers and architects are still necessary. They contribute by guiding the team’s incremental design and architecture efforts and by helping team members see ways of simplifying complex designs. They act as peers—that is, as programmers—rather than teachers, guiding rather than dictating.

Technical specialists

In addition to the obvious titles (programmer, developer, software engineer), the XP “programmer” role includes other software development roles. The programmers could include a database designer, a security expert, or a network architect. XP programmers are generalizing specialists.

XP Core Practice #1- The Planning Game

- Business and development cooperate to produce **max business value** as quickly as possible.
- The planning game:
  - Business comes up with a list of desired **features**.
  - Each feature is written out as a **User Story**,
    - feature has a name, and is described in broad strokes what is required.
  - **User stories** are typically written on 4x6 cards. (You saw a variation in your book)
  - **Development estimates** how much effort each story will take, and **how much effort the team** can produce in a given time interval.
  - **Business then decides**
    - order of stories to implement,
    - And when and how often to produce a **production release** of the system.

XP – Core Practice #2: Simple Design

- Simplest possible design to get job done.
- Requirements will change tomorrow, do what's needed to meet today's requirements

- Design in XP is not a one-time; it is an "all-the-time" activity. Have design steps in
  - release planning
  - iteration planning,
  - teams engage in quick design sessions and design revisions through refactoring,
- through the course of the entire project.

XP – Core Practice #3: Metaphor

- Extreme Programming teams develop a <u>common vision </u>of how the program works, which we call the "metaphor".

- At its best, the **metaphor** is a simple evocative description of how the program works.

- XP teams use
- common system of **names** to be sure that everyone understands how the system works
- and **where to look to find the functionality** you're looking for,
- or to find the right place to **put the functionality** you're about to add.

Metaphor

•Metaphor is something you start using when your mother asks what you are working on and you try to explain her the details. How you find it is very project-specific. Use your common sense or find the guy on your team who is good at explaining technical things to customers in a way that is easy to understand.

•What XP suggests in my opinion are the following:

- Try to design a system that is easy to explain using real-life analogies. Your systems are complex, try to use a design, where the relationship and interactions between sub-components are clear and resemble something that people with common sense have already seen.

- Use the analogies in all communications: source-code, planning meetings, speaking to users, or God forsake, writing documentation. If you find that the concepts you use do not fit to some area, try to find a better metaphor. (Wiki)

XP – Core Practice #4: Simple Design

- Always use the simplest possible design that gets the job done.

- The requirements will change tomorrow, so only do what’s needed to meet today’s requirements.

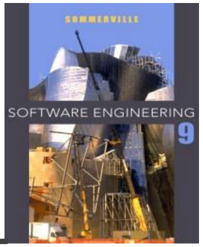
XP – Core Practice #5: Continuous Testing

- XP teams focus on **validation** of the software at all times

- Programmers develop software by **writing tests first**, and **then code** that fulfills the requirements reflected in the tests.

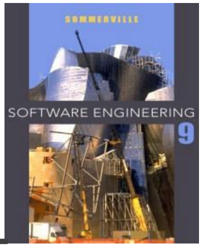
- Customers provide **acceptance tests** that enable them to be **certain** that the features they need are provided.

Testing in XP



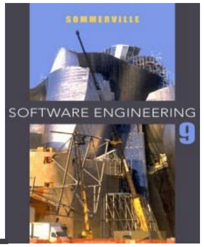
- Testing is central to XP and XP has developed an approach where the **program is tested after every change has been made.**
- XP testing features:
  - Test-first development.
  - Incremental test development from scenarios.
  - User involvement in test development and validation.
  - Automated test harnesses are used to run all component tests each time that a new release is built.

Test-First Development



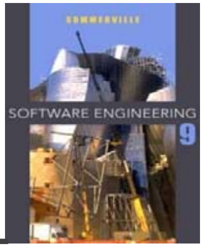
- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
  - Usually relies on a testing framework such as Junit.
- All previous and new tests **are run automatically** when new functionality is added, thus checking that the new functionality has not introduced errors.

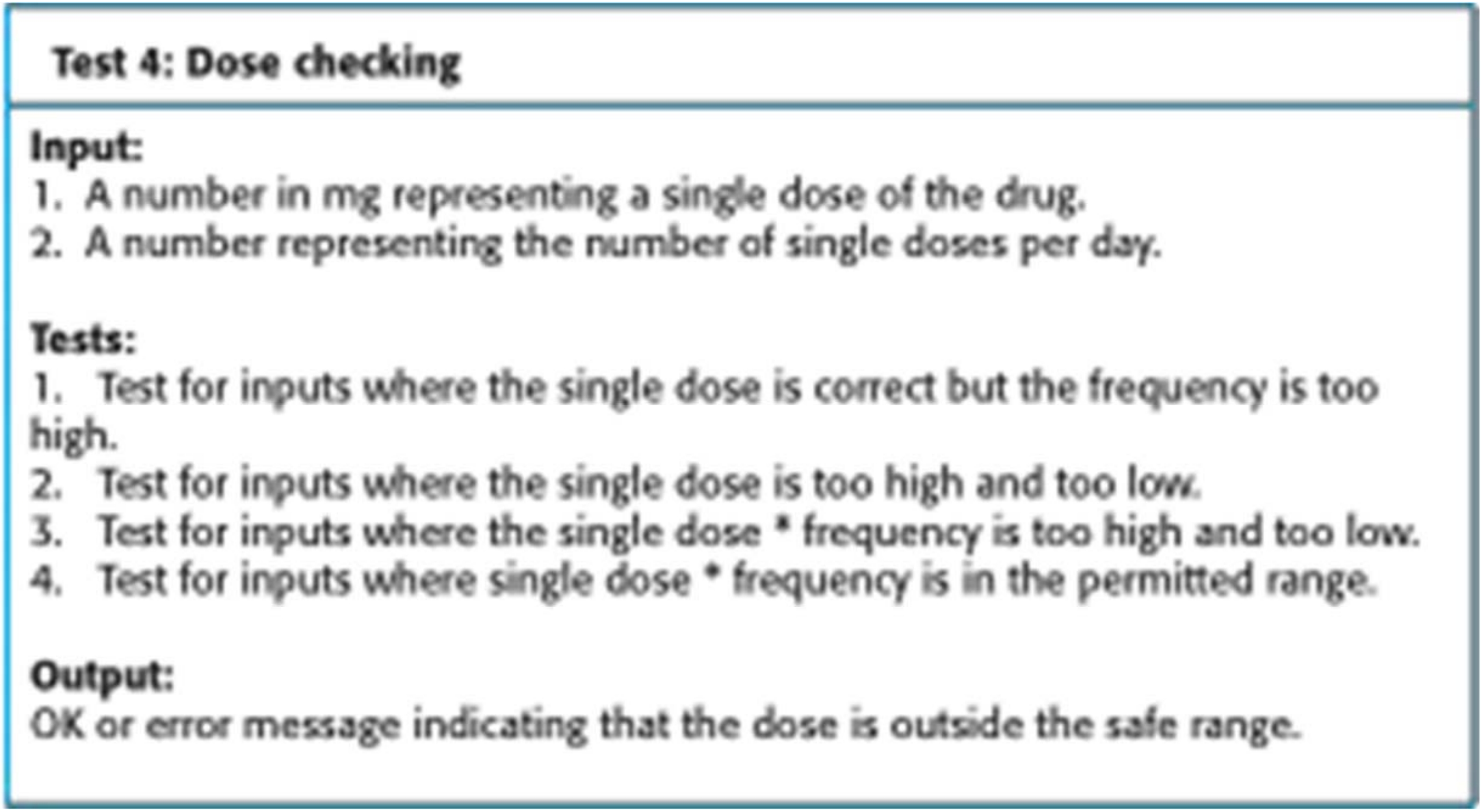
Customer Involvement



- The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- The customer **who is part of the team** writes tests **as development proceeds**. All new code is **therefore** validated to ensure that it is what the customer needs.
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

Test case Description for Dose Checking



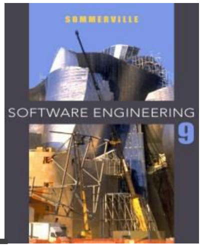


Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

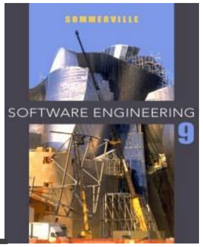
Output:
OK or error message indicating that the dose is outside the safe range.



Test Automation

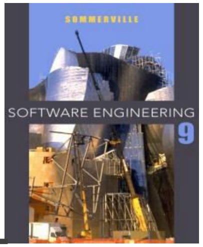
- **Test automation means that tests are written as executable components before the task is implemented**
  - These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification. An automated test framework (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.
- **As testing is automated, there is always a set of tests that can be quickly and easily executed**
  - Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.

XP Testing Difficulties



- Programmers prefer programming to testing and sometimes they take short cuts when writing tests.
- For example, they may **write incomplete tests** that do
- not check for all possible exceptions that may occur.
- Some tests can be **very difficult to write** incrementally. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the 'display logic' and workflow between screens.
- It **difficult to judge the completeness** of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.

XP and Change



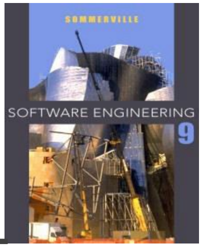
- **Conventional wisdom in software engineering is to design for change**.
  - It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- **XP, however, maintains that this is not worthwhile** as changes cannot be reliably anticipated.
- **Rather**, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.

XP – Core Practice #6: Refactoring

- XP Team **Refactor** out any duplicate code generated in a coding session.

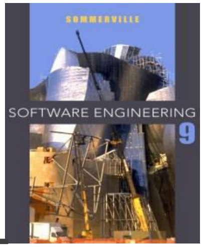
- Refactoring is simplified due to extensive use of **automated** test cases.

Refactoring



- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

Examples of Refactoring



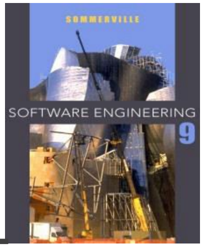
- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

XP – Core Practice #7: Pair Programming

- All production code is written by two programmers sitting at one machine.
  - This practice ensures that all code is reviewed as it is written and results in better Design, testing and better code.

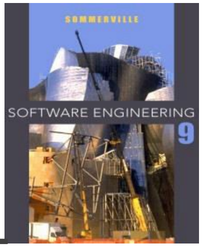
- Some programmers object to pair programming without ever trying it.
  - It does take some practice to do well, and you need to do it well for a few weeks to see the results.
  - Ninety percent of programmers who learn pair programming prefer it, so it is recommended to all teams.
- Pairing, in addition to providing better code and tests, also serves to **communicate knowledge** throughout the team.

Pair Programming



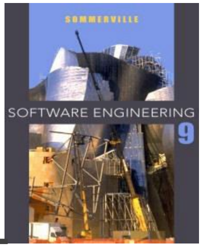
- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

Pair Programming



- In pair programming, programmers sit together at the same workstation to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair programming is not necessarily inefficient and there is evidence that a pair working together is more efficient than 2 programmers working separately.

Advantages of Pair Programming



- It supports the idea of collective ownership and responsibility for the system.
  - Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
- It acts as an informal review process because each line of code is looked at by at least two people.
- It helps support refactoring, which is a process of software improvement.
  - Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process.

XP – Core Practice #8: Collective Code Ownership

- No single person "owns" a module.
- Any developer is expected to be able to work on any part of the codebase at any time.

XP – Core Practice #9: Continuous Integration

- All changes are integrated into the codebase at least daily.
- Unit tests have to run 100% both before and after integration.
  - Infrequent integration leads to serious problems on a project.

- Although integration is critical to shipping good working code, the team is not practiced at it, and often it is delegated to people not familiar with the whole system.

- Problems creep in at integration time that are not detected by any of the testing that takes place on an un-integrated system.

- **Code freezes** mean that you have long time periods when the programmers could be working on important shippable features, but that those features must be held back.

XP – Core Practice #10: 40-hour Week

- Programmers go home on time.
  - In crunch mode, up to one week of overtime is allowed.
- Multiple consecutive weeks of overtime are treated as a sign that something is very wrong with the process and/or schedule.

XP – Core Practice #11: On-Site Customer

- Development team has **continuous access** to the customer who will actually be using the system.

- For initiatives with lots of customers, a **customer representative** (i.e. Product Manager) will be designated for Development team access.

XP – Core Practice #12: Coding Standards

- Everyone codes to the same standards.
- The specifics of the standard are not important: what is important is that **all** of the code looks familiar, in support of **collective ownership**.

XP Values – Summarized.

- XP is a values-based methodology. The **values** are Simplicity, Communication, Feedback and Courage.

- XP's core values:best summarized in the following statement by Kent Beck: **Do more of what works and do less of what doesn't**.

Highlights of the four values itemized:

- **Simplicity** encourages:
  - Delivering the simplest functionality that meets business needs
  - Designing the simplest software that supports the needed functionality
  - Building for today and not for tomorrow
  - Writing code that is easy to read, understand, maintain and modify

Highlights of the four values itemized:

- **Communication** is accomplished by:
  - Collaborative workspaces
  - Co-location of development and business space
  - Paired development
  - Frequently changing pair partners
  - Frequently changing assignments
  - Public status displays
  - Short standup meetings
  - Unit tests, demos and oral communication, not documentation

Highlights of the four values itemized:

- **Feedback** is provided by:
  - Aggressive iterative and incremental releases
  - Frequent releases to end users
  - Co-location with end users
  - Automated unit tests
  - Automated functional tests
  - Courage is required to:
    - Do the right thing in the face of opposition
    - Do the practices required to succeed

SCRUM

- Idea first appeared in a business journal in 1986 (applied to product development management).
- Used in software development and presented in 1995 paper.
- Term is based on rugby term
- Small cross-functional teams

SCRUM Practices

- Product and release *backlog*
  - A list of the features to be implemented in the project (subdivided to next release), ordered by priority
  - Can adjust over time as needed, based on feedback
  - A product manager is responsible for maintaining

SCRUM Practices

- *Burn-down* chart
  - Make best estimate of time to complete what is currently in the backlog
  - Plot the time on a chart
  - By studying chart, understand how team functions
  - Ensure burndown to 0 at completion date
    - By adjusting what’s in the backlog
    - By adjusting the completion date

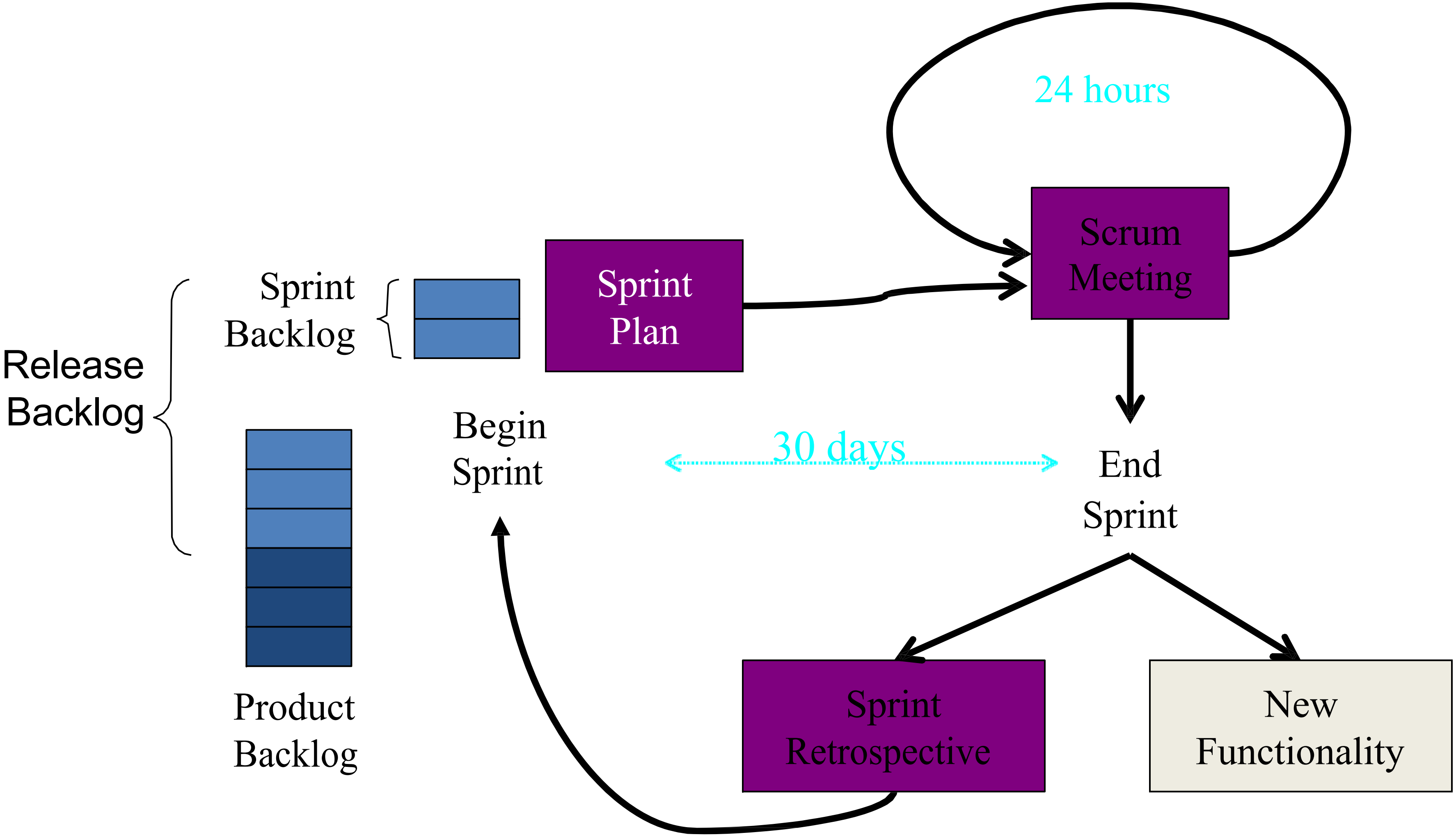
SCRUM Practices

- The *sprint*
  - The sprint is a ~1 month period after which some product is delivered
  - Features are assigned from the product backlog to a sprint backlog
    - Features divided into smaller tasks for sprint backlog
    - Feature list is fixed for sprint
  - Planning meeting
    - Tasks can be assigned to team members
    - Team members have individual estimates of time taken per item
  - During sprint, work through features, and keep a burn-down chart for the sprint
  - New functionality is produced by the end of the sprint
  - After sprint, a review meeting is held to evaluate the sprint

SCRUM Practices

- Scrum meeting
  - 15 minute *daily* meeting
  - All team members show up
  - Quickly mention what they did since last Scrum, any obstacles encountered, and what they will do next
  - Some team member volunteers or is appointed to be the *Scrum Master* - in charge of Scrum meeting, and responsible for seeing that issues raised get addressed
  - Customers, management encouraged to observe

SCRUM Practices



Collaborating

Collaborating

- The more effectively a programmer can access and understand the information they need , the more effective they will be at creating software
- The better information customers and managers have the better they can manage the schedule and provide feedback to the programmers
- Eight practices to help your team and it’s stakeholders collaborate efficiently and effectively

➢Trust

➢Sitting together

➢Real customer involvement

Contd…….

- Ubiquitous language
- Stand up meetings
- Coding standards
- Iteration demo
- Reporting

Trust

- We work together effectively and with out fear
- Trust is essential for a team to perform
- 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

Team Strategy1: Customer- Programmer Empathy

- Customer often feel that programmers don’t care about their needs and deadlines
- Programmers are forced to commitments they can’t meet
- Customers react by ignoring programmer estimates and applying schedule pressure
- The biggest missing component is empathy for other group
- Sitting together is the most effective way to build empathy

Team Strategy 2: Programmer –Tester Empathy

- Programmers does not show respect for the testers abilities
- Testers see their mission as shooting down the programmers work
- The component missing here is empathy
- Programmers should remember that testing takes skill and careful work
- Take advantage of testers ability to find mistakes you would never consider and thank them for helping prevent problems

Team Strategy 3: Eat Together

- Another way to improve team cohesiveness is to eat together
- 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
- If you go to restaurant ask for a single long table rather than a separate table

Team Strategy 4:Team Continuity

- After a project ends the team typically breaks up
- The next project starts with a brand new team
- You can avoid this waste by keeping productive team
- Most organization think of people as basic resource in the company instead of that think of the team as a resource
- Rather than assigning people to a project assign a team to a project
- Some teams will be more effective than others
- Rotate junior members into those teams so that they can learn from the best, rotate experienced team members out to lead teams of their own

Organizational Strategy 1: Show some hustle

- In software team hustle is energised, productive work
- It is the sense that the team is putting in a fair days work for a fair days pay
- Energised work , an informative workspace , appropriate reporting and iteration demos help convey this feeling of productivity

Organizational Strategy 2: Deliver on Commitments

- Stake holders may not know how to evaluate your process , but they can evaluate results
- 2 kinds of result that speak to them are working software and delivering on commitments
- XP team demonstrate both of this every week
- You make a commitment to deliver the working software when you build your iteration and release plans
- You demonstrate that you’ve met the iteration commitment in iteration demo and release commitment on your predefined release date

Organizational Strategy 3: Manage Problems

- First limit your exposure to problems
- Work on the hardest , most uncertain tasks early in the iteration
- When you encounter a problem start by letting the whole team know about it
- Bring it up by the next stand up meeting at the very latest
- This gives the entire team a chance to help solve the problem
- If the bug is small , you might be able to solve it in iteration slack
- Or else you can go for no critical refactoring, postponing a non essential meeting or even cancelling research time

Contd……

- We can also work n hour or so longer each day until it is resolved
- If the problem is big enough then bring that into stakeholders attention and product manger is the best person to decide who to talk and when
- Suppose you need a few more hours to finish a valuable story a little bit of overtime is fine

Organizational Strategy 4: Respect Customer goals

- When XP team first form, the programmers , customers often see themselves as separate group
- When we start a project programmers should make an extra effort to welcome the customers
- One way to do so is to treat customer goals with respect
- Another way is to come up with creative alternatives for meeting customer goals

Organizational Strategy 5: Promote the team

- You can also promote your team
- One team posted pictures and charts on the outer wall of workspace that showed what they were working on and how it was progressing
- Another team invited anyone and everyone in the company to attend their iteration demos

Organizational Strategy 6: Be Honest

- Be honest to stakeholders
- Don’t do any fraud – inform about all the problems your are facing in the project to stake holders if you are not able to stick in to the schedule

Sit Together

- We communicate rapidly and accurately
- Compared to teleconferences face to face conversations will be good

➢Accommodating Poor Communication

- As the distance between people grows the effectiveness of their communication decreases
- Misunderstanding occur and delays creep in
- To combat this problem most development methods attempt to reduce the need for direct communication
- The primary tool team use to reduce direct communication are development phase and work in progress document
- It’s sensible idea but it has flaws. It’s hard and impossible to anticipate all possible questions

A better way

- In XP, the whole team including experts in business, design , programming and testing sit together in open workspace
- When you have a question you need only turn your head and ask
- You get an instant response and if something isn’t clear , you can discuss it at the whiteboard

Exploiting Great Communication

- Sitting together eliminates the waste caused by waiting for an answer which improves productivity
- In XP a team spends far greater percentage of their time programming
- Teams that sit together not only get rapid answers , they experience what calls osmotic communication
- It helps team jell and breaks down us versus them attitude between groups

Secrets of Sitting Together

- Make sure that you have a complete team. If product manager fails in attending meeting in his place you can ask domain expert to answer the questions
- Sit close enough to each other so that you can have a quick discussion without greeting up from the desk
- In pair programming if we interrupt a team who is busy with the work the programmer will not be interrupted, the narrator will think and give the answer

Making Room

- Sitting together is easy to say and hard to do
- We have to find space for that
- A team that sits in adjacent cubicle can convert them into an adjacent shared workspace
- But even with cubicle it takes time and money to rearrange the wall
- Meanwhile we can use a big conference room as an alternative

Designing Your workspace

- Make sure there is a good sound insulation between your team workspace and rest of the organization
- Programmers should sit next to each other because they collaborate moment to moment
- Testers should be nearby so programmers can over hear them talk about issues
- Domain experts and interaction designers should not be quite so close, but should be close enough to answer questions without shouting
- Be sure that everyone has a space they can call their own. So also you need some cubes away from open space, so that people can have privacy for personal phone calls and individual meetings

Contd……

- In workspace include plenty of white boards and wall space for an informative workspace
- You can also have a projector in workspace
- The center of XP workspace should be a set of pairing stations
- Provide extra pair stations which can be used by testers and customers to pair

Adopting an open workspace

- Some team members resist moving to an open workspace
- Common concerns are loss of individuality and privacy
- Team members worry about distractions and noise
- Talk to your team members before going to an open workspace
- Try to discuss the advantage and benefits of it. If most of them disagree for it then leave that idea else go for it

Real Customer Involvement

- We understand the goals and frustrations of our customers and end users
- In XP team on-site customers are responsible for choosing and prioritising features
- The value of the project is in their hands
- The onsite customers can be real customers also

Personal Development

- In personal development the development team is its own customers
- They are developing software for their own use
- Here there is no need to involve external customer

In-House Custom Development

- In house custom development occurs when your organization asks your team to build something for the organization own use
- In this environment team has multiple customers to serve: the executive sponsor who pays for the software and end users who use the software
- In this environment make executive sponsor as product manager and some end users as domain experts

Outsourced Custom Development

- The software will be outsourced to some other agencies
- In this case the real customer cannot act as on-site customer
- One way to recruit on-site customer is to move your team to customer office
- If you can't being real customers onto your team make effort to meet them in person for the first week or two of the project
- If you're located near each other meet again for each iteration demo , retrospectives and planning session
- If you are far stay in touch with instant messaging and phone conferences
- Try to meet monthly once. If it is not feasible try meeting atleast once per release

Vertical Market Software

- Vertical market software is developed for many organizations
- Like custom development , it’s built for a particular industry and it’s often customised for each customer
- Vertical market software has multiple customers
- Your organization can appoint products manager who can understand the needs of real customer
- His job is to take into account all your real customers needs and combine them into single compelling vision
- You can also ask your customers to provide end users to join your team as onsite domain experts

Horizontal Market Software

- Horizontal market software is software that’s intended to be used across a wide range of industries
- Here also you can go for in house product manager
- To involve customer you can create focus groups , user experience testing, community previews, beta release etc

Ubiquitous Language

- We understand each other

➢The Domain Expertise Conundrum

- One challenge of professional software development is that programmers aren’t necessarily experts in the area for which they write software
- The people who are experts in the problem domain – the domain experts are rarely qualified to write software
- The people who are qualified to write software – the programmers –don’t always understand the problem domain
- The challenge here is communicating the information clearly and accurately

Two Languages

- Imagine you are driving to a job interview and your friend is guiding the way through map
- You will be talking about what you see in road and your friend will be talking about what he is seeing in map
- So you both are speaking two languages
- This will happen in case of programmers and domain experts
- So you have to pick one language for the whole team to communicate i.e ubiquitous language

How to speak the same language

- Programmers should speak the language of their domain experts
- Use domain terms instead of technical terms
- So both will understand and there will not be any misunderstanding

Ubiquitous language in code

- As programmer , it will be tough for you to speak language of domain expert
- Better approach to do it is you use your thr language of domain
- You can name your classes, methods and variables anything the terms thr domain expert use
- One powerful way to design your application to speak the language of the domain is to create a domain model

14. Additional topics beyond the syllabus

Not covered due to less time and pandemic.

15. University Question papers of previous years

R16

Code No: 138DK

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
B. Tech IV Year II Semester Examinations, December - 2020
MODERN SOFTWARE ENGINEERING
(Common to CSE, IT)

Time: 2 Hours **Max. Marks: 75**

Answer any Five Questions
All Questions Carry Equal Marks

- - -

1. State and explain the need of Agile methodology process mode. [15]

2. Discuss about Practicing XP and Root cause analysis for extreme programming. [15]

3. Explain about coding standards and collective ownership in software process development model. [15]

4. Discuss the role of collaboration in software development, during iteration demo and reporting process. [15]

5. Discuss the significance of collective ownership. Explain the importance of Real customer involvement. [15]

6. Explain about Risk management and estimation. [15]

7. Discuss about spike solutions and Performance optimization. [15]

8. Explain about Incremental design and architecture. [15]

---ooOoo---

Code No: 138DK **R16**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

B. Tech IV Year II Semester Examinations, September - 2020

MODERN SOFTWARE ENGINEERING

(Common to CSE, IT)

Time: 2 Hours **Max. Marks: 75**

Answer any Five Questions

All Questions Carry Equal Marks

- - -

1. Appraise the importance of organizational success. [15]

2.a) How to hold a daily standup meeting?
b) What is iteration demo? How does it help the team? [7+8]

3.a) How to prevent security defects and other challenging bugs?
b) Code goes through four levels of completion. Discuss these levels. [8+7]

4.a) Discuss ways to introduce slack into iterations.
b) Explain combining stories with illustration. [7+8]

5. Explain the life cycle of test driven development with suitable example. [15]

6. XP teams are self organized. Support this statement. [15]

7.a) Contrast in-house custom development with outsourced custom development.
b) How to deal with disagreement regarding coding standards in the team? [8+7]

8. Is asynchronous integration more efficient than synchronous integration? Substantiate your answer. [15]

---ooOoo---

16. Question bank, Unit wise Quiz Questions

1. We have a configuration management (CM) department that's responsible for maintaining our builds.
2. The ultimate goal of continuous integration is to be able to deploy all but the last few hours of work at any time.
3. There's a lively community of open-source *continuous integration servers* (also called *CI servers*).

4. Synchronous integration reduces integration problems.

5. Collective code ownership spreads responsibility for maintaining the code to all the programmers. Collective code ownership is exactly what it sounds like: everyone shares responsibility for the quality of the code. No single person claims ownership over any part of the system, and anyone can make any necessary changes anywhere.

6. Always leave the code a little better than you found it.
7. Continuous integration decreases the chances of merge conflicts.
8. XP practices support work-in-progress communication in other ways—ways that are actually more effective than written documentation.
9. Alistair Cockburn describes a variant of Extreme Programming called "Pretty Adventuresome Programming":
10. Vision reveals where the project is going and why it's going there.
11. Release Planning provides a roadmap for reaching your destination.
12. The Planning Game combines the expertise of the whole team to create achievable plans.
13. Risk Management allows the team to make and meet long-term commitments.
14. Iteration Planning provides structure to the team's daily activities.
15. Slack allows the team to reliably deliver results every iteration.
16. Stories form the line items in the team's plan.
17. Estimating enables the team to predict how long its work will take.

18. One person gets a bright idea, evangelizes it, and gets approval to pursue it. This person is a visionary.

19. Frequent releases are good for the organization. Frequent releases can actually make your life easier. By delivering tested, working, valuable software to your stakeholders regularly, you increase trust.

20. To take the most advantage of the opportunities you create, build a plan that allows you to release at any time. At any time, you should be able to release a product that has value proportional to the investment you've made.
21. Some people try to fix the release date and features. This can only end in tears; given the uncertainty and risk of software development, making this work requires adding a huge amount of padding to your schedule, sacrificing quality, working disastrous amounts of overtime, or all of the above.
22. "Done done" applies to release planning as well as to stories. Just as you shouldn't postpone tasks until the end of an iteration, don't postpone stories until the end of a release. Every feature should be "done done" before you start on the next feature.
23. Risk management allows you to make and meet commitments.
24. One of the hardest things about project-specific risks is remembering to follow up on them.
25. As you evaluate your risks, think about the risk to the success of the project, not just the risk to the schedule.
26. Every team member is responsible for the successful delivery of the iteration's stories.
27. The amount of slack you need doesn't depend on the number of problems you face.
28. One way to introduce slack into your iterations might be to schedule no work on the last day or two of your iteration.
29. Continue refactoring new code as you write it. It's OK to defer cleaning up existing technical debt temporarily, but incurring new technical debt will hurt your productivity.
30. Slack is a wonderful tool. It helps you meet your commitments and gives you time to perform important, nonurgent tasks that improve your productivity.

31. Stories are for planning. They're simple one- or two-line descriptions of work the team should produce. Alistair Cockburn calls them "promissory notes for future conversation."* Everything that stakeholders want the team to produce should have a story.
32. A good way to ensure that your stories are customer-centric is to ask your customers to write the stories themselves.
33. Select story sizes such that you complete 4 to 10 each iteration.
34. Splitting stories is more difficult because it tempts you away from vertical stripes and releasable Stories

35. Bug stories can be difficult to estimate. Often, the biggest time sink in debugging is figuring out what's wrong, and you usually can't estimate how long that will take.

36. Programmers will often use a spike solution to research the technology, so these sorts of stories are typically called spike stories.
37. We provide reliable estimates. Programmers often consider estimating to be a black art—one of the most difficult things they must do.
38. One reason estimating is so difficult is that programmers can rarely predict how they will spend their time. A task that requires eight hours of uninterrupted concentration can take two or three days if the programmer must deal with constant interruptions.
39. Although estimates are almost never accurate, they are consistently inaccurate. While the estimate accuracy of individual estimates is all over the map—one estimate might be half the actual time, another might be 20 percent more than the actual time—the estimates are consistent in aggregate.
40. Estimate in terms of ideal engineering days (story points), not calendar time.
41. Incremental Requirements allows the team to get started while customers work out requirements details.
42. Customer Tests help communicate tricky domain rules.
43. Test-Driven Development allows programmers to be confident that their code does what they think it should.
44. Refactoring enables programmers to improve code quality without changing its behavior.
45. Simple Design allows the design to change to support any feature request, no matter how surprising.

46. In incremental requirements we define requirements in parallel with other work.
47. Sometimes the best way to create a UI mock-up is to work in collaboration with the programmers. The iteration-planning meeting might be the best time for this work.
48. Test-driven development, or TDD, is a rapid cycle of testing, coding, and refactoring.
49. Every few minutes, TDD provides proven code that has been tested, designed, and coded.
50. Unit tests focus just on the class or method at hand. They run entirely in memory, which makes them very fast. Depending on your platform, your testing tool should be able to run at least 100 unit tests per second.
51. *Mock objects* are a popular tool for isolating classes for unit testing.
52. A spike solution is a technical investigation. It's a small experiment to research the answer to a problem.
53. We optimize when there's a proven need.
54. Performance optimizations must serve the customer's needs.
55. Throughput : is how many operations should complete in a given period of time?
56. Latency: is how much delay is acceptable between starting and completing a single operation?

57. Responsiveness: is How much delay is acceptable between starting an operation and receiving feedback about that operation

58. Exploratory testing can be done manually or with the assistance of automation. Its defining characteristic is not how we drive the software but rather the tight feedback loop between test design, test execution, and results interpretation.
59. Optimization has two major drawbacks: it often leads to complex, buggy code, and it takes time away from delivering features. Neither is in your customer's interests. Optimize only when it serves a real, measurable need.
60. Performance optimization can consume an infinite amount of time.
61. Exploratory testing can be done manually or with the assistance of automation.
62. Exploratory testing works best when the software is ready to be explored— that is, when stories are "done done."
63. We have a configuration management (CM) department that's responsible for maintaining our builds.
64. The ultimate goal of continuous integration is to be able to deploy all but the last few hours of work at any time.
65. There's a lively community of open-source *continuous integration servers* (also called *CI servers*).

66. Synchronous integration reduces integration problems.

67. Collective code ownership spreads responsibility for maintaining the code to all the programmers. Collective code ownership is exactly what it sounds like: everyone shares responsibility for the quality of the code. No single person claims ownership over any part of the system, and

anyone can make any necessary changes anywhere.

68. Always leave the code a little better than you found it.
69. Continuous integration decreases the chances of merge conflicts.
70. XP practices support work-in-progress communication in other ways—ways that are actually more effective than written documentation.
71. Alistair Cockburn describes a variant of Extreme Programming called "Pretty Adventuresome Programming":
72. Vision reveals where the project is going and why it's going there.
73. Release Planning provides a roadmap for reaching your destination.
74. The Planning Game combines the expertise of the whole team to create achievable plans.
75. Risk Management allows the team to make and meet long-term commitments.
76. Iteration Planning provides structure to the team's daily activities.
77. Slack allows the team to reliably deliver results every iteration.
78. Stories form the line items in the team's plan.
79. Estimating enables the team to predict how long its work will take.

80. One person gets a bright idea, evangelizes it, and gets approval to pursue it. This person is a visionary.

81. Frequent releases are good for the organization. Frequent releases can actually make your life easier. By delivering tested, working, valuable software to your stakeholders regularly, you increase trust.

82. To take the most advantage of the opportunities you create, build a plan that allows you to release at any time. At any time, you should be able to release a product that has value proportional to the investment you've made.
83. Some people try to fix the release date and features. This can only end in tears; given the uncertainty and risk of software development, making this work requires adding a huge amount of padding to your schedule, sacrificing quality, working disastrous amounts of overtime, or all of the above.
84. "Done done" applies to release planning as well as to stories. Just as you shouldn't postpone tasks until the end of an iteration, don't postpone stories until the end of a release. Every feature should be "done done" before you start on the next feature.
85. Risk management allows you to make and meet commitments.
86. One of the hardest things about project-specific risks is remembering to follow up on them.
87. As you evaluate your risks, think about the risk to the success of the project, not just the risk to the schedule.
88. Every team member is responsible for the successful delivery of the iteration's stories.
89. The amount of slack you need doesn't depend on the number of problems you face.
90. One way to introduce slack into your iterations might be to schedule no work on the last day or two of your iteration.
91. Continue refactoring new code as you write it. It's OK to defer cleaning up existing technical debt temporarily, but incurring new technical debt will hurt your productivity.
92. Slack is a wonderful tool. It helps you meet your commitments and gives you time to perform important, nonurgent tasks that improve your productivity.

93. Stories are for planning. They're simple one- or two-line descriptions of work the team should produce. Alistair Cockburn calls them "promissory notes for future conversation."* Everything that stakeholders want the team to produce should have a story.
94. A good way to ensure that your stories are customer-centric is to ask your customers to write the stories themselves.
95. Select story sizes such that you complete 4 to 10 each iteration.
96. Splitting stories is more difficult because it tempts you away from vertical stripes and releasable Stories
97. Bug stories can be difficult to estimate. Often, the biggest time sink in debugging is figuring out what's wrong, and you usually can't estimate how long that will take.

98. Programmers will often use a spike solution to research the technology, so these sorts of stories are typically called spike stories.
99. We provide reliable estimates. Programmers often consider estimating to be a black
art—one of the most difficult things they must do.
100. One reason estimating is so difficult is that programmers can rarely predict how they will spend their time. A task that requires eight hours of uninterrupted concentration can take two or three days if the programmer must deal with constant interruptions.

101. Although estimates are almost never accurate, they are consistently inaccurate. While the estimate accuracy of individual estimates is all over the map—one estimate might be half the actual time, another might be 20 percent more than the actual time—the estimates are consistent in aggregate.
102. Estimate in terms of ideal engineering days (story points), not calendar time.
103. Incremental Requirements allows the team to get started while customers work out requirements details.
104. Customer Tests help communicate tricky domain rules.
105. Test-Driven Development allows programmers to be confident that their code does what they think it should.
106. Refactoring enables programmers to improve code quality without changing its behavior.
107. Simple Design allows the design to change to support any feature request, no matter how surprising.

108. In incremental requirements we define requirements in parallel with other work.
109. Sometimes the best way to create a UI mock-up is to work in collaboration with the programmers. The iteration-planning meeting might be the best time for this work.
110. Test-driven development, or TDD, is a rapid cycle of testing, coding, and refactoring.
111. Every few minutes, TDD provides proven code that has been tested, designed, and coded.
112. Unit tests focus just on the class or method at hand. They run entirely in memory, which makes them very fast. Depending on your platform, your testing tool should be able to run at least 100 unit tests per second.
113. *Mock objects* are a popular tool for isolating classes for unit testing.
114. A spike solution is a technical investigation. It's a small experiment to research the answer to a problem.
115. We optimize when there's a proven need.
116. Performance optimizations must serve the customer's needs.
117. Throughput : is how many operations should complete in a given period of time?
118. Latency: is how much delay is acceptable between starting and completing a single operation?

119. Responsiveness: is How much delay is acceptable between starting an operation and receiving feedback about that operation

120. Exploratory testing can be done manually or with the assistance of automation. Its defining characteristic is not how we drive the software but rather the tight feedback loop between test design, test execution, and results interpretation.
121. Optimization has two major drawbacks: it often leads to complex, buggy code, and it takes time away from delivering features. Neither is in your customer's interests. Optimize only when it serves a real, measurable need.
122. Performance optimization can consume an infinite amount of time.
123. Exploratory testing can be done manually or with the assistance of automation.
124. Exploratory testing works best when the software is ready to be explored— that is, when stories are "done done."

DESCRIPTIVE QUESTIONS:

UNIT-I

Short Answer Questions-

S.NO	QUESTION	BLOOMS Taxonomy
1.	Define: Successful, Challenged, Impaired	**L1: REMEMBER**
2.	Define : Method and Agile Method, Refactoring	**L1: REMEMBER**
3.	What is Iteration Planning?	**L1: REMEMBER**
4.	What is the Role of Onsite Customers?	**L1: REMEMBER**
5.	What is the role of Product Manager?	**L1: REMEMBER**
6.	What is Time Boxing, Iteration, and Velocity?	**L1: REMEMBER**
7.	How to energies the work in Agile	**L1: REMEMBER**
8.	Define Informative Workspace	**L1: REMEMBER**
9.	What is Root - Cause Analysis?	**L1: REMEMBER**
	10. Define Retrospectives	**L1: REMEMBER**

Long Answer Questions-

S.NO	QUESTION	BLOOMS Taxonomy
1.	Explain in detail about Organization, Technical, Personal	**L2:UNDERSTAND**
2.	Explain the Principles of Agile Development	**L2:UNDERSTAND**
3.	Distinguish Traditional S/w Life cycle and Agile Life Cyle	**L4:ANALYZING**
4.	Explain the pre-requisite of adopting the XP[Extreme Programming]	**L2:UNDERSTAND**
5.	Explain the Assessment of Agility	**L2:UNDERSTAND**
6.	Explain the Tips for pairing	**L2:UNDERSTAND**
7.	Explain the process improvement chart with examples	**L2:UNDERSTAND**
8.	Explain in detail about root-cause analysis	**L2:UNDERSTAND**

UNIT-2

Short Answer Questions

S.NO	QUESTION	BLOOMS Taxonomy
1.	What are the 8 peaches that help a team and its stakeholder collaborate?	**L1: REMEMBER**
2.	What do you mean by collaborating?	**L1: REMEMBER**
3.	What is the group dynamics involved when people work through team?	**L1: REMEMBER**
4.	What are stand up meeting?	**L1: REMEMBER**
5.	What do you mean by coding standards?	**L1: REMEMBER**

Long Answer Questions-

S.NO	QUESTION	BLOOMS Taxonomy
1.	Explain the steps involved in collaborating	**L2:UNDERSTAND**
2.	Elaborate the strategies for generating trust	**L6: CREATE**
3.	Discuss in detail about the organizational strategies for maintain impressions.	**L6: CREATE**
4.	Criticize the daily stand up meeting	**L5:Evaluate**
5.	Demonstrate the Iteration Demo Process	**L2:UNDERSTAND**

UNIT-3

Short Answer Questions-

S.NO	QUESTION	BLOOMS Taxonomy
1.	Define Releasing.	L1: REMEMBER
2.	How to release No bug software?	L1: REMEMBER
3.	Define Version Control and list the terminologies used in it.	L1: REMEMBER
4.	Define Releasing Documentation.	L1: REMEMBER
5.	Compare the build a project and automate the build	L5:EVALUATE

Long Answer Questions-

S.NO	QUESTION	BLOOMS Taxonomy
1.	Explain in detail of production - Ready software	L2:UNDERSTAND
2.	Illustrate how to achieve nearly zero bugs.	L2:UNDERSTAND
3.	Explain in detail about continuous integration	L2:UNDERSTAND
4.	Demonstrate collective code ownership	L2:UNDERSTAND
5.	Examine the Documentation	L4:ANALYZING

UNIT-4

Short Answer Questions-

S.NO	QUESTION	BLOOMS Taxonomy
1.	What is product vision and how to identify the vision?	L1: REMEMBER
2.	Distinguish release early and release often	L6: CREATE
3.	What do you mean my adaptive planning?	L1: REMEMBER
4.	Define Risk Management	L1: REMEMBER
5.	Assess the iteration planning	L5:EVALUATE

Long Answer Questions-

S.NO	QUESTION	BLOOMS Taxonomy
1.	Discuss the vision statement & promote it to stakeholder	L6: CREATE
2.	Explain the method to create a release plan	L2:UNDERSTAND
3.	Design the strategy for Game and Play to win	L6: CREATE
4.	Discuss how can we make a release commitment?	L6: CREATE
5.	Explain in detail about estimation and velocity	L2:UNDERSTAND

UNIT-5

Short Answer Questions-

S.NO	QUESTION	BLOOMS Taxonomy
1.	Define Customer Review Questions	L1: REMEMBER
2.	What do you mean by Multiple name / Value pair?	L1: REMEMBER
3.	What is unit test?	L1: REMEMBER
4.	Define refactoring	L1: REMEMBER
5.	What do you mean by Risk - Driven Architecture	L1: REMEMBER

Long Answer Questions-

S.NO	QUESTION	BLOOMS Taxonomy
1.	Explain in detail about customer Test	L2:UNDERSTAND
2.	Explain in detail about customer TDD	L2:UNDERSTAND

3.	Write short notes on (i) Focused Integration Test (ii) End to End Test	**L2:UNDERSTAND**
4.	Discuss the Effective Designing	**L6: CREATE**
5.	Explain in detail about Incremental design and Architecture	**L2:UNDERSTAND**

OBJECTIVE & FILL IN THE BLANKS QUESTIONS

UNIT -1

1)------------------Appreciate the team's focus on providing a solid return on investment and the software's longevity.

a) Users b) stakeholders c) domain expert d**) domain experts**

2. -----------will appreciate their ability to change direction as business needs change, the team's ability to make and meet commitments, and improved stakeholder satisfaction.

a) Users b) **project managers** c) domain expert

3. ------------will appreciate their integration as first-class members of the team, their ability to influence quality at all stages of the project, and more challenging, less repetitious work.

a) **Testers** b) project managers c) domain expert

4. -----------help the team work with the rest of the organization. They are usually good at coaching nonprogramming practices

a) **Testers** b) project managers **c) project engineer** d**)** domain experts

5. ---------------is the visible tip of the software development iceberg.

a)Horizontal-market software b)web based software c)both d)none

Fill in the blanks

1. User requirements are expressed as ------------------ in Extreme Programming
2. Tests are automated in Extreme Programming.---------------------.
3. Developers work individually on a release and they compare their results with other developers before forwarding that release to customers.
4. In XP Increments are delivered to customers every _______ weeks.
5. How many documents in the vision statement **-------------**

Answers : fill in the blanks Answers

1. Scenario 2.True 3.false 4.2 Weeks 5.Three

UNIT -2

1----------------is essential for the team to thrive.

a)**Trust** b)sitting c)involvement d)language

2------------------ together leads to fast, accurate communication.

a)Trust **b)sitting** c)involvement d)language

3. -------------------helps the team understand what to build.

a) **Real customer involvement** b) sitting c)involvement d)language

4) A --------------------language helps team members understand each other.

a)c language **b) ubiquitous** c)involvement d)language

5. ----------standards provide a template for seamlessly joining the team's work together.

a)coding b)testing c)design d)SRS

Fill in the blanks

1. In software engineering, defects that are discovered ______ are ______ to fix
2.----------------------is called as agile model
3.-------------------- helps reassure the organization that the team is working well.
4.------------------------ keep the team's efforts aligned with stakeholder goals
5.------------ provide high-level information that allows management to analyze trends and set goals

Answers:
1)later; more expensive 2. Customer collaboration over contract negotiation
3) Reporting 4) Iteration 5)management reports

UNIT -3

1. Select the option that suits the Manifesto for Agile Software Development
   a) Individuals and interactions
   b) Working software
   c) Customer collaboration
   d) All of the mentioned
2. Agile Software Development is based on
   a) Incremental Development
   b) Iterative Development
   c) Linear Development
   d) Both Incremental and Iterative Development
3. How many phases are there in Scrum ?
   a) Two
   b) Three
   c) Four
   d) Scrum is an agile method which means it does not have phases
4. How is plan driven development different from agile development ?
   a) Outputs are decided through a process of negotiation during the software development process
   b) Specification, design, implementation and testing are interleaved
   c) Iteration occurs within activities
   d) All of the mentioned
5. Which of the following does not apply to agility to a software process?
   a) Uses incremental product delivery strategy
   b) Only essential work products are produced
   c) Eliminate the use of project planning and testing
   d) All of the mentioned

Fill in the blanks

1In agile development it is more important to build software that meets the customers' needs today than worry about features that might be needed in the future is ------------
2. ----- test do you infer from the following statement: "The coordination and data management functions of the server are tested."?
3. A client is assigned all user presentation tasks and the processes associated with data entry". Which option supports the client's situation-------------?
4) -------------enables a software engineer to defined screen layout rapidly for interactive applications.
5. -------------------tools are used to modify online database systems

Answers fill in the blanks

1. True 2.Server test 3.Distributed logic 4.Screen painter's 5.online reengineering tools

UNIT 4

1. ------------------ is not a conflict in software development team?
a) Simultaneous updates
b) Shared and common code
c) Versions
d) Graphics issues

2. Which of the following is not a typical environment in communication facilitation?
a) Multiple teams
b) Multiple user groups
c) Multiple fests
d) Multiple locations

3. Which of the following is not a part of Software Configuration Management Basics?
a) Identification
b) Version
c) Auditing and Reviewing
d) Status Accounting

4. What is one or more software configuration items that have been formally reviewed and agreed upon and serve as a basis for further development?
a) Configuration
b) Baseline
c) Software
d) All of the mentioned

5. Why is software difficult to build ?
a) Controlled changes
b) Lack of reusability
c) Lack of monitoring
d) All of the mentioned

Fill in the blanks:

1) ----------------- is a specific instance of a baseline or configuration item?

2. ITG stands for----------------------------

3 Which one is not a software quality model?
a) ISO 9000
b) McCall model
c) Boehm model
d) ISO 9126

4. IMC Networks is a leading ________ certified manufacturer of optical networking and LAN/WAN connectivity solutions for enterprise, telecommunications and service provider applications.

5. Software reliability is defined with respect to---------------------

Answers 1.version 2.independent test group 3.IS000 4.telco systems 5.time

UNIT 5

1. The team is unable to decide whether it makes sense to buy an off-the-shelf from the vendor or go about building it themselves. Both options have its merits and demerits. As a Scrum Master what would be your recommendation to the team? -------------------------

a). Consult with the product owner of what he is willing to sponsor.

b) Conduct a spike to evaluate both options.

c)Do a fist of five voting.

d) None of the above.

2. ________ is a low-fidelity prototype that shows a mockup for a set of screen, containing the basic layout of the different widgets on it

a)Persona

b)Wireframe

c)Spikes

d)Story map

3. If you happen to hire for a new Agile team, you should prefer:

a) Developers

b) Specialists in the technologies to be used

c)Generalists with cross-functional skillsets

d) People who exhibit adaptive leadership skill

4. During which Scrum ceremony are risk audits held?

a) Sprint planning **b)Sprint execution** c)Sprint review d)Sprint retrospective

5. By tracking velocity trends, a team can---------

a**). Gauge the rate of progress** b)Estimate how much longer it will take to complete c)Correcting estimation errors d) All of the above.

Fill in the blanks :

1.The pillars of Scrum are-----------and----------------------

2. XP teams use the technique of ________ to enhance code quality, while keeping its behavior unchanged.

3. During which Scrum ceremony are risk audits held?-----------------------------

4. The Y-axis of an iteration burndown chart depicts ----------------------

5. The pillars of Scrum are---------------------------

Answers

1. Transparency, Inspection 2.spikes 3.sprint execution 4.no of features to be compleraed 5. Transparency, Inspection and Adaptation

17. Assignment Questions

Modern Software Engineering

Assignment-1

Explain about the following:

1) Agile software development
2) Principles of Agile
3) Successful, Challenged and Impaired
4) Values of Agile
5) XP lifecycle
6) Refactoring
7) Timeboxing
8) Stories
9) Agile requirements
10) Definition of Done

Modern Software Engineering

Assignment-2

1) Pair Programming
2) Mob programming
3) Test-driven development
4) Refactoring legacy code
5) Scrum and Scrum practices
6) Continuous Integration
7) Continuous Delivery
8) Real customer involvement
9) Behavior-driven development
10) Collaborating

20. Known gaps ,if any

18. Mid Wise Question Paper including Quiz

Nawab Shah Alam Khan College of Engineering and Technology
Modern Software Engineering
B.Tech IV year I semester 2020-21
Quiz-I Date: 5-5-2021

1. ______ means effective (rapid and adaptive) response to change
2. Drawing the _______ into the team to eliminate us and them attitude.
3. ____ development emphasizes an incremental delivery strategy.
4. Agile development is also known as _____________
5. Pair programming consists of two programmers sharing a single workstation (one screen, keyboard and mouse among the pair). The programmer at the keyboard is usually called the __________ the other, also actively involved in the programming task but focusing more on overall direction is the __________.

6. ____________ is often neglected by software teams in favor of the more easily achieved technical and personal successes.
7. First-class members of the team are
a) testers b) developers c) project managers d) stakeholders
8. To be _______ you need to put the agile 4 values and 12 principles into practice.

9. ___________ satisfaction by early and continuous delivery of valuable software.
10. Agile development welcomes changing requirements, even in ________ development.
11. ______ is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. [Score]
12. Self-organization is a hallmark of ______ teams. [Score]
13________________is an activity that inextricably weaves together testing, coding, design, and architecture. [Score]
14. _________ is the process of changing the structure of code—rephrasing it—without changing its meaning or behavior. It’s used to improve code quality, to fight off software’s unavoidable entropy, and to ease adding new features.
15. The best architectures, requirements and emerge from _________ team. [Score]
16. Of all the on-site customers, the ___________is likely the most important. He/ She makes the final determination of value.

CO1

17. An ____ retrospective, or sprint retrospective as Scrum calls it, is a practice used by teams to reflect on their way of working, and to continuously become better in what they do.
18. _______is essential for the team to thrive.
19. Sitting together leads to fast, _________ communication.
20. Real ____ involvement helps the team understand what to build.
21. A ___________ language helps team members understand each other.
22. ___ meetings keep team members informed.
23. ________ standards provide a template for seamlessly joining the teamâ€™s work together.
24. ____________ demos keep the team’s efforts aligned with stakeholder goals.
25. ___________ helps reassure the organization that the team is working well.
26.________ continuity is an advanced practice—not because it’s hard to do, but because it

challenges normal organizational structures. While team continuity is valuable, you don’t need to do it to be successful. 27. Adding manpower to a late software project makes it____. 28. In __________, the whole team including experts in business, design, programming, and testing sits together in an open workspace. 29. Teams that sit together not only get rapid answers to their questions, they experience what calls ________ communication. 30. You not only hear your name, you hear a bit of the conversation around it, too, in a phenomenon known as the ______ party effect. 31._____________ and its language-independent cousin, function points, are common approaches to measuring software size. Unfortunately, they are also used for measuring productivity. 32. Measuring the variation in _______ may produce interesting information for discussion in the retrospective, but the information is too ambiguous to report outside the team. 33. ________ shows that the more lines of code a program has, the more defects it gets. 34. More lines of code is likely to have defects and the more it will ______ to develop. 35. ______ control allows team members to work together without stepping on each others toes. 36. __________integration prevents a long, risky integration phase. 37. A _______ or defect is any behaviour of your software that will unpleasantly surprise important stakeholders. 38. Exploratory _______ is a very effective way of finding unexpected bugs. It’s so effective that the rest of the team might start to get a little lazy. 39. When you produce nearly zero bugs, you are confident in the _________ of your software. 40. ____________means delivering value to the organization.	CO2

Modern Software Engineering
B.Tech IV year I semester 2020-21
Quiz-II Date: 3-6-2021

125. We have a configuration management (CM) department that's responsible for maintaining our builds.

126. The ultimate goal of continuous integration is to be able to deploy all but the last few hours of work at any time.

127. Synchronous integration reduces integration problems.

128. Continuous integration decreases the chances of merge conflicts.

129. Vision reveals where the project is going and why it's going there.

130. Release Planning provides a roadmap for reaching your destination.

131. Risk Management allows the team to make and meet long-term commitments.

132. Iteration Planning provides structure to the team's daily activities.

133. Slack allows the team to reliably deliver results every iteration.

134. Stories form the line items in the team's plan.

135. Estimating enables the team to predict how long its work will take.

136. Frequent releases are good for the organization. Frequent releases can actually make your life easier. By delivering tested, working, valuable software to your stakeholders regularly, you increase trust.

137. "Done done" applies to release planning as well as to stories. Just as you shouldn't postpone tasks until the end of an iteration, don't postpone stories until the end of a release. Every feature should be "done done" before you start on the next feature.

CO3

138. Risk management allows you to make and meet commitments.

139. Every team member is responsible for the successful delivery of the iteration's stories.

140. Slack is a wonderful tool. It helps you meet your commitments and gives you time to perform important, nonurgent tasks that improve your productivity.

141. Stories are for planning. They're simple one- or two-line descriptions of work the team should produce. Alistair Cockburn calls them "promissory notes for future conversation."* Everything that stakeholders want the team to produce should have a story.

142. Bug stories can be difficult to estimate. Often, the biggest time sink in debugging is figuring out what's wrong, and you usually can't estimate how long that will take.

143. Programmers will often use a spike solution to research the technology, so these sorts of stories are typically called spike stories.

144. Estimate in terms of ideal engineering days (story points), not calendar time.

145. Incremental Requirements allows the team to get started while customers work out requirements details.

146. Customer Tests help communicate tricky domain rules.

147. Test-Driven Development allows programmers to be confident that their code does what they think it should.

148. Refactoring enables programmers to improve code quality without changing its behavior.

149. Simple Design allows the design to change to support any feature request, no matter how surprising. 150. In incremental requirements we define requirements in parallel with other work. 151. Test-driven development, or TDD, is a rapid cycle of testing, coding, and refactoring. 152. Unit tests focus just on the class or method at hand. They run entirely in memory, which makes them very fast. Depending on your platform, your testing tool should be able to run at least 100 unit tests per second. 153. *Mock objects* are a popular tool for isolating classes for unit testing. 154. A spike solution is a technical investigation. It's a small experiment to research the answer to a problem. 155. We optimize when there's a proven need. 156. Performance optimizations must serve the customer's needs. 157. Throughput : is how many operations should complete in a given period of time? 158. Latency: is how much delay is acceptable between starting and completing a single operation? 159. Responsiveness: is How much delay is acceptable between starting an operation and receiving feedback about that operation 160. Exploratory testing can be done manually or with the assistance of automation. Its defining characteristic is not how we drive the software but rather the tight feedback loop between test design, test execution, and results interpretation. 161. Optimization has two major drawbacks: it often leads to complex, buggy code, and it takes time away from delivering features. Neither is in your customer's interests. Optimize only when it serves a real, measurable need. 162. Performance optimization can consume an infinite amount of time. 163. Exploratory testing can be done manually or with the assistance of automation. 164. Exploratory testing works best when the software is ready to be explored— that is, when stories are "done done."	CO4 & CO5

19. Tutorial problems

None

20. Known gaps ,if any

21. Discuss topic if any

22. References, Journals, websites and E-links if any

WEBSITES

1. http://www.agiledeveloper.com/downloads.html
2. https://www.sanfoundry.com/software-engg-mcqs-extreme-programming/
3. http://mcqspdfs.blogspot.com/2016/06/100-top-agile-testing-multiple-choice.html

JOURNALS

1. http://www.123seminarsonly.com/Seminar-Reports/002/50486044-what-is-agile-software-development.pdf
2. https://www.ijitee.org/
3. http://www.jardcs.org/

23. Attainments

NAWAB SHAH ALAM KHAN COLLEGE OF ENGINEERING AND TECHNOLOGY, JNTUH Hyderabad
DEPARTMENT OF INFORMATION TECHNOLOGY
B.Tech.IV YEAR, II SEM - ATTAINMENT CALCULATIONS - Academic Year: 2020-21

ubject: **MODERN SOFTWARE ENGINEERING** Subject Code: **138DK** Faculty: **Mr.Q.M.A BASHEER**

		M I D - 1											M I D - 2											MID		SEE
S.No.	Hall Ticket No.	ASG-1 (2.5M)	ASG-2 (2.5 M)	Quiz-1 (20 M)								Mid-1 TOTAL	ASG-3 (2.5M)	ASG-4 (2.5 M)	Quiz-2 (10 M)		Q 1 (5 M)	Q 2 (5 M)	BEST OF Q1&Q2	Q 3 (5 M)	Q 4 (5 M)	BEST OF Q3&Q4	Mid-2 TOTAL	Average MID	TOTAL Marks	End Exam
		C01	C02	C01	C02							(25 M)	C03	C04	C03	C04	C03	C03	C03	C04	C04	C04	(25 M)	(25 M)	(100 M)	(75 M)
1	15RT1A1224	2.5	2.5	9	10							24	2.5	2.5	5	5	5		5	5		5	25	25	37	12
2	15RT1A1228	2.5	2.5	10	10							25	2.5	2.5	5	5		5	5	5		5	25	25	20	5
3	15RT1A1237	2	2	8	9							21	2.5	2.5	4	3	4		4			0	16	19	20	1
4	15RT1A1241	2.5	2.5	10	9							24	2	2	4	3		3	3		3	3	17	21	46	40
5	16RT1A1202	2	2	10	10							24	2.5	2.5	5	5		5	5	5		5	25	25	32	21
6	16RT1A1205	2	2	6	6							16	2.5	2.5	4	3		3	3	3		3	18	17	31	29
7	16RT1A1232	2.5	2.5	10	10							25	2.5	2.5	5	5		5	5		5	5	25	25	36	1
8	16RT1A1244	2.5	2.5	8	8							21	2.5	2.5	4	3		4	4	2		2	18	20	49	30
9	16RT1A1248	2	2	9	9							22	2	2	4	4	5		5	2		2	19	21	27	13
10	17RT1A1201	2	2	10	10							24	2	2	4	5		2	2	2		2	17	21	34	31
11	17RT1A1205	2.5	2.5	10	10							25	2.5	2.5	4	5		5	5		5	5	24	25	46	43
12	17RT1A1210	2.5	2.5	10	10							25	2.5	2.5	5	5		5	5	5		5	25	25	46	19
13	17RT1A1211	2	2	8	8							20	2	2	4	3	3		3	3		3	17	19	22	31
14	17RT1A1218	2	2	7	8							19	2.5	2.5	5	5		5	5	5		5	25	22	31	35
15	17RT1A1224	2.5	2.5	9	10							24	2.5	2.5	4	4		5	5	5		5	23	24	70	42
16	17RT1A1228	2	2	5	6							15	2.5	2.5	4	4		5	5	4		4	22	19	53	41
17	17RT1A1230	2.5	2.5	6	7							18	2.5	2.5	4	4		5	5		4	4	22	20	50	34
18	17RT1A1233	2	1	7	7							17	2.5	2.5	4	3	5		5		2	2	19	18	26	18
19	17RT1A1235	2.5	2.5	8	10							23	2	2	4	3		5	5	5		5	21	22	29	15
20	17RT1A1236	2.5	2.5	9	10							24	2.5	2.5	5	5		5	5	5		5	25	25	49	18
21	17RT1A1237	2.5	2.5	10	10							25	2.5	2.5	4	3		5	5		2	2	19	22	27	41
22	17RT1A1239	2	2	10	9							23	2.5	2.5	5	5		5	5	4		4	24	24	32	28
23	17RT1A1240	2.5	2.5	8	9							22	2.5	2.5	4	3		5	5	5		5	22	22	30	38
24	17RT1A1242	2.5	2.5	7	6							18	2	2	4	4		5	5	5		5	22	20	67	29
25	17RT1A1244	2.5	2.5	6	7							18	2	2	4	5	5		5	5		5	23	21	35	32
26	17RT1A1245	2.5	2.5	9	10							24	2.5	2.5	4	5	5		5	5		5	24	24	86	42
27	17RT1A1246	2	2	10	10							24	2.5	2.5	5	5		4	4	4		4	23	24	30	32
28	17RT1A1250	2.5	2.5	5	5							15	2	2	4	3		5	5	5		5	21	18	31	47
29	17RT1A1252	2.5	2.5	9	10							24	2.5	2.5	5	5	4		4	3		3	22	23	45	30
30	17RT1A1255	2	2	10	10							24	2.5	2.5	4	4		5	5		4	4	22	23	53	28
31	17RT1A1257	2.5	2.5	10	9							24	2.5	2.5	4	4		4	4	5		5	22	23	47	54
32	17RT1A1259	2.5	2.5	7	7							19	2.5	2.5	4	4		4	4	2		2	19	19	33	35
Average Marks		2.31	2.28	8.44	8.72							21.75	2.38	2.38	4.31	4.13	4.50	4.54	4.53	4.13	3.57	3.88	21.59	21.67	39.67	28.59

CIE (Mid Exam) CO Wise Percentage

COURSE OUTCOME	CO Wise Sum	CO Wise Percentage %
C01	10.75	86.00
C02	11.00	88.00
C03	11.22	89.75
C04	10.38	83.00
Average	**10.84**	**86.69**

CIE - CO Wise Sum Formula

C01 = ASG(C01) + Q1(C01) + BestOfQ2&Q3(C01)
C02 = ASG(C02) + Q1(C02) + BestOfQ4&Q5(C02)
C03 = ASG(C03) + Q1(C03) + BestOfQ2&Q3(C03)
C04 = ASG(C04) + Q1(C04) + BestOfQ4&Q5(C04)

CIE - CO Wise Percentage

C01 % = (C01 SUM/total C01 Marks(12.5))*100
C02 % = (C02 SUM/total C02 Marks(12.5))*100
C03 % = (C03 SUM/total C03 Marks(12.5))*100
C04 % = (C04 SUM/total C04 Marks(12.5))*100

Average Marks	28.59
Student Count >Avg	11
Total Students	32
Percentage	34.375

SEE (End Exam) CO Wise Percentage

C01-C04	28.59	34.38

SEE - CO Wise Percentage

C01-C04 = End Exam Avg Marks

SEE - CO Wise Percentage

C01-C04 % = (End Exam Avg Marks/75)*100

CO ATTAINMENT	Internal Marks %	Internal Attainment	External Marks %	External Attainment	DIRECT ATTAINMENT LEVEL
CO1	86	3	34.38	0	0.75
CO2	88	3	34.38	0	0.75
CO3	90	3	34.38	0	0.75
CO4	83	3	34.38	0	0.75
Average					0.75

INTERNAL EXAM ATTAINMENT LEVEL SCALE

Attainment Levels	0	<=49
	1	50-64
	2	65-79
	3	>=80

EXTERNAL EXAM / FINAL ATTAINMENT LEVEL SCALE

Attainment Levels	0	<=39
	1	40-49
	2	50-59
	3	>=60

Direct Attainment %

C01=(CO1IntAtn*0.25+CO1ExtAtn*0.75)
C02=(CO2IntAtn*0.25+CO2ExtAtn*0.75)
C03=(CO3IntAtn*0.25+CO3ExtAtn*0.75)
C04=(CO4IntAtn*0.25+CO4ExtAtn*0.75)

CO-PO Matrix

Course	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2		Attainment
CO1	3															0.75
CO2	3	3	3		3				3	3		3				0.75
CO3	3	3	1		3				3	3		3				0.75
CO4			3		3				3	3		3	3	3		0.75
verage	3	3	2.3333	0	3	0	0	0	3	3	0	3	3	3		0.75

Final Attainment %

C01 = (DIRECT ATTAINMENT*0.8) + (INDIRECT ATTAINMENT*0.2)
C02 = (DIRECT ATTAINMENT*0.8) + (INDIRECT ATTAINMENT*0.2)
C03 = (DIRECT ATTAINMENT*0.8) + (INDIRECT ATTAINMENT*0.2)
C04 = (DIRECT ATTAINMENT*0.8) + (INDIRECT ATTAINMENT*0.2)

Course PO Attainments

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
ect tainment	0.75	0.75	0.58333	0	0.75	0	0	0	0.75	0.75	0	0.75	0.75	0.75
irect tainment	0.75	0.75	0.6667	0.375	0.75	0.375	0.375	0.375	0.75	0.75	0.375	0.75	0.75	0.75
nal ttainment	0.75	0.75	0.6	0.075	0.75	0.075	0.075	0.075	0.75	0.75	0.075	0.75	0.75	0.75

PO ATTAINMENTS

DIRECT ATTAINMENT (PO1)= (Average of PO1*Average of CO Direct Attainment)/3
Similar for PO2 TO PO12 & PSO1 TO PSO3
INDIRECT ATTAINMENT (PO1) = (Average of PO1*Average of CO Direct Attainment)/2
Similar for PO2-PO12 & PSO1 TO PSO3
FINAL ATTAINMENT = (DIR ATNM-PO1)*0.8 + (INDIR ATNM-PO1)*0.2

24. Student List with Slow Learners and Advance learners

Advance Learners

1	15RT1A1235
2	16RT1A1205
3	16RT1A1229
5	17RT1A1205
6	17RT1A1206
7	17RT1A1209
8	17RT1A1210
9	17RT1A1211
10	17RT1A1218
11	17RT1A1224
12	17RT1A1228
13	17RT1A1229
14	17RT1A1230
15	17RT1A1231
16	17RT1A1233
17	17RT1A1235
18	17RT1A1236
19	17RT1A1237
20	17RT1A1238
21	17RT1A1239
22	17RT1A1240
23	17RT1A1241
24	17RT1A1242
25	17RT1A1244
26	17RT1A1245
27	17RT1A1246
28	17RT1A1250
29	17RT1A1252
30	17RT1A1254
31	17RT1A1255
32	17RT1A1257
33	17RT1A1258
34	17RT1A1259

Slow Learners

1	15RT1A1235
4	17RT1A1201