

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES

EXTREME PROGRAMMING AND
MANAGEMENT OF EXTREME
PROGRAMMING PROJECTS

by
Ali BAKAN

January, 2007
İZMİR

EXTREME PROGRAMMING AND MANAGEMENT OF EXTREME PROGRAMMING PROJECTS

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering, Computer Engineering Orientation Program**

**by
Ali BAKAN**

**January, 2007
İZMİR**

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “**EXTREME PROGRAMMING AND MANAGEMENT OF EXTREME PROGRAMMING PROJECTS**” completed by **ALİ BAKAN** under supervision of **ASST.PROF.DR. ŞEN ÇAKIR** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

.....

Asst.Prof.Dr. Şen ÇAKIR
Supervisor

.....

Asst.Prof.Dr. Gökhan DALKILIÇ
(Jury Member)

.....

Prof.Dr. Haldun KARACA
(Jury Member)

.....

Prof.Dr. Cahit HELVACI
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

Dedicated to my family...

I would like to express my deep gratitude to my supervisor Asst. Prof. Dr. Şen ÇAKIR for his guidance, patience, excellent suggestions and encouragements development of this thesis.

To Res. Asst. Kökten Ulaş BİRANT for suggestions.

I would like thanks to my mother Sevgi BAKAN, my father İbrahim BAKAN, my sister Şeyma BAKAN and my brother Aykut BAKAN for their great supports, encouragements, and helps throughout my this study and all my life.

To Selen GÖKKAYA for her help on formatting this thesis report.

Ali BAKAN

EXTREME PROGRAMMING AND MANAGEMENT OF EXTREME PROGRAMMING PROJECTS

ABSTRACT

Extreme Programming (XP) is actually a deliberate and disciplined approach to software development. About nine years old, it has already been proven at many companies of all different sizes and industries worldwide. XP improves a software project in four essential ways; communication, simplicity, feedback, and courage. XP programmers communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested. XP programmers are able to courageously respond to changing requirements and technology. The aim of this study is investigating the Extreme Programming and determine whether it is important or not for developers and customers. Management of extreme programming projects are searched thoroughly. To do this, a survey was applied to companies of all different sizes and industries. In this thesis, Information will be given about survey and details.

Keywords : Communication, courage, extreme programming, feedback, simplicity.

EKSTREM PROGRAMLAMA VE EKSTREM PROGRAMLAMA PROJELERİNİN YÖNETİMİ

ÖZ

Ekstrem Programlama(XP) yazılım geliştirme konusunda karar vermede acele etmeyen disiplinli bir yaklaşımdır. Dünyada küçük,orta ve büyük ölçekli bir çok şirkette ve çeşitli endüstri alanlarında dokuz yıldır uygulanmaktadır. Ektrem Programlama yazılım projesini dört esas yol ile geliştiriyor; İletişim, Basitlik, Geri Bildirim ve Cesaret. XP programcılarını müşterileri ve arkadaşları ile devamlı iletişim halindedirler. Tasarımlarını basit tutarlar. Başlanılan ilk gün yazılımlarını test ederek geri bildirim alırlar. En kısa zamanda sistemi müşteriye teslim ederler ve öneriler doğrultusunda değişiklikleri uygularlar.Bu temel düşünce ile XP programcılarını cesurca değişen ihtiyaçlara ve teknolojiye cevap verebilirler. Bu çalışmanın amacı ekstrem programlamayı inceleyerek programcılar ve müşteriler açısından önemi ortaya çıkarmaktır. Ekstrem programlama projelerinin yönetimi etrafıca araştırıldı. Bunun için de farklı büyüklükteki şirketlere anket uygulandı. Bu tezde anket sonuçları ve detayları hakkında bilgiler verilecektir.

Anahtar sözcükler : İletişim, Cesaret, Ekstrem Programlama, Geri Bildirim, Basitlik

CONTENTS

	Page
M.Sc THESIS EXAMINATION RESULT FORM	ii
ACKNOWLEDGMENTS	iii
ABSTRACT	iv
ÖZ.....	v
CHAPTER ONE - INTRODUCTION	1
1.1 Introduction.....	1
1.2 Software.....	1
1.3 Software Methodology	2
1.3.1 Heavyweight Methodology.....	3
1.3.2 Lightweight Methodology	3
CHAPTER TWO-EXTREME PROGRAMMING	4
2.1 Kent Beck & Ward Cunningham	4
2.2 What is XP?	4
2.2.1 Planning	8
2.2.2 Designing	9
2.2.3 Coding.....	9
2.2.4 Testing	9
2.3 Four Values.....	10
2.3.1 Simplicity	10
2.3.2 Communication	11
2.3.3 Feedback	13
2.3.4 Courage.....	14
2.4 The Twelve Practices	16
2.4.1 Simple Design	17
2.4.2 Testing	18
2.4.3 Refactoring.....	19
2.4.4 Coding Standards.....	22

2.4.5	Collective Ownership	23
2.4.6	Continuous Integration	25
2.4.7	Metaphor	26
2.4.8	40-hour week.....	27
2.4.9	Pair Programming.....	28
2.4.10	On-site Customer.....	29
2.4.11	Small Releases.....	30
2.4.12	Planning Game	30

CHAPTER - THREE SURVEY.....32

3.1	Where Is Your Centre Office Of Your Company?	32
3.2	Do Your Offices Communicate And Work Each Other In Your Projects?..	33
3.3	What Is Your Position In Your Company?.....	34
3.4	What Type Software Is Developed In Your Company?.....	37
3.5	How Many People Develop Software In Your Company?	38
3.6	What Do You Think About Team Members ?.....	39
3.7	What Is The Frequency Of Customers' Meeting?	40
3.8	Do You Have A Test Unit In Your Company?.....	42
3.9	Have You Been Working At Home After Company About Your Job?	44
3.10	Do You Work Overtime In Your Company To Deliver Software At The Right Time? What Do You Think About Reasons?.....	45
3.11	Do You Use Unified Modeling Language (UML) On Projects?	47
3.12	What Do You Think About Any Developer Talk Over Customers About System?.....	49
3.13	How Often Do You Make Meeting In Your Company To Discuss Software?	50
3.14	What Do You Do To Determine Needs Of Customers?	51
3.15	When The Customers See The Software System?	52
3.16	If The Experienced Developer Leaved From Your Company, Is This Affected You Negative?	53
3.17	How Long Time Is A New Developer Adaptation Duration In Your Company?	54

3.18	Some Managers Set Some Limits To Developers To Affective? For Example Tea, Coffee Limit. What Do You Think About These Limitations?.....	55
3.19	What Do You Think About Week Job Time?	56
3.20	What Do You Do To Increase Communication Between Developers In Company?	57
3.21	Do You Have Any Coding Standarts?.....	57
3.22	What Do You Do For Junior Developers To Interfere Code?.....	59
3.23	Does He Or She Make Some Changes On Your Code That Is Written By You Easily?.....	61
3.24	What Do You Think About Code Ownership?.....	62
3.25	Do You Think, Pair Programming Is Useful For Quality Software?.....	63
3.26	What Do You Think About YAGNI (You Aren't Going To Need It)?	69
3.27	Do You Think Refactor Is Applied On Working Complex Code?.....	70
3.28	How Often Do You Refactor Your Code?	87
3.29	Do You Add Comments In Your Code?	87
3.30	Any Function That Is Done A Developer Can Be Written By Another Developer. Is This Situation Happening In Your Company?.....	88
3.31	What Is Three Factors To Success Software?.....	89
3.32	What Are The Reasons Delayed Software?.....	90
3.33	Do You Know Extreme Programming Practices?	91
3.34	Do You Apply XP On Your Company?.....	92
3.35	Do You Know That Apply XP Practices On Any Company?.....	93
CHAPTER FOUR - MANAGEMENT OF XP PROJECTS		94
4.1	Communication	94
4.2	The Management of Data Integrity and Accuracy	94
4.3	Management of Documentation.....	94
4.4	Management of Customer Needs	95
4.5	Management of Errors	95
4.6	Project Team Management	95
4.7	Pair Programming	96

4.8	YAGNI Management	97
4.9	Refactoring Management.....	97
4.10	Code Comments Management.....	98
4.11	Project Schedule Management.....	98
4.12	Test Management	98
4.13	Fourty Hours of Work Management	99
4.14	Modelling Management.....	100
4.15	Project Delivery in Time	100
4.16	Database Management.....	100
4.17	Programmer Dependency Management	100
4.18	Customer Satisfaction Management.....	101
4.19	Technology Management	101
4.20	Simple Coding Management.....	101
4.21	Coding Standards Management	101
4.22	Continuous Learning Management	102
4.23	Programmer Satisfaction	102
4.24	Planning Management	102
4.25	Adaptation Management.....	103
4.26	Resource Requirements Management	103
 CHAPTER FIVE - CONCLUSIONS.....		104
 REFERENCES		106
APPENDICES		108

CHAPTER ONE

INTRODUCTION

1.1 Introduction

Since computer was invented, people have been trying to solve their problems by using computer as it makes easy. In late 60's and early 70's, developing software was irregular and many programmers were used to it. Bugs or errors occurred in that period of time was really hard to solve.

In the end of 70's, some rules were set to develop software more easily and people like Edsger Dijkstra published articles about this subject. Since then, by the help of variety of rules, more effective programs have been developed and coding without rules has been left behind.

In 80's, many rules and practices were obvious. However, having a lot of rules made software development hard to put in practice. At this point, software methodologies were suggested.

Extreme programming is one of these methodologies. It includes very few rules and practices. It is a method where developers feel themselves more independent. Developers do not forget their being followed and controlled. Continuously evaluating the project velocity, it is aimed to be submitted on time.

1.2 Software

Software is a group of program and procedures that can operate integrated with the hardware and provide to perform specific operations.

In 1935 Alan Turning mentioned this subject in one of his articles and it is firstly implemented by John W.Turkey in 1957. (Hally, Mike, 2005)

There are three types of software; System software, programming software and application software.

System software; provides the programs to operate with the computer hardware. Operating system, drivers, utilities are this type of software. Through system software, the complexity occurring with the relation between the computer and the hardware is not indicated to the user.

Programming software; forms the medium where developers can develop software. Tools like text editor, compiler, interpreter, debug are this type of software. In IDE (Integrated development environment) the developer sees only one tool and all the operations are implemented through this tool. By this tool compile operation is performed and executable file is developed.

Application software; provides people to follow their work easily in computer environment. Nowadays in industrial, automation, education and medical areas computer software are used.

1.3 Software Methodology

It is a cluster of practices and rules needed in the development of computer software.

At the end of 1960s and in the beginning of 1970s software are not developed according to definite rules. Lots of developers are specialized in the field of making complex codes. The occurrence of an error or a bug causes nearly no recoverable results.

In 1968 an article about GOTO statement was published by Edsger Dijkstra. After this article software development gained a new face. It was thought to make more

effective codes according to a set of rules. Cowboy programming was shelved in this era.

1980s is the beginning of good things for the developers. In this period a set of rules and practices were being used by the developers. Quality software were being developed and the problems were rapidly being solved.

But in the 21st century same rules started to occur a lot. To solve this situation having difficulty in applicability, case tools are started to be used. Through these tools it is thought that rules whose implementation is difficult can easily be implemented. In the following years instead of heavyweight methodology including lots of applications and rules, lightweight methodology including few applications and rules is preferred by the developers. (Brewer, J., 2001)

In fact in this period the developers did not want to turn back to cowboy programming totally. They want a structure where software sheriffs and cowboy developers can work together. It is thought that this will be more efficient.

1.3.1 Heavyweight Methodology

Heavyweight methodology includes lots of rules, applications and documents. In this methodology it is required to fit regularly to the rules and follow them.

1.3.2 Lightweight Methodology

Compared to heavyweight methodology, it includes very few rules, practices and documents. It's tracking and pursuance is easy.

Extreme programming is one of the lightweight methodologies. It includes very few rules and practices. It is a method where developers feel themselves more independent. Developers do not forget their being followed and controlled. Continuously evaluating the project velocity, it is aimed to be submitted on time.

CHAPTER TWO

EXTREME PROGRAMMING

2.1 Kent Beck & Ward Cunningham

In the beginning of 1990s a research on how good software should be is made by Kent Beck. Working with Ward it is aimed to access set of realities. By using their experiences in the company where they work as developers, rules that should be noticed to develop good software is determined. In 1996, in May a new project was started in Daimler Chrysler with Ward and in this project they used the methods determined before. In this methodology which they were going to call “Extreme Program” later, they believe that everything should be simple. They defend that its being difficult is not good for customers and developers. (Extreme Programming. (n.d.))

Kent Beck and Ward emphasize on four values. They say, “You should develop communication”, “You should search simplicity”, “You should approach with courage”, and “You should take permanent feedback”. “Communication, Simplicity, Feedback and Courage” are the four important values of XP. XP is built over these four values.

2.2 What is XP?

XP is a model suggested by Kent Beck. Kent Beck and Ward Cunningham are the two developers developing the object oriented software. Extreme programming is a lightweight approach in software development. Quick feedback and high communication power stand in the forefront with the permanent contact to the customers and constant tests. (Brewer, J., 2001)

In traditional approach:

Everything is documented in the stage when needs are determined and analysis is made. Customer needs, user interfaces and how design will be are stated in this document.

In traditional approach, analyst sometimes asks customers complex questions on behalf of understanding their needs. There are questions predominantly about operating system and hardware in them. Customer of course can not understand some of these questions. When the indistinct points are not asked, needs can not be determined by the analyst. Thus, software is developed according to their ideas. When the project is submitted, codes not wanted by the customers occur.

Still in this approach, sometimes data validations, correction data are not asked or start operation without asking if the relevant area is numeric or character.

One of the biggest handicaps in traditional approach is the documents' being fairly long. In any situation when both customer and developer start to read the relevant document, they read the first twenty pages and get bored of reading other pages. As this document usually includes technical expressions, it does not appeal to them.

Actually if a customer is able to understand the structure, this will facilitate the developers' work. Yet this can only be provided with the fact that a customer is an engineer or designs are simply made.

Kent Beck claims in the software development model designed to get rid of the all problems and slowness above that if no importance is given to documentation, development is quickly made, instead of detailed matters a document stating user's needs and demands is prepared, everything is going to be quick and needs are going to be supplied. In this situation developers do not need to read the minds of customers. In the document they will get, as there will be their needs and demands, it is thought that programs can be integrated easily. As the questions are directly to their needs and demands, customer will feel himself good.

Customer groups and developer groups start analyst work together. After completing this study in 2-3 weeks, priorities and things going to be made are determined by getting through this document for 1- 2 hours. After 1 or 2 months document is prepared, program is analyzed, tested and it can usually be encountered with an unexpected situation or undone work is determined even if it is in the document. And this causes poor quality software to occur.

In XP approach operation happens in this way.

It starts with meeting with the customer. Customer in fact sits at developers' elbows.

One project is performed in 3 phases:

Release planning phase: It is the first phase. In this phase, customer's needs are determined in user's stories. Discussing with the customer, needs are arranged by developers.

Iteration phase: It is the second phase in software development. In this phase tests are written and questions are answered by the customer, in the meantime program is being developed by developers.

Release phase: It is the last phase in software development. The developed program is downloaded to be tested by the customer. Having been analyzed, customer accepts or rejects the program. In XP the orientation of the developer is made by the customer. (Wake,W.C., 2001)

Iteration phase starts in every two weeks. It is due to the importance of tests in XP and the customer is aware of this fact. When customer sees that the project is improving regularly, he takes courage and his confidence to the program increases.

XP is successful as it emphasizes on the customer satisfaction. It adapts to answerback to the customer's demands whenever they need. In every stage of the project, as developers see that they can answerback to customer's demands, they take courage.

Another side of XP is the importance they give to team work. The manager, customer and all the developers are the parts of the team and the collective feature of the team is to develop quality software. The corner stone of it is due to the system's simplicity.

In XP, the roles are shared between the customer and the developer. They are in the same team but they take different decisions. Customer states what he wants and the developer states how much it will cost.

When the areas decided by the customer are checked,

Scope: What should a system do?

Priority: What is the most important?

Composition of Releases: Which operations should be done is decided.

Dates of Releases: The date of the works to be done is determined. (Wake,W.C., 2001)

When the decided points of the developer is checked,

Duration to add a new feature.

The control of how much of the technical features can be implemented.

How the teams work?

Business plan.

XP develops a software with four values; communication, simplicity, courage, feedback.

XP developers are always in touch with other developers and customers. They keep their designs simple and regular. Making test, they always get feedback. They submit the project to the customer as soon as possible. XP developers are fairly courageous as they keep up with customer and technology.

XP is resembled to an onion. The inner layer is the programming layer. It operates by using a special programming technique. The middle layer includes the team practices. The outer layer provides the communication with the developers and customer.

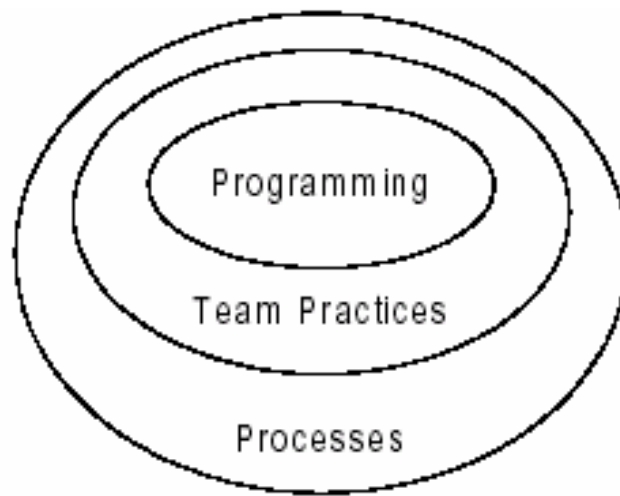


Figure 2.1 XP as an onion

XP is different from other systems. There are small releases and when these parts are combined, the main picture is accessed. This is an important difference from the traditional approach.

When the stages are quickly checked,

2.2.1 Planning

- User stories are written.
- Time planning is made in release planning.

- Small releases are often removed.
- Project velocity is constantly measured.
- Project is divided into small pieces.
- Iteration planning starts with iteration.
- Operation constantly happens around people.
- If there is a problem, it is fixed without fail.

2.2.2 *Designing*

- Everything is simple.
- A metaphor describing the system is chosen.
- CRC cards are used for design.
- To reduce the risk in design, fast solutions are found.
- No function is added without the right time.
- Refactoring can be made when it is needed or at any time.

2.2.3 *Coding*

- Customer is always available.
- Coding is made in coding standards.
- Unit tests are constantly controlled during coding.
- All projects are encoded by pair developers.
- At the same time only one person can make changes on the code.
- Codes are often integrated.
- In coding ownership belongs to everyone.(collective code ownership)
- Optimization is left to final.
- In coding it is not worked overtime.

2.2.4 *Testing*

- All codes have unit test.
- Before a code is released, it should certainly support all unit tests.
- When encountered with a bug, new tests are prepared.

- Acceptance tests are often prepared by the customer and they are followed. Score is given to program parts and they are published.

2.3 Four Values

2.3.1 Simplicity

Simplicity is the heart of XP. Simplicity affects everything made. XP's success in implementation depends on this. Simple design prevents customer's undesirable long designs and decreases the risk. It is again possible to add the absolute required features easily with the simple design. The most essential advantage of simplicity is to prevent the exceeding time in solving the today's problems.

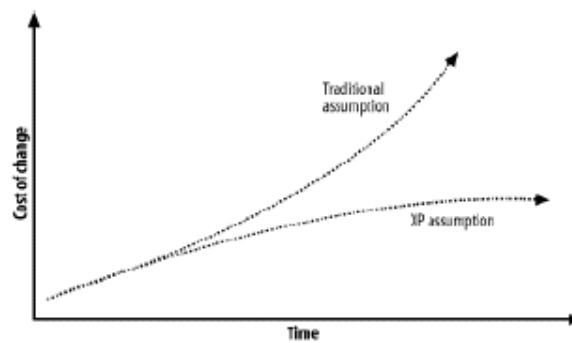


Figure 2.2 Cost of change on an XP Project.

The graphic shows in traditional approach that when the time increases, the cost of change is also increases. As when not a simple structure is used in coding, interference to the codes gets hard as the time passes. Whereas in XP as it is seen in the graphic the changes in the codes according to time first takes time but later time taking operation is fixed. As seen in XP changes that take up time at first continue increase as well. This situation is inevitable; however XP can perform this by keeping the code simple and refactoring. One of the methods that fixes the time is the unit tests. Through these automatic tests it is possible to make changes quickly. If the tests were not automatic, after every change test operation would have to be done and this would take lots of time.

One of the factors that reduce this time is the collective code ownership and pair programming. By this way all the team is in the project and has information about the whole project.

A simple design is always completed more quickly than a complex structure. XP developers keep their designs considerably simple. If there is structure thought to be complex, that part is immediately simplified and then it is continued.

When a new feature is added, it is solved with the simplest solution. It shouldn't be tried to prove oneself by using a super structure. Operation should be performed by using "if" expressions when needed. It should be avoided from pompous coding. If code ensures all the unit tests, it is passed to other tests.

2.3.2 *Communication*

In the first analysis meeting and the stage to determine the demands, everything is demanded by the customer. Here the most important is to decide which of these are the most essential. In this situation without hurting the customer, it can be said "We are taking notes so they won't be forgotten. But we don't take this to our design. We will add when it is needed. This approach can prevent works to turn into a very complex structure. Against this kind of situations developer should be in the mood to say "no".

A good communication is essential for team work. When it is thought that XP team is composed of developer and customers, this importance increases more. Each individual of the team has different capability. Their educations, personalities, knowledge levels and their point of views to event are all different. Thus it is really very difficult to communicate with these different featured individuals. It is important to be agreed on common ideas. Some may be very young and may not be experienced. From time to time customer may be very experienced and developer may not be that much experienced. In both situations people should handle the event with tolerance and sympathy. By this way the communication will be much easier.

The most important difference between a computer developer and a person outside is the social status. In general the developers are launched as people who do not communicate with people a lot and who always sit and encode programs. Customers are more active. XP aims to balance this situation. Because of this they want developers to be involved in the work and be in touch with customers. (Cunningham, W,2006)

A good boxer or a person who makes judo takes his position unconsciously and by instinct. In communication with the customer, it is tried to provide this situation on developers as well.

Developers can not only understand their customer's demand by looking at their check list. They can reveal customer's demands from their body language. Some people even do not like to talk much. To perform an efficient project, they should make them talk and approve the meanings of their facial expressions. If the developer already has knowledge about the relevant subject, he should act by affirming his knowledge to the customer.

Heavy volume documentation causes harm instead of providing advantage. The best communication is face to face communication. Otherwise nobody will read those documents.

When we say communication, it is not only communication with customer. Developers also communicate with each other.

Another communication is code communication. Code communication means the ability of different developers studying on codes to understand and interfere without any help. This can only happen with simplicity. If there is a complex structure, it should be simplified or thrown light on indistinct points with the remark lines written in codes. The remark written in code is much more useful than the code hold outside. In version change or code change as the remark line is in the code, by moving

together with the code the occurrence of a missing remark is prevented (Burke,E.M., & Coyner,B.M., 2003)

The other communication way is unit tests. By this way it is decided how regularly encoding works, is provided to be shown to the customer. The information of how the written classes and methods can be used is hidden inside these units. (Burke,E.M., & Coyner,B.M., 2003)

Pair programming is the other way of communication. As programming is pair, the other mind can be read by a developer. In pair programming, when you understand the thing your partner doesn't understand, to educate him and transfer your experiences is also important for communication. With this method, developers perform to think and decide together. At least two of the developers are in communication. (Burke,E.M., & Coyner,B.M., 2003)

Customer and the developer are always in communication as well. As they work together, communication is inevitable.

2.3.3 *Feedback*

People generally do not like feedback. The documents followed in synchronized way and big test operations are the most used methods for feedback and this does not appeal to most people.

In fact, with the result of feedback's short and long term advantages they are going to like it.

If you ask to engineers, you will get answers that feedback should be done. Without feedback, work can not be submitted. At the end a product that does not satisfy the demands of the customer occurs. Even one person has difficulty in seeing his own error. Different point of views will bring different solutions along with.

XP developers are resembled to driving a car in the dark. With feedback, customer shows the shortest distance to be reached. But if the feedback is not done or stated at the right time, it won't provide advantage.

In XP feedback should be received after the developer submits every code. Otherwise it will be late. Feedback should be received from the real users. If a bug occurs, a new feedback should be received in 24 hours after it is fixed. For developer a feedback is positive only if the user says "magnificent". The other statements will not be positive.

Customer's always being near the developer has an important effect in getting feedback. After every change customer tells his ideas to the developer and the developer is able to fix if there is a wrong. Provided that delivery was after few months and the developer received the feedback at this moment, he could maybe change the decisions of few months ago and it wouldn't receive the positive feedback.

Through short term feedback customer can follow every new development, states the points that developers should focus if it is needed, and provides these subjects to be done priority. They can answer the questions such as: which subject is important or not , which one should be done first or later.

For a developer one of the biggest feedback resources is the codes. Codes actually show developers whether the works are going well or not. Unit tests give the information about the quality of the code. Developer operates the unit test after every change. Provided that it didn't pass any test, firstly the problem is fixed and after all the tests are applied, feedback is received.

2.3.4 *Courage*

The biggest trouble in coding is the lack of knowledge and to enter crisis. If there is only one person taking part actively in the project and if some things are gotten very late, project velocity must be check. In this kind of situation, the best thing to do

is to inform the other developers'. In fact cross training is the best solution to prevent the loss of information. In XP through pair programming, cross training is actually performed. By this way instead of one developer knowing the part of a code, it is provided another developer or the whole team to know the program. Even if a developer knowing the program quitted the job, there would be no change in the project velocity.

One team being informed about the program provides advantage. In case of critical operations, by giving the work to any one in the team, the problem can be solved. By this way a balance between developers is established.

The biggest courage example is pair programming. Through pair programming, both continuity and efficiency are obtained. Even one of the developers gets bored, another developer replaces and operation continues. The change is tried to be provided in the boredom of developers like the teachers' saying in the university exam, "If you're bored of studying Turkish, pass Mathematics".

XP developers are always in touch with other developers and the customer. It is in the situation of making changes every moment. It forms the parts within the shortest possible time. This actually makes them gain courage in keeping up with the technology and answering the customer's demands.

The Professional developer's behavior like saying "Do not worry, we can handle this" in crisis situation is in fact a good example of courage. They started to trust themselves in solving the problem and took courageous steps.

Actually in XP the managers are also behaving courage. To accept the pair programming without seeing it as waste of time and try to apply it, is a courage indicator. In every work to develop code by considering simplicity is a courageous approach.

2.4.1 Simple Design

In the first coding, it will be a fault to expect simple coding from developers. For instance, the coded method names can be long. The coded methods can not be appropriate for the tests or they could code a method that they had coded before. A complex code with these features, is hard to understand, long term to change and difficult to control in the event of system's development. Whereas through simple code, in coding both documentation is provided and coding is more regular.

For semantic denomination, the names of the method should be short and brevity and the responsibilities should be well plotted. (Burke,E.M., & Coyner,B.M., 2003)

A basic code:

- Operates all the tests.
- Can explain everything needed to be explained.
- Everything is done only for one time.
- Unnecessary parts are not included.

There is maximum consistency and minimum relation between codes.

Code's operation is important to pass all the tests with success. After this comes the comprehensible of the code. It is wanted to both of them is provided. By this way the maintenance of the code will be easy.

In XP the simple design is maybe the biggest constraint. When there is much inheritance, there can be lots of classes and methods. In this situation XP wants these classes and methods to be combined after controlling them. XP is a follower of refactoring.

About a thing done for only one time, Wayne Carson thinks like this, "According to me, operation done for only one time provides codes to be combined tightly. In

fact this situation forms trouble for the developers.” (Burke,E.M., & Coyner,B.M., 2003)

If it is looked at the statements above, he seems rational. But Kent Beck does not agree with Wayne. He says, “Wayne’s statements are opposite statements for the situations I faced up to now. According to me this operation does not provide codes to combine tightly to each other .In XP everything is small pieces and have function independently. If a change is a functional change that will be made in one of these small pieces, anyway as it is a change, it is inevitable.

It shouldn’t be confused in simple design. Simple design does not solve the problems or does not make predictions about the future. Simple design provides customer’s acceptance tests and program to pass successfully.

Simple design helps developers with remarks. By this way a system that can expand is formed. Because customer can not demand new needs without seeing the system. You could have developed according to the current system but after the customer used it and saw the simplicity, he would say, “It will be good if it is followed this in the system”. This can only be done easily with simple design.

2.4.2 Testing

It is one of the most important practices in XP. Through this, it is made sure that when a new feature is added, system works or after corrections are made or new bugs are solved it can be decided whether it is still working as before. Yet here the most important is this. In Waterfall model tests are made. However XP’s most important and best feature is that tests are made automatically. If the tests were not automatic, tests would not always be made in the result of continuous refactoring. Despite of these developers can make refactoring with composure in whichever point they want.

Tests are fundamentally divided into two. First one is the acceptance tests that customer introduces and checks whether the program supplies his own demands. The second one is the unit tests that developers encoded and applied while developing the

program to control whether the parts of the program are operating regularly. (Hightower,R., & Lesiecki, N , 2002)

2.4.3 Refactoring

Through refactor simplicity that one of the basic practices of XP can be provided. In fact all the practices are related to each other. For a simple design refactor should be done. After refactor operation, test should be done to control whether the program is operating regularly. Test being automatic provides refactoring operation to be done without fear.

Refactoring does not pose an obstacle to the adding of a new feature. Adding of a new feature and refactor operation are two different operations. Although at the end of two, test is needed to be done, refactor operation does not change the function of the system, it only causes a more tangible and simple structure to appear. Actually the point triggering refactor is the thought of the complexity of the structure after change. . (Hightower,R., & Lesiecki, N , 2002)

In some software books there are statements that believe adding of a new feature will ruin the feature existing before. That's why it is believed that the outline of the project should be drawn well. There is a probability of feature's corruption. However, XP still is not favor of adding feature in advance for simplicity. It can be added whenever it is needed. By this way, the adding of features that will not be needed later on is prevented.

The refactoring operation is decided with the sound coming behind the brain while coding a program. "Do not enter the shortcut way, it should be simpler, It is not very tangible". When sounds like this are heard, it means time to apply refactor has come.

Below there is an example of refactor operation.

Table 2.1 Class person in Java

```

Public class Person {
    private String firstName;
    public void setFirst(String n) {
        this.firstName = n;
    }
}

```

In the table 2.1 there is a class. Here the first point that will attract attention is the change of parameter stated as n to be more tangible. For example n parameter can be changed as firstname.

Table 2.2 Class person with better argument name.

```

public class Person {
    private String firstName;
    public void setFirst(String firstName) {
        this.firstName = firstName;
    }
}

```

Another method is to change the method name as SetFirst. For this it can be named as setFirstName.

Table 2.3 Class person with better argument name and better method name.

```

public class Person {
    private String firstName;
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}

```

The last mode became as above. Here what is done is to change the argument name and procedure name as being more tangible. No change is done in the function of program. In fact the most important subject in refactor is this. No change should be done in function. The program parts using method and class and change made by controlling should be reflected in there. (Burke,E.M., & Coyner,B.M.,2003)

2.4.3.1 Goals

In refactor the purpose is, as mentioned before, to provide simplicity without ruining the previous features and the operation logic. Refactor is operation that takes time in projects.

That's why, the mentality that there is refactor in any case, after it is coded we can simplify it is not right. To deliver in lead time of project, simplicity should be thought at first sight. It should be tried to be done fairly simple and after refactor should be done once more. The situations overlooked can be simplified in this phase.

The other purpose is to combine the repeated code parts. The repeated statements seemed as innocent points, are actually the points that will cause more error. (Coyner, B.M., 2003)

2.4.3.2 When to Refactor

There is not a specified time for refactor operation. Refactor should always be done throughout the project. As mentioned before, if inside a sound is telling, refactor should also be done in that situation. In a company a manager or a user never comes and asks whether you did refactor. They want to control whether the program is operating regularly. That part interests them. For this moment refactor does not interest them directly but later refactor interests them indirectly as they do not want a needed change take too much time.

In fact unit tests are the factors that triggers refactor. API (Application Program Interface) coded for test also instructs. Called method name can attract attention, but later it can be changed in parameter numbers or denomination. (Burke,E.M., & Coyner,B.M.,2003)

2.4.3.3 How to Refactor

Refactor operation should be an operation supported with automatic tests. Before code is refactored, all the unit tests should surely be completed. No uncontrolled test should be left.

By this way, the tests done in the beginning will be done once more at the end of refactor and these will be waited to finish. Of course, on the code part made refactor no one should make a change at that moment. If it is specified them in sections;

- It is made sure whether all the codes ensure all the unit tests.
- Refactor operation is made.
- It is controlled whether the operation made creates changes in the system's operation logic.
- The second and third sections are repeated till refactor is completed.

(Burke,E.M., & Coyner,B.M.,2003)

2.4.4 Coding Standards

In XP coding is encoded in a definite standard because of pair programming and collective code ownership. Actually the existence of code standard is not an easy operation for developers. Because they have to abandon the methods that are determined by his own experiences or applied in his previous company. If all the developers are agreed on the same coding standard, anyhow they do not need to do an extra work for coding standard. If you are not taking a part in the team, you can encode in a way as you wish. The names of method and class can be called in a way as wanted. You can even form a coding standard peculiar to yourself. Here the point that should not be forgotten is being a part of the team and the possibility of other interferences to the program. If you are taking part in the team and if a new member of the team is wanted to understand the program quickly, encoding should be done in definite standards. In short term these standards may not have advantages, however when it is seen in long term, it will shorten the adaptation process of new developers

and the developers can easily understand the program. Whether the program is encoded in coding standards can be evaluated by someone's understanding the program easily.

If it is needed to give examples, about object denomination these decisions can be taken:

- While database table denomination is being made, Turkish names should be given and no base line should be put.
- Views are to be terminated with `_v`
- Functions are to be finished with `_f`
- Procedures are to be finished with `_p`
- Packages are to be completed with `_pkg`.

Actually in the first plan to take decisions like this and apply them is very difficult. At least there is one standard and they can be easily explained to a new developer? He can decide at first sight whether a method in the code is a procedure or a function.

When it is said coding standard, only denomination shouldn't come into mind. While coding in C, how to code `{ }` symbols defining begin and end should also be decided.

2.4.5 Collective Ownership

When it is said code ownership, it can be encountered with different methods.

One Owner per Class or Package: Every class and package has an owner. Relevant operations are done in control of it.

Sequential Owner: There is ownership in sequence. That is to say according to version the ownership of class and methods changes.

Ownership by layer: There is ownership according to the levels of employees. (According to their quality, their working positions)

Collective Code Ownership: Through collective code ownership, everyone is acquired to have knowledge about any part of the program. Anyone can make changes, apply refactor, determine the bugs and fix them in any part of the program. This situation provides developer to keep dependence in the lowest level.

This can not be understandable at first sight. The impossibility of this application can even be defended. However it should have the lowest dependence on the function of extraordinary projects. Sometimes an urgent change may occur and at that time the coder of that program may be on holiday or may have quitted the job or may be ill.

When it is considered all these situations, the necessity of everyone's being informed about everything is a true idea.

Time to time developers can use their dependence against their managers. A developer brings his own work in the foreground and demands high wage or changes his job by using his acquirements in a different company. Actually XP aims to remove the big gap between developers. XP wants a team of developers who have knowledge and trust themselves.

Pair programming actually supports collective code ownership to some extent. In fact instead of one person being informed about the part of the coded program, two people are not something bad. To spread all the works to the team, after a refactor operation is made on the part of the program, it should be given to another pair of developers. By this way that couple will also be informed about the relevant part of the program.

Through this it can both be controlled whether it is coded according to code standards and can be tested whether the code is understood adequately.

Collective code ownership in fact also prevents to make more refactor afterwards. Because the coder of the program thinks that checked by someone else and tries to code the program carefully and simply.

At the same time the fact that tests are done by others also supports the application of collective code ownership.

Through collective code ownership profit is made out of time. Any change can be made very quickly by any one from the team and this provides both developer's trust in himself and customer satisfaction.

Generally in companies works are done in tatters. But one developer is responsible from every department. One codes and tests the part that he is responsible for. Then it is delivered to the customer.

If there is a problem in coding, changes are made, yet this add can ruin another part. That's why after every adding, test should be done.

2.4.6 Continuous Integration

Continuous integration is to combine every part in application nearly everyday. Of course for this, a collective code repository is needed. All the works done and tested should be kept in this repository. If one completes the work, he should add this to collective code repository and make the tests. At that moment another person should not reach this repository. Only one person can make changes at the same time. To apply all these operations, tools following the previous versions of codes are needed.

Continuous integration is in fact a critic process. If all the small releases are working, the culmination of all releases shouldn't be waited to see all the system. For this, test should be done by integrating the releases one time in a day and user should be informed about the application.

Through continuous integration, which add ruins the program in which stage can be determined and structure is set up on sound basis. XP predicts integration operation to be done often in the shortest time.

Here one of the important points is that integration operation does not take long time. Anyhow the changes should be put into service in the same day. Another developer can again code a code release that he couldn't see in the collective code repository. In fact this part is coded but not yet integrated. Two people doing the same work should be prevented.

In XP every pair of developers is responsible from their own codes and adding the codes to the repository. The added code to the repository is now belongs to everyone. All the programs should pass %100 the entire unit test before they are added to repository. So developer should prepare very well and encode according this.

Through continuous integration, problem of consistency is also removed. How a code release encoded by one and the other part coded by another pair communicate can be seen at that moment. According to this precautions can be taken.

2.4.7 *Metaphor*

It provides the project to be referred from a functional side. In fact, the object forming the project includes objects like method. It is the projects' window opening to outside and the summary. By this way both the customer and the developer can talk about the project comfortably. Metaphor can respond to both the developer and the user in telling the structure.

For instance metaphor can be a paper that provides to keep object names and methods. To show the relations between them is important for fulfilling its duty. Here of course, the coded denominations should be understandable. As mentioned in refactoring, denomination from parameter names to method's name should be beware.

In projects while choosing a metaphor, it is important for the tool to be design as easily to be learned or not to have very much knowledge. As it is no need to spend too much time to learn the tool.

For instance, in project developed as object oriented the metaphor should include these:

- Class
- Method
- Variables
- Basic functions

When it is examined in short term, with a kind of study like this everyone understands what is done in which level. In long term, it integrates design and coding, it helps the new to comprehend quickly what is done, to see the works done till that moment and to prevent the program to be encoded once more. Always provides the design to be simple.

By this way all the system can be displayed. In a system without a metaphor is really very difficult to reach the objects. Because of this coding should be analyzed.

2.4.8 40-hour week

In XP, a work for 40 hour a week is thought to be enough. If it is working over 40 hours, it means there is a mishap. Too much working causes damages than providing benefit. Developers make more mistakes when they work too much. It is ideal to work 5 days a week, each day 8 hours. (Hightower,R., & Lesiecki, N,2002)

Software is a stressful work. To provide customer satisfaction requires effort. The best way to release this stress is to overcome it in weekends.

2.4.9 Pair Programming

XP team is composed of pair of developers. In pair programming, there is a keyboard, a mouse and a computer. If one developer has difficulty in seeing the monitor, second monitor can be used. Here of course the place where developers sit is important. The room and the desk should be suitable for working.

Pair programming operates like this:

- For the next work user stories(they are small releases stating user's demands) are taken and analyzed.
- Later it is wanted an other developer to help about the subject.
- Afterwards, it is analyzed and this small release is completed. After it is tested, integration operation is done. Later another release is found and with another partner this operation is also done. This operation continues till the project finishes.

Workings with small releases and different partners have some advantages.

- All the team members being in relation with eachother is supplied.
- Information and experiences are transfered from different individuals to different individuals.
- Generally experienced developers make encoding more regular. With the help this transform other new graduated developers have chance to encode regular programs.

Pair programming in XP requires a very high level of disciplined operation. Developers should code and apply all the unit tests after every new add. Pair of developers should alert each other and when they do not fit the XP practices, the other should take the keyboard. When one of them gets bored, the other should continue to code the program. By this way their being thrown out of gear will be prevented.

Pair programming is in fact one of XP's revolutionary practices. It is a very big courage example to put two developers in front of a computer. Some of the managers object to pair programming by saying "how can I put two developers in front of one computer when there is less time for the delivery of the project ". And XP developers ask these questions to people who are thinking like that, "If all the system is wasted, will the cost of it be low? In a subject dominated by one person when the relevant person quits, what will happen? Are you considering these? With these questions, they try to attract attention to developer dependence." (Burke,E.M., & Coyner,B.M. ,2003)

The most important advantage of pair programming is its redoubling the team communication. By this way, later you have to communicate with someone from the team that you don't have any dialogue before. Beside a better communication, cross training can also be established.

If a developer have high level knowledge of database like Oracle, Sql or knowledge of accounting, exchequer, stock modules, this developer's education to another developer will be better with pair programming.

Pair programming is important for a quality coding. As two people analyzes the program, without encountering lots of bugs they can develop software whose tests give regular results. While a developer is encoding for the relevant part, the other thinks of the forthcoming situations. In brief, while one is looking at a small picture, the other is looking at a big picture.

One rotten apple can be enough to spoil a basket of apples. That is to say, a developer coding a program carelessly, can slain the efforts of all developers. But in pair programming, developers coding carelessly make an effort to code carefully.

2.4.10 On-site Customer

In XP system customer is always together with the developer. It plays an active role in tests being coded, questions being answered and the determination of

priorities. The most important thing is that this situation enables to give answers to the developer's instant questions. If a team finds answers to questions in a few hours instead of few days, the other operations will be more rapid. (Hightower,R., & Lesiecki, N ,2002)

2.4.11 Small Releases

Continuous integration operation can only be successful with pair programming and small releases. By integrating every release after being tested, all the team can be provided to work in a synchronized way. If complex releases are tried to be integrated, in this no success can be gained and the integrity of the team can not be obtained. If the releases are not set well, lots of changes done will cause other releases to change and this has negative effects. Because of this, while setting the releases; it should be set coordinated with other releases in a fairly minimum way.

The size of the small releases should be in a way that has values for the customer and spends minimum time. Through small releases feedback will be get from the customer and it will be understood that the works are continuing in accordance with customer's demands. At the same time the basis of simplicity will be obtained by the agency of this.

Release planning meeting is a meeting where small releases are set and small releases prepared before are delivered.

2.4.12 Planning Game

Planning game is a game whose actors are customer and developers. In this game customer determines the order of works. And also the outline of the shared works and the delivery period are determined. In some situations, for the demands of the customer developer provides the operation to be done in a different way by directing the customer.

In this game operation such as:

- The outline of the Project.
- Priorities.
- Delivery times.
- Technical informations.
- Results.
- Risks of the Project.
- The organization of the team is done.

This game lasts for one or three weeks. Customer determines the field and the developer determines the cost. Actually the outline of the forthcoming game is plotted in this game.

Release planning phase is established on the customer's stating the small releases, the developer's analysis of these releases, determining the approximate time and making the plan over this.

Iteration planning phase in fact resembles release planning. Customer chooses the release set on the previous plan. Later developer implements the relevant release. If attention is paid, in XP system everything depends on customer user stories. Scheduling of the project is completely established on this system. The stories written on the index cards are made according to the order determined by the customer. (Hightower,R., & Lesiecki, N ,2002)

CHAPTER

THREE SURVEY

With this study, it is aimed to research Extreme programming in detail and to state the yields of it for the customer and the developers. A survey is applied to different software companies. In this part information about the survey will be given. By asking questions to developer, manager, people at the level of analyst, answers are seeked.

3.1 Where Is Your Centre Office Of Your Company?

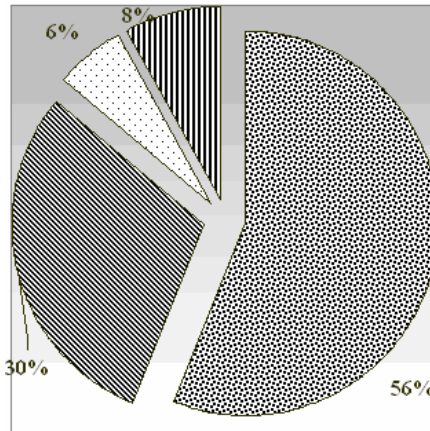


Figure 3.1 Locations of companies.

Table 3.1 Table contains locations of companies in survey

Where is your centre office of your company?	
%56	Istanbul
%30	Izmir
%8	Other cities
%6	Ankara

The basic reason to ask this question is to specify the location of the people attended the survey. Generally software work is in high level in the three big cities of Turkey, İstanbul, İzmir, Ankara. Developers prefer these cities after they graduate. Companies pay attention to the location of the company developing the software that

they are going to buy. Because they consider a company in İstanbul can respond to technology better and supply a better service.

3.2 Do Your Offices Communicate And Work Each Other In Your Projects?

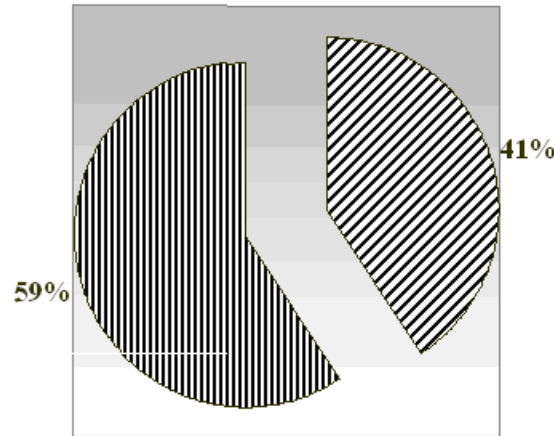


Figure 3.2 Communication of software companies.

Table 3.2 Table contains communications software companies.

Do your offices communicate and work each other in your projects?	
%59	No
%41	Yes

The basic reason to ask this question is learn whether the companies have more than one office and whether they are developing a project by getting across with these offices. According to the answer of this question, it is going to be decided whether these companies have difficulty in communication.

But %59 of the them develop project in the same office instead of different offices.

3.3 What Is Your Position In Your Company?

The aim of this question is to learn their positions.

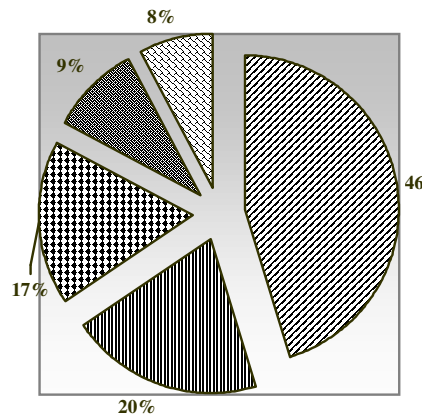


Figure 3.3 People position's in company

Table 3.3 People positions in company.

What is your position in your company?	
%46	Programmer
%20	Project manager
%17	System analyst
%9	Manager
%8	Student

A software company have employees who have different locations. They can be developer, managers, project managers, system analysts or part time students. All these people have different responsibilities.

Especially developers have big responsibilities in XP system. In this system XP first wants developers to write the tests and then make coding. Later, XP wants them to make arrangement in coding while making refactor. This operation can not be accepted by all the developers. Whereas this operation is difficult for developers who are used to traditional methods. Because it may not be easy to adapt a developer who is used to the method, first design, then code and after test to a this method.

Moreover, while in classical method this is taking weeks, XP deals with minutes not with weeks.

A developer comes in the morning and starts the day with a 10 minutes meeting. In this meeting the purpose is to determine the problem and the person that is going to resolve the problem. Any operation to resolve the problem is not made. Developers always start and finish the day on time. They work for 40 hours a week. At the end of the meeting, developer move to coding. Two person start coding. While one is encoding, the other thinks. The coder thinks in tactical meaning, whereas the other thinks in strategical meaning. This duty changes after a definite time.

This pair of developers write the tests as possible. In the points they are not able to write, developers try to complete the operation by getting help from the customer. During coding, beside writing test , trying to code as simple as possible is the main duty of the developer.

Coding should be done according the coding standards specified before. After it passes all tests, it is moved to refactoring operation. After refactoring operation , it is again subjected to tests.

From time to time, set of question can come to the developer's mind. He should immediately ask these questions to the customer by him. The customer should respond these questions properly and should also direct the developer properly.

But the developers should still not trust the things customer says very much, should also get approval from others. Here especially users have a big duty. They should not direct wrongly about the subjects they are not sure of.

If the developers finish their own parts, they should reflect the change on the machine where integration is made and should provide the customers to do test. After this operation, to apply acceptance tests are the duty of customer. If there are parts

failing in the tests, relevant parts are informed to relevant developers and developers start to make new changes for the relevant parts.

One day of a developer is summarize as above.

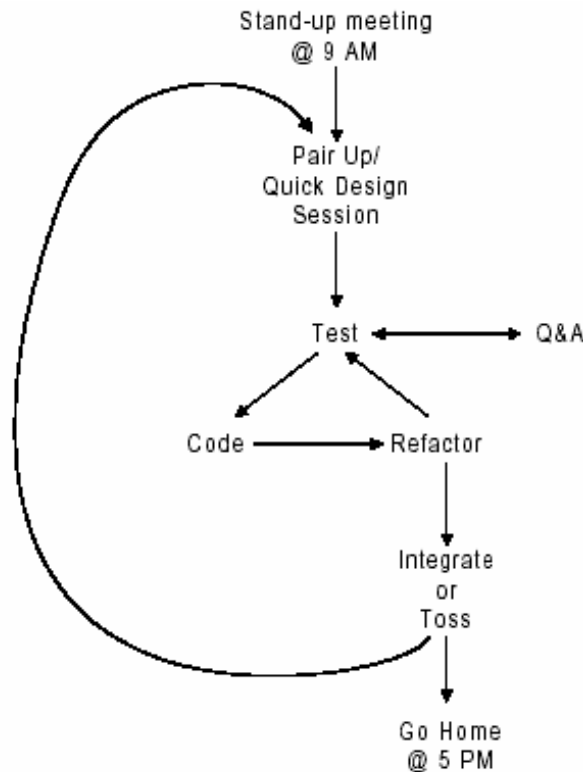


Figure 3.4 One day for developers.

In XP managers have also important duties. Manager is in fact a window opening outside. They arrange the team and try to solve the problems. They go in resource search for developers. Manager is in fact the person controlling the state of the project. He is a viewer. He controls whether the developers are on the right way or not. It is his duty to distribute the projects to the members of the team according to works.

Manager do not decide the priorities of works. The user already does this work. It is not the duty of the manager to distribute the works in developer terms. To define which work will last how long is also not the duty of the manager. In XP manager is generally responsible from the financial and the solution of the problems parts. From

financial and facilities view the source belongs to the manager. If appropriate conditions are not supplied to the team, team will not be beneficial enough and it will not be a healthy project.

Another duty of the manager is to find the members of the team. This can be supplied by a method of renting or transferring from another software company. Manager provides to form a relation between these individuals.

Sometimes developers need to get support from outside. In this situation, managers should provide developers to attend trainings by arranging training companies. In trainings, the efforts of the developers to learn the parts they don't will be the best answer to give the managers.

From time to time, managers should learn whether the developers have any trouble or boredom. They should keep the distance with the developers but not keeping it far, they should show that developers are valuable for them.

3.4 What Type Software Is Developed In Your Company?

The aim of this question is to learn what type software is developed in companies, what is the relation between customers and team members?

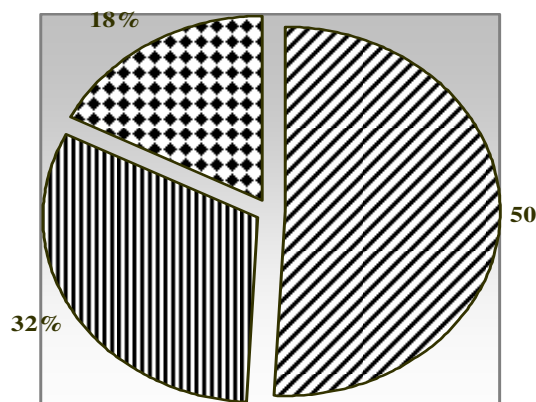


Figure 3.5 Software types which is developed in companies.

Table 3.4 Ratio of software types.

What type software is developed in your company?	
%50	Special for customers.
%32	We are a holding company and we have been developed to ourselves.
%18	General purpose software.

With which aim the projects are developed is important for success. If a Project with general aim is being developed, it should be controlled whether the demands are suitable for the general needs. Even if the developers have knowledge about the subject, the customers near them should show the regular directions.

As it is mentioned before, the outline of the project should be plotted well and the order of the project should be determined.

%82 of the part in the research develops projects special to the customers. Although a program with a low cost can be developed by this way, it is aimed to benefit from customer's experiences by developing a software special to the customer instead of a general project.

One who is inside the work knows it better than any one. Even if a developer is experienced enough, he may not have more knowledge than someone constantly doing the same work. For quality software, it should always be benefited from the customer's experiences.

3.5 How Many People Develop Software In Your Company?

Generally the survey is applied to companies having employees under 10. Naturally companies find suitable to develop project by keeping their employees in small numbers.

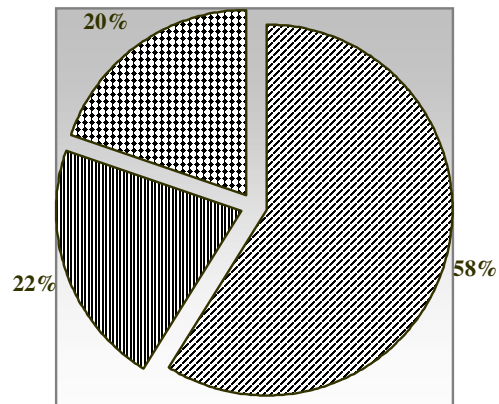


Figure 3.6 shows number of developers in company.

Table 3.5 Shows number of developers in company.

How many people develop software in your company?	
%58	Less than 10
%20	Between 10 and 20
%22	Greater than 20

3.6 What Do You Think About Team Members ?

What the developers and managers think about the size of the team in ideal is aimed to be listed in this question.

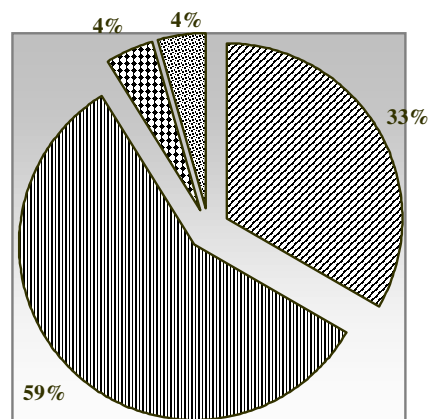


Figure 3.7 shows team members ratio.

Table 3.6 Table contains team members' ratio.

What do you think about team members?	
%59	3-6 members.
%33	1-3 members.
%4	9-12 members
%4	Greater than 12 members.

As it is mentioned before, opposite to traditional approach team is not only consist of developers and managers. It also includes the customers. Human means difference. People working in different jobs have different experiences, knowledge and talents. Some may be snob, some may think themselves as best and some may be very young.

People attended to the survey gave different answers.

- If your team is very big, it is difficult to control.
- The size of the team changes according to the size of the Project.
- If your team is experienced, you can keep the size of your team small.
- The size of the team is decided after the meetings with the customer.
- In a team there should be 2 or 3 specialist developers and 1 or 2 new graduate developers.

If a team is very big, there can be communication problems. The share and the management of collective codes may be distressing. Meetings may take long time. But XP tells that meetings inside the team should be done everyday. In these meetings, besides producing solution problems are determined. But this kind of meetings may not be suitable for every company. In this kind of meetings if it is not controlled, it can cause damages.

3.7 What Is The Frequency Of Customers' Meeting?

The aim of this question is to learn customers' relation between them and check customer on-site practices have been applied.

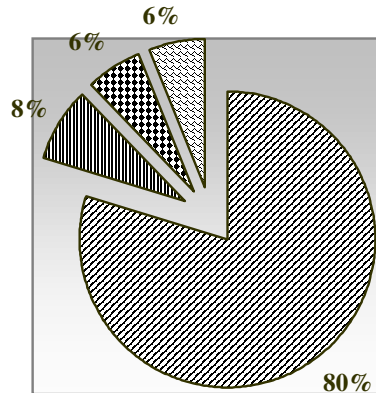


Figure 3.8 shows frequency of customers's meeting.

Table 3.7 Frequency of customers' meeting ratio.

What is the frequency of customers' meeting?	
%80	At any moment if necessary.
%8	During analyse.
%6	Once a month.
%6	Once a week

%80 of people attended to the survey, believes that it is necessary to meet with the customer when it is needed.

In traditional approach developer and customer come together during the analysis meeting. But with one meeting or limited number of meetings healthy decisions may not always be taken. That day the spoken person may not be well or may not help because of the fear to lose his job or may not prefer as he is not approving this project.

But XP wants developer to work with the customer at any moment. They want developers to get written feedbacks after the meetings. And they should give feedbacks to the customers as soon as possible.

But there are also participants to the survey who do not agree with XP's suggestion. The point they defend is that they feel indisposed from the interference of another person while making the software. While working on a critical operation,

customer's question about the previous subject may increase the risk of bug and cause defective operations. The aim to develop a program through the demands of customer can cause program with bugs to occur. To solve this problem, some developers consider the necessity of call centers. Developers should keep in touch with the customer only if it is urgent. Here, of course the important point is the necessity of people having well knowledge about the program to direct the customer correctly on the phone.

3.8 Do You Have A Test Unit In Your Company?

The aim of this question is to learn that exist a test unit in companies.

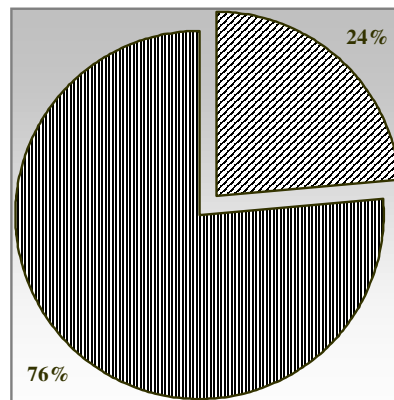


Figure 3.9 shows ratio that has test unit companies.

Table 3.8 Table shows ratio that has test unit companies.

Do you have a test unit in your company?	
%24	There is a test unit.
%76	There is not any test unit.

In XP there are two kinds of test. One of them is the unit tests. They are prepared and applied by the developers. Through these tests nothing is left in the air. If there is a missing thing, it is determined immediately. Unit tests are applied nearly every day. With the agent of unit tests developer gets feedback. He defines what is done and undone.

For the design of the system, unit tests have benefits. If program is coded first and then tested, the missing or incompatible points can be founded. But if test is made according to design and then coded compatible to this, code best fitting with the design occurs.

The second test module is acceptance tests. These tests are prepared by the customer. The user stories written during iteration are converted in to acceptance tests in the iteration planning meeting. One user story can be consisting of one or more acceptance tests. When user story is completed, by checking if it fits these tests, it can be decided whether the system is functionally compatible or not.

According to the survey, %72 of them do not have test module. Software companies actually do not give importance to test. Managers prefer to deliver the program immediately, instead of delivering late but well made. Generally it operates like this: program is coded. Developer tests it not very intensively. Then it is copied to the customer to be controlled. Later customer starts to use the program. He encounters with some bugs and informs the developer. Developer fixed the bug and fixes it. Then he sends it to the customer. But sometimes the change made at that moment damages another part. This process continues like this. And at the end of these operations, the customer does not have any confidence left to the program.

It is necessary to have test modules. Programs should be tested by these modules. If a customer finds a bug, it should be consulted to this module and if there is an operation not solved by this module, it should be transferred to the developer. After the solution of the developer, this module should send it to the developer by trying all the tests.

3.9 Have You Been Working At Home After Company About Your Job?

The aim of this question is to learn planning and 40-hour week.

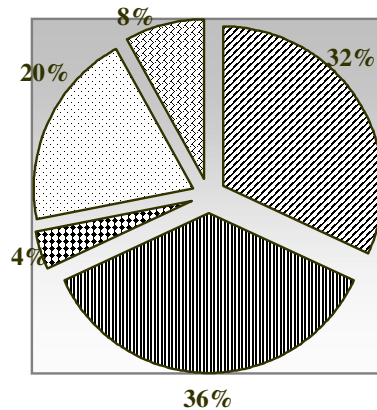


Figure 3.10 shows ratio that programmer's working time.

Table 3.9 Developers ratio of working at home after company about job.

Have you been working at home after company about your job?	
%36	No. Only I work at company. I pass the time with my family at home.
%32	Rarely.
%20	Two or three day of week.
%8	Once a week
%4	Every day

According to the survey, %32 of them does not take work to home. %64 of them rarely takes work to home.

In XP developers go to home at 5 o'clock. Everything manipulates in a planned way. That's why XP does not approve taking work away. In release planning meeting small releases are handled and in what period of time it is going to be coded is estimated. They do not want to encounter with any surprise.

In many companies design and coding is done by the same person. The designer passes to coding later. During these operations, the project velocity is also not determined. Despite the greatness of work, they want developers to complete and

determine the relevant work in a short period of time. Actually for the manager, the place where the work is being done is not important. Most of them work at home to overtake arrears of work. However, either the workload or the lacks of developers send invitation to bugs. In brief, this process causes damages then benefits. As a result of this, the relevant release of the program is known by that individual. In the event of quitting his job, the other members of the team will strand.

Why some developers want to work at home? This should not be permitted by the managers. Managers should suggest them to rest.

3.10 Do You Work Overtime In Your Company To Deliver Software At The Right Time? What Do You Think About Reasons?

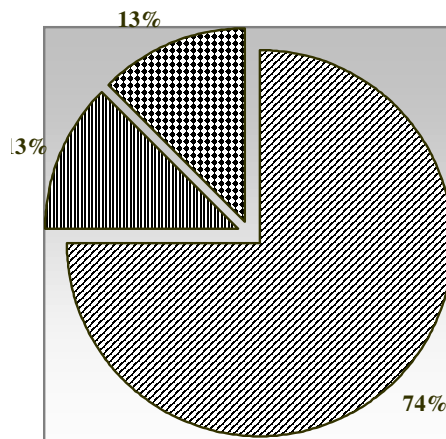


Figure 3.11 shows ratio that work overtime to deliver software at the right time

Table 3.10 Reasons that work overtime to deliver software.

Do you work overtime in your company to deliver software at the right time? What do you think about reasons?	
%74	Yes
%13	No
%13	Rarely

According to the survey, most of them like %74 seem not to deliver the work on time. As mentioned before, the main cause of it is being unplanned. But the participants have their own causes. These:

Insufficient developer: When 5 people try to complete the work that 10 people can do, works do not continue in a way it is planned and so it is not delivered on time.

Lack of analysis: During analysis, it is not treated attentively and this causes later the occurrence of the points not paid attention during the analysis.

Lack of planning: It is not healthy decided which work is priority and which one is in secondary priority. Works are not planned regularly.

Limitation of time: Lots of work is wanted to be done in a very short period of time.

The communication problem between Project offices in different offices: There is a communication problem between offices.

Insufficient capacity: The developers are unfamiliar with the technology going to be applied in software. The demand about the subject they do not know.

The wrong measurement of project velocity: The measurement of project velocity is not done or wrongly done.

The demands of customers: The features that the customer demands do not have an end. Someone should say "Stop!"

The promises given to the customer: While managers are negotiating with the customer, they give promises without checking whether the infrastructure is available or not and without applying to the ideas of developer.

The competition with other companies: It is hurried to show that better work can be done than competitor companies.

Extra demands of customer: Not rejecting the extra demands of customer.

Other problems: Other than software, problems about family, material and facilities. Managers do not do anything to solve these problems.

The communication problem between the customer and developer: From time to time the developer and the customer do not talk the same language. The developer or the customer may be technologically insufficient.

3.11 Do You Use Unified Modeling Language (UML) On Projects?

The aim of this question is to learn that using UML on their projects, exist metaphor, CRC Cards.

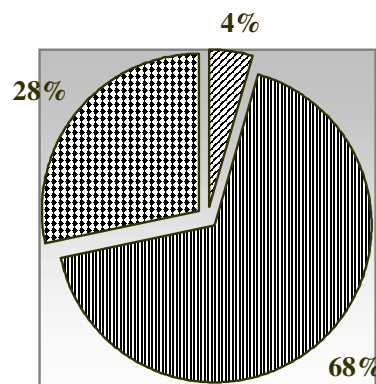


Figure 3.12 contains using UML ratio of companies.

Table 3.11 Table contains using UML ratio of companies.

Do you use Unified Modeling Language (UML) on projects?	
%68	We don't use UML on our projects.
%28	We think UML is uselessly.
%4	We use UML on our projects.

In XP, UML can be used. The UML diagram is in fact good in defining the communication between classes. It provides the developers to do their work fast. But some developers do not prefer using UML. Most of the participants of the survey do

not like UML. It is needed to update the UML diagram coordinately in a situation of change. The supply of this coordination is not very easy. From time to time the initial structure do not fit the structures at the end. But for this kind of situations, J Builder Enterprise Studio, Together Control Center, Rational XDE tools can be used. These tools can form the UML diagram automatically according to the code of the program. These diagrams can be renewed when wanted.

In XP, Class Responsibilities and Collaboration Cards (CRC Cards) that perform the same function with UML are used. The most important feature of the CRC Cards is its transformation the developer from a procedural structure to an object structure. By this way, CRC Cards also help the design.

The name of the object is written on the card. The responsibilities of this class are listed under the right side of the name. With which classes it is in relation and its details are defined in front of every responsibility.

Through CRC cards, objects are made to talk with each other. This causes to understand better over which structure the system is developed. The explanation of the design can be easily over these cards.

The most critical point of CRC cards is that design operation is only left on these cards. The information in CRC cards should be documented as design.

Although UML or CRC cards are not suitable for some developers, as it provides the explanation of the system, these should be applied. By this way whether there is any object can be understood by checking these cards. It is better than design document as it is visual and summary.

3.12 What Do You Think About Any Developer Talk Over Customers About System?

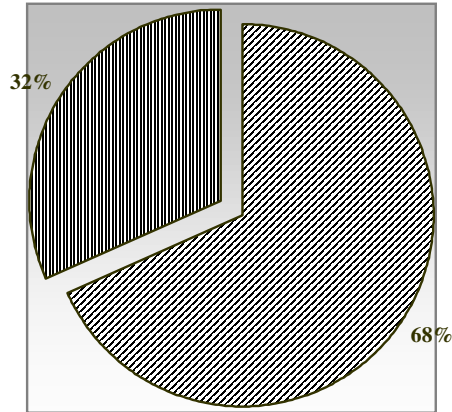


Figure 3.13 shows talk over customers about system is suitable or unsuitable ratio.

Table 3.12 Developers talk over customers about system.

What do you think about any developer talk over customers about system? Why?	
%68	Unsuitable
%32	Suitable

In XP developers and customers should negotiate with each other like a team and help each other in designing. The customers answer the questions of developers and the developers answer the questions of customers. The customer is informed about every change.

But in reality the customer can not spare this much time. The customer has also lots of things to do during the day. He should not retard these works. It can also be not possible for the customer to sit closely with the developer. This is valid between developers and customers only.

Another important subject is the necessity for the customer to adopt the project with self-sacrifice. In the project, the biggest communication between the customer and the developer will be during tests. There are actually 3 phases in the project. Release planning is the phase that customer determines the small releases and their

duration. Iteration planning is the phase where it is determined, which work is done in how much time.

The last phase is on the fly phase. The developers should have the sufficient knowledge to negotiate with the customer. Although in XP it is said that customer educate the developer, most people do not want to negotiate with the insufficient developer. Developer's suggestions of methods may not be compatible. When it is considered that the customer can not always know everything, in this situation it is the developer's duty to warn and direct the customer. And this can only be possible with a good knowledge of business. The negotiation of a developer experienced in his field with the customer is not a problem.

3.13 How Often Do You Make Meeting In Your Company To Discuss Software?

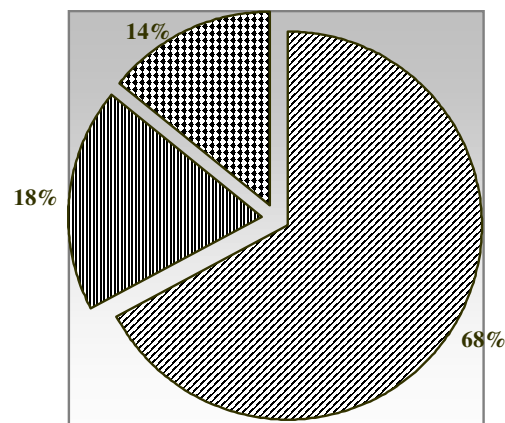


Figure 3.14 ratio of making a meeting in companies to discuss software.

Table 3.13 Frequency of meeting to discuss software.

How often do you make meeting in your company to discuss software?	
%68	Once a week
%18	Two or three days of week
%14	Only once during software

The purpose to ask this question is to state the frequency of the project meetings. Lots of people stated that there is one meeting in a week. But it is a very late period

of time. To measure the project velocity one meeting per week may be made, however the most suitable is to meet instantly when a situation about the project is encountered. XP, in fact do not determine this process as the customer and the developer are always together. Every moment is suitable to negotiate.

3.14 What Do You Do To Determine Needs Of Customers?

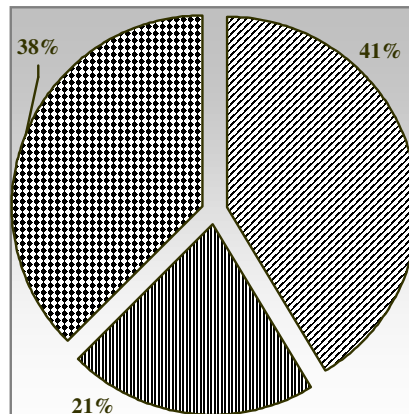


Figure 3.15 Determining needs of customers.

Table 3.14 Determination needs of customers.

What do you do to determine needs of customers?	
%41	Meeting
%38	Analysis
%21	Reports

In XP user stories are stated to determine these demands. They serve as use case but the same. They do the works that big need documents fulfill at the same time. But the most important feature is that it is method where customer determines his needs as he wishes without entering technical subject. There is not a specific format. He can specify his demands with his own sentences in a way as he wishes.

They are mixed with the documents where the demands are stated in traditional method. Or they are believed to be the same. However they are completely different things. The most important difference is the detail level. User stories should be

enough to determine the definition and the priority of the work. The remaining are the cases to be understood by negotiating face to face.

The developers read these user stories. If some parts are not clear, they negotiate with the customer and ask a set of questions. Later they inform how long it will take to finish these stories. This duration changes between 1 or 3 weeks. If it is considered that it will take more than 3 weeks, the necessity to divide this operation is decided and the operation is transformed into works of 3 weeks. If one work seems to be completed less than one week, in this situation it is combined with other works and transferred into a one week work. In release planning meeting between 60 and 100 user stories should be formed. It is necessary to plot the works as dependently as possible from each other. But later when they are integrated, they should form the whole.

Another difference between the user stories and demand documents is the distance focusing on the demands. In user stories the database structure and algorithms are certainly not stated. There are information that the customer and the developer can understand.

3.15 When The Customers See The Software System?

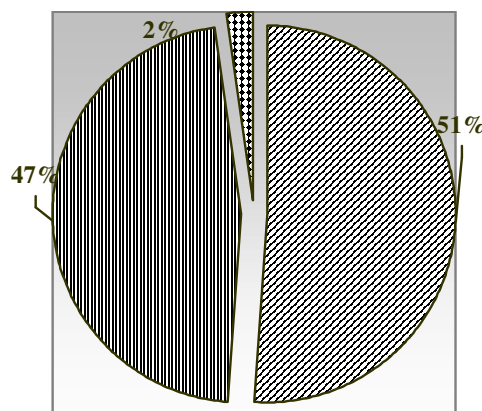


Figure 3.16 Customers see the software system.

Table 3.15 Customers see the software system.

When the customers see the software system?	
%51	When complete system
%47	At each weekend
%2	After completing data forms

Lots of people actually said that the customer can see the project when it is completed. Of course this prevents to notice the surprises to be encountered before in the project delivery. The right way is to provide the customer to have knowledge about the system by showing to the customer after every release has finished and doing it in versions.

3.16 If The Experienced Developer Leaved From Your Company, Is This Affected You Negative?

The aim of this question is to learn dependency of the developers and collective code ownership practice.

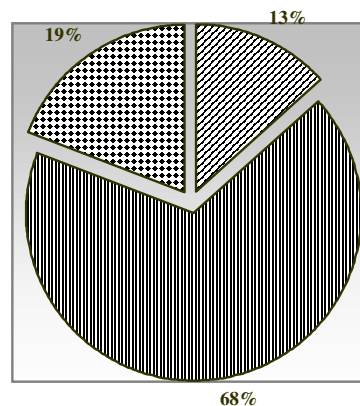


Figure 3.17 Dependency of developers ratio.

Table 3.16 Shows dependency of developers.

If the experienced developer leaved from your company, is this affected you negative?	
%68	This is a problem for us.
%19	Every developers are training each others so no problem
%13	There is no dependent code to any developer so there is no problem.

In Extreme Programming nobody is completely responsible for any code. Everyone is responsible for everyone's code. All the codes should be understandable to the developers. If one quits the job, the others are not affected negatively.

According to the survey, %68 of people is affected negatively from this situation. Here the most important question to be asked is, "why experienced developers do not educate other developers?" Pair programming may be applied for the education of developers. The biggest reason why it is not applied by most of the project managers is the cost. Project managers may be right here, but at least the critical points can be developed with more than one person and as a result more than one person can have knowledge about the code.

3.17 How Long Time Is A New Developer Adaptation Duration In Your Company?

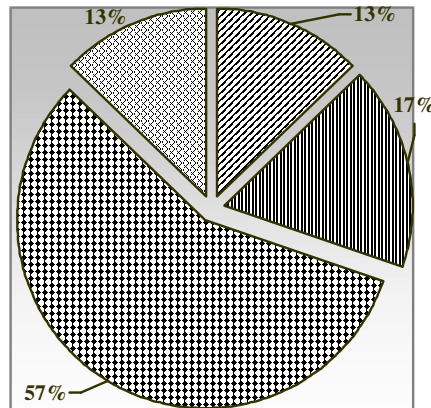


Figure 3.18 New developer adaptation duration.

Table 3.17 Adaptation duration of developers.

How long time is new developer adaptation duration in your company?	
%57	9-12 month.
%17	3-6 month
%13	Greater than 12 month
%13	1-3 month

With this question it is aimed to inquire the simplicity in software companies. When it is asked questions to lots of project managers, they claim that they make simple coding. But according to the adaptation durations in the results of the survey, it does not suit with the realities.

It should not be forgotten that simplicity removes the project risk, enables the adding of new features easily, reduce the dependence to a person and provides developers and managers to feel themselves better.

The long adaptation duration results from the insufficient simple coding. %74 of people has stated that the adaptation lasts more than 6 months in a company. It is quite long adaptation duration. The reduction of this duration can only be provided with simple design and simple code.

3.18 Some Managers Set Some Limits To Developers To Affective? For Example Tea, Coffee Limit. What Do You Think About These Limitations?

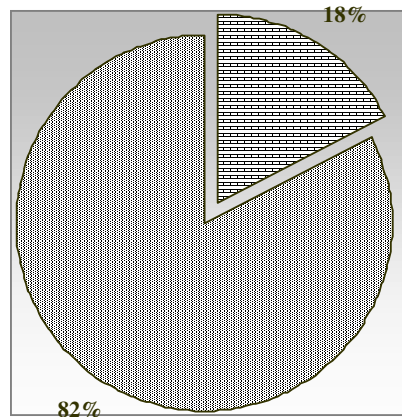


Figure 3.19 shows some limits to developers to be affective positive or negative?

Table 3.18 Some limits to developers.

Some managers set some limits to developers to affective? For example tea, coffee limit. What do you think about these limitations?	
%82	I don't agree with limitations.
%18	I agree.

The developers think that these limitations set by some managers are not beneficial.

An event told in software book even confirms this.

A manager of a company sees that people in the information processing department spend too much time near the coffee machine when they go to take some coffee. To prevent this, he puts the coffee machine away and orders some coffee to their desk at a certain period of time. After occurrence of bugs in the software are seen. The manager does not understand this situation. The reason of this is not understood while a more quality code is hoped to be coded. Later the coffee machine is put again and a camera is placed near it. He starts to observe. While the developers are taking coffee, it is observed that they are talking about program issues and they produce solutions. Through this, the manager understands that defective coding is prevented. A different point of view can put forward a new suggestion not thought before and can submit a solution. That's why it is necessary to take everyone's suggestion into consideration.

3.19 What Do You Think About Week Job Time?

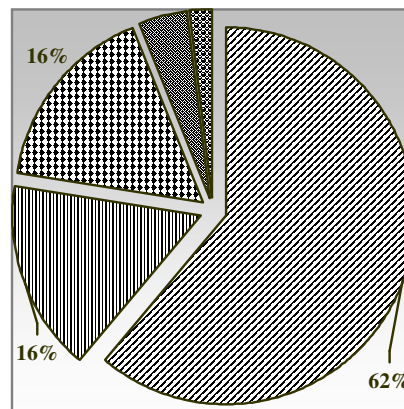


Figure 3.20 shows week job time.

Table 3.19 Week job time.

What do you think about week job time?	
%62	40 hours
%16	45 hours
%16	35 hours
%4	30
%2	Greater than 45 hours

Most of the participants consider working for 40 hours per week is ideal. If everything is planned, working for 40 hours is enough.

3.20 What Do You Do To Increase Communication Between Developers In Company?

Table 3.20 Increase communication between developers.

What do you do to increase communication between developers in company?	
	Social activities
	Training
	Tournaments
	Going cinemas
	Nothing

The problem of communication is the main problem of developers. It is a problem to be got over. A developer sitting in front of a computer from morning till night is not able to get across and this causes him to diverge from people. Because of this it is necessary to make them spend time away from stress by providing them social activities. By this way they will understand how valuable they are for their managers.

3.21 Do You Have Any Coding Standarts?

The aim of this question is to learn they write codes on standards or not.

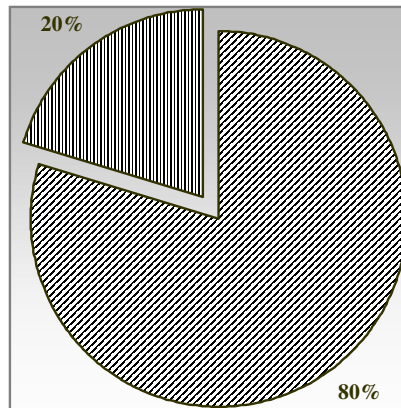


Figure 3.21 Coding standards.

Table 3.21 Coding standards ratio.

Do you have any coding standards?	
%80	Yes.
%20	No.

XP provides its coding standards with pair programming and collective code ownership. A definite code standard is inevitably formed as everyone can constantly examine everyone's code.

Although coding standards are seen as an extra work load, the application of standards determined in the beginning in fact provides benefit.

The control of these standards can be performed by system metaphor. By checking the metaphor, it can be understood whether the denomination of objects and methods are noticed and whether the defined standards are fitted. With the agent of this, benefit can be provided both in short term and long term. In short term it can be understood which objects there are and what they do. However in long term a simple structure is formed. By this simple structure, the adaptation of the new developer can be provided in a short period of time by explaining him the standards.

In fact at first sight, trying to fit these standards can annoy developers. To try to fit the new standards in his new company, by leaving the standards that he defined on

his own or in his previous company is not something easy. It is difficult to remove the customs. It is necessary to endure this work as it will provide more benefit later.

Although the participants state that there is coding standard in their company, this rate is not that much high. As the adaptation duration is also expected to be less at the same rate.

Whether there is a coding standard, it should be inquired at which rate it is applied.

3.22 What Do You Do For Junior Developers To Interfere Code?

Continuous learning is important for quality software.

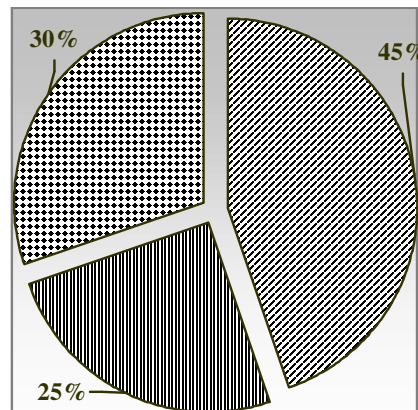


Figure 3.22 Shows doing for junior developers to interfere code.

Table 3.22 Doing for junior developers to interfere code.

What do you do for junior developers to interfere code?	
%45	Examine code documentation
%30	Examine example code in company
%25	Code training

Continuous learning is part of extreme programming and all practices take attention to this. Continuous learning is the heart of extreme programming. In

iteration meetings both the developer and the customer learn something. With the agent of pair programming, test and refactor developers always learn something. The project coach learns the communication. It is decided how to approach towards the problems. Through feedback, developers have knowledge about the program's functioning. As a result in XP some things are continuously learned.

If a team really applies the feedback and the communication values of XP, it will see its benefit in a short time. Project managers are responsible for the developers' notice of this situation.

Works done before are like this:

In 2001, January the 27th Bob and Eric formed the framework of XML by doing refactoring operations over the XML placed on the database layers. By this way they have knowledge about XML framework. They noticed that they were able to learn easily with the help of refactoring. (Senge, P,1990)

During pair programming partners warn each other when they do mistakes. The one, who does the mistake, notices his mistake. After some time they like this situation.

“We were in a difficult situation and did not know how to solve it. The problem is explained to the coach. It is negotiated with the customer. After a 10 minutes meeting, the suggestion made by the customer is transmitted. Really a very good solution is suggested. It is noticed that being a developer is not always mean to find best solutions. (Jerry) (Senge, P,1990)

Peter Senge in his book called “The Fifth Discipline: The Art & Practice of the Learning Organization” says, “We are actually the parts of a team. If you are in a team that finds extraordinary solution and has more collective purposes than individual purposes, you are the part of that team at that time. In a team like this by solving problems, you start to find extraordinary solutions even to the biggest

problems. From now on this is s feature that you gained from your experiences.
(Senge, P,1990)

3.23 Does He Or She Make Some Changes On Your Code That Is Written By You Easily?

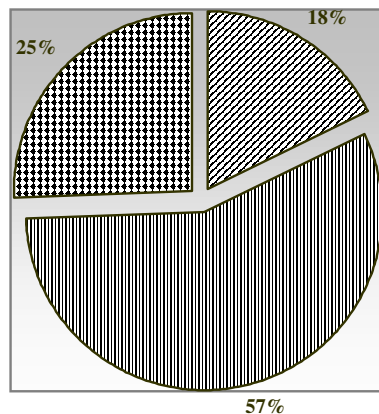


Figure 3.23 Collective code ownership.

Table 3.23 Collective code ownership.

Does he or she make some changes on your code that is written by you easily?	
%57	Only I must.
%18	We have some standards so that he can understand easily.
%25	Mostly understandable if not he ask me.

In XP with collective code ownership any one can make a change in any part of the program without waiting any one. After test is well done, changes can be made easily.

By examining the classes written for automatic tests, it is understood how a class is called and which operation it performs. Through these tests passed from hand to hand, one can have knowledge about how a system operates. A part of the software can be known better by some of the developers and this is very normal. In this kind of situation, beside the examination of automatic tests, he can explain the parts that he knows well to another developer. If one is ill, he can ask the partner of him or can

have information from another person that made refactor before to the relevant part. By this way, without depending on one person, the problem can be solved.

But if it is depended on one person, then it is waited him to come and can not be interfered. In a situation like this, if a customer demands a new change, even in urgent situations a developer can not respond customer's demand.

If a change is going to be made on coding, it is necessary to make this operation inside the company. The reality not to be forgotten is that the solution developed at home, near the customer does not produce healthy solutions.

3.24 What Do You Think About Code Ownership?

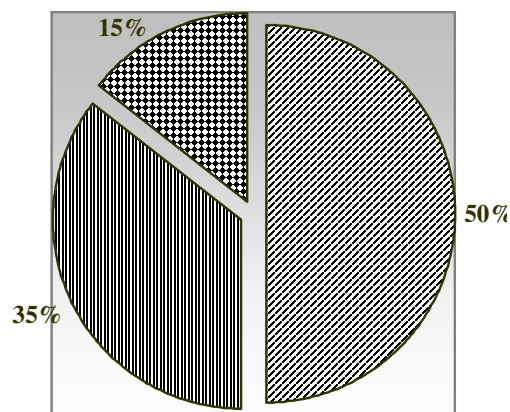


Figure 3.24 Collective code ownership.

Table 3.24 shows collective code ownership

What do you think about code ownership?	
%50	Every program and sub program should have owners and changes should be making with knowledge of them.
%35	There should not be any owner. Everyone can change any time.
%15	There should be sub program unit and make changes with them.

Collective code ownership provides developers to feel themselves better. When he sees himself making changes on the programs, he psychologically feels comfortable. It is a morale resource for the junior developers to be seen by the customer that are

able to solve the problems. The lack of code ownership in XP is important for educating developers more rapidly.

3.25 Do You Think, Pair Programming Is Useful For Quality Software?

The aim of this question is to learn what they think about pair programming.

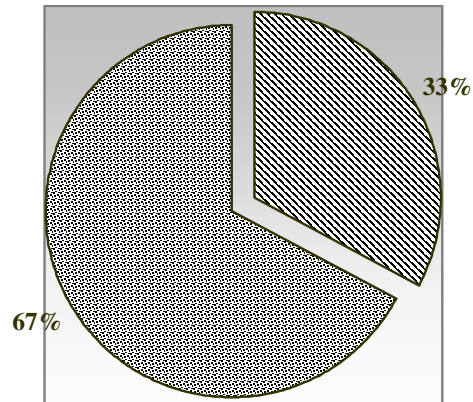


Figure 3.25 Pair programming ratio.

Table 3.25 Table shows pair programming ratio

Do you think, pair programming is useful for quality software?	
%67	I don't think it is useful.
%33	I think it is useful.

Opposite to XP, pair programming is not seen useful or is found difficult to apply by most people. %67 of the participants to the survey thinks that it is useful. Whereas, XP especially emphasizes on pair programming. It is considered that work to be done in front of one computer with one mouse, one keyboard and two monitors are going to produce useful results. It is also believed that their success can only be possible only if the problems of the developers are solved and conform is provided.

With pair programming it is reached at a so advanced level that developers can easily understand what they think towards the events and how should they act. This is a feature gained with the application of pair programming.

In Utah University, in 1999 spring term, pair programming is tried to be applied in the web programming course. Students are educated to do their all homework with their pair partners in pair groups. With this method, the benefit of pair programming is seen. How quality is the homework, is appeared. Through pair programming, they learned more easily and fast. With the confidence to each other, they carried out some things more quickly.

The coding done by both junior and senior developer is better than coding done by one developer. Software having less bugs can be developed.

Lary Constantine is a columnist. He visits P.J. Plaugher' Software Company. He witnesses to the application of pair programming in this company. He notices that 2 developers work in front of one computer. And he mentions these in his report: "Developers coding in pair deliver the programs rapidly and they can test more rapidly and better according to the system only made. The codes formed by two minds are more efficient, more solid and better. Two developers being in this work is certainly not a surplus, it is the cause of forming more efficient and quality code releases."

Professor Nosek in Temple University is challenging the problems by using his own equipments. For this, he makes a study on 15 developers working full time. He makes 5 developers work alone and the other 10 work in pairs. All the conditions are same for both individual developers and pair developers. Same equipments are used. When he evaluates the results statistically, he obtains a result that managers can not believe. According to the results it is stated that developers working in groups develop better code than developers working alone and they are %40 more rapid. The algorithm that they applied is more efficient than single developers. (Constantine, L.L,2003)

Beside all these experiments, when it is asked to the developers, most of them accept that they develop better code while working in pairs and they say that they develop very solid codes having nearly no bugs.

Again in a university, 20 specialists, who do not know ASP but know programming, by working with 20 inexperienced developers, provided all individuals make better coding at the end of a 11 week study. All the students made the programming part in pairs but made the test parts one by one. And what they observed during the tests is the appearance of very solid codes. (Williams, L., 1999)

One of the students used these statements, “While being in front of the machine, I resemble myself to a driver. I concentrate to my work very much. In fact I try to show my experience and my good sides in coding to my partner. When I succeed it, I feel myself better. Whenever I make a mistake, I know that I will be warned by my partner and this provides me to code comfortably. If I am not the driver, in this situation I control the code releases made by my partner. I try to be careful for not being a bug. If a bug occurs and if I did not able to capture it, I think that it is my fault. (Williams, L., 1999)

According to a customer’s point of view, quality software depends on: 1- a product that the customer demands, 2- the responds that he wants to get, 3- the coding structure without bugs.

Pair programming contributes to all these clauses. To the students, two minds are better than one mind. 80 pieces of the homework given in the university are delivered on time and the average of the homework is 98.

When intermediate exam and homework are given all alone to the students studying in groups, it is seen that the average decreased to 78.1. The standard drift is measured as 20.91.

When the average of homeworks done in groups is measured as 97.9, the standard drift decreases to 6.74. This shows that when students are in groups, they are more careful and do few mistakes.

	Individuals	Collaborative Teams
Program 1	73.4%	86.4%
Program 2	78.1%	88.6%
Program 3	70.4%	87.1%
Program 4	78.1%	94.4%

Figure 3.26 Percentage of Test Cases Passed on Average

When it is interviewed with the students, lazy students even assess their success to pair programming. They say, “We see that we can do everything with my partner. We trust the programming done in pairs.” The statement of one student even shows that pair programming has contribution in the capture of bugs. “One day I did all the tests about the program. I was sure about the quality of the project. I gave the project to my partner and I think that he could not find bug. But in 2 minutes my partner found the bug. And there we understood that working in pairs is always better.” (Kessler, R. & Williams A., 2000)

In the rates above, it is seen that how low the rate of error in the programs is made in pair programming.

The students also liked pair programming. 84% of the students state that homeworks in pair become more enjoyable with pair programming. 84% of them again stated that they learned ASP in a more rapid and better way. They said that they learned quickly by interfering each other in the parts they did not understand. They stated that they thought ASP to each other, like this.

“When I declare an opinion, I concentrate on what he is going to say. I try to be sure if he understood or wrongly understood what I said. If he understands the problem well, he is going to help me. At the same time, he can help us to discover

some of the errors. That's why I can not decide without negotiating with my partner." (Talbot, N., 2001)

When there is a bug in single programming, developer focuses on that bug very much and the big picture is being forgotten. From time to time it is even thought that the problem is very big.

Of course in transition to pair programming, it is necessary that developers get free of some obstacles. First, developers should understand the benefits of pair programming and should accept that more quality implementations are made than implementations done on their own.

Secondly, they should confidentially share their studies and should explain their experiences to their partners on behalf of increasing quality. They should be patient on the point of explaining to their partners. Even if he does not understand, he should explain again and should not forget that his partner helps him in finding the bugs. People making pair programming should be open to structural criticisms and should value all the suggestions made for the quality of the system.

In pair programming it is not acted with a sense of first design and then coding. On the keyboard design can be done instantly. Because of this, if it is thought that one person is doing this, it is not possible to create healthy designs. With only two people better designs can be made.

Some developers may not like the other's looking at own code. In fact, through pair programming readable simple coding is performed away from bugs. However, the programmer bugs can already be found by compilers. Even Stephen King made his editor read his book before it was published. After his editor corrected the grammar mistakes, he published his book. (N. V. Flor & E. L. Hutchins, 1991)

In pair programming developers meet in a room. This can cause the discomfort of working atmosphere. Toys distracting attention, internet, noise, food etc can decrease the performance of the developers. Thus it is necessary to pay attention to this.

In fact XP is a methodology that has lots of flexibility contrary to the chain of rules that most of them say. One developer starts coding with another developer. The customer writes informally his/her demands on the user story cards. The developer takes this cards and starts implementing without any formal design process. For this operation, before it is decided by who the coding is done, change is bravely done. When XP's principals are examined, it is seen that they are simple. The most difficult one is pair programming.

But there are also situations where pair programming can not be implemented besides its benefits. Design can be very bad and codes can be illegible. Some developers may not be socially active. He may not get across with another person. Communication starts with talking and asking question. However communication continues or finishes according to the reasonability of the questions. A very absurd question from a developer may annoy developers. This is like the problem between a child who can draw a tree and who can not. When it is thought that you are insufficient in approach to the problems, no one wants to do pair programming. One partner being better than the other may be a problem. One of the partners sits and stays passive. As long as junior developers do not ask questions, they can not have knowledge about the system. And thus senior developers should make their partners have knowledge about the subject by explaining to their partners what they did.

The partner may not have used perfume while coming in the morning. This can annoy while working together. The food eaten in the lunch time may cause the decrease of business tempo.

In a system where more than one actor taking part, differences occur because of three causes.

1- Different actors may state differences as priority. 2- Information about the system may be obtained from different ways. 3- He/she may approach problems differently because of functional responsibilities.

Despite of the benefits of pair programming, managers do not prefer it. They do not find two people being appointed to one work reasonable.

It should be agreed on a common point for the managers and developers. Instead of a complete pair programming, it should be used in critical points by plotting the standards well. By this way the occurrence of bugs should be prevented. After the people coding with pair programming, it should not be forgotten that they will have difficulty while coding alone.

3.26 What Do You Think About YAGNI (You Aren't Going To Need It)?

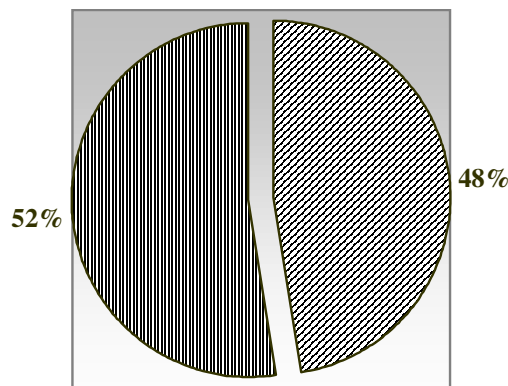


Figure 3.27 You Aren't Going to Need IT (YAGNI) ratio.

Table 3.26 YAGNI ratio

What do you think about YAGNI (You Aren't Going to Need It)?	
%52	I don't agree.
%48	I agree

52 % of the participants do not agree with this idea. XP rejects doing the works not demanded. In a project only 10 % of the works done without needed are used. 90% of it is a waste of time in vain. Thus, the necessity of not doing when it is not needed is emphasized.

However, here it should not be forgotten that customer may not always know everything. It is necessary to inquiry critical operations. Target oriented questions should be asked and it should be sure about the customer demands. Approval should be received from few people about the subject if it is needed.

3.27 Do You Think Refactor Is Applied On Working Complex Code?

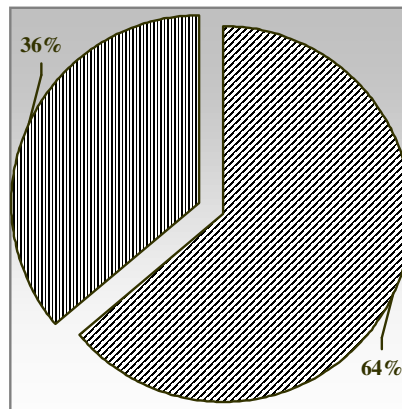


Figure 3.28 Refactor is applied on working complex code.

Table 3.27 Applied refactor on working complex code.

Do you think refactor is applied on working complex code?	
%64	Yes
%36	No

Refactoring operation is to make a change on code release on condition that not a change is made on the system's operating logic. These changes are in fact a series of small operations.

After every little change, it should be controlled whether the system is operating properly.

The use of stables defined in code by withdrawing them to variables. At that moment when the change is made, although it is seemed beneficial, it will give better results than circulating all the codes. (Refactoring (n.d.))

Here are some refactoring examples:

Table 3.28 Consolidate duplicate conditional fragments

<pre> if (isSpecialDeal()) { total = price * 0.95; send(); } else { total = price * 0.98; send(); } </pre>
<pre> if (isSpecialDeal()) total = price * 0.95; else total = price * 0.98; send(); </pre>

Table 3.29 Decompose conditional

<pre> if (date.before (SUMMER_START) date.after(SUMMER_END)) charge = quantity * _winterRate + _winterServiceCharge; else charge = quantity * _summerRate; </pre>
<pre> if (notSummer(date)) charge = winterCharge(quantity); else charge = summerCharge (quantity); </pre>

Table 3.30 Encapsulate field

<i>Make it private and provide accessors.</i> <pre> public String _name </pre>
<pre> private String _name; public String getName() {return _name;} public void setName(String arg) {_name = arg;} </pre>

Extract Class: Create a new class and move the relevant fields and methods from the old class into the new class.

Table 3.31 Extract class

<i>PERSON</i> <i>Name</i> <i>Officeareacode</i> <i>Officenumbr</i> <i>Get_telephone_number</i>
<i>PERSON</i> <i>Name</i> <i>Get_telephone_number</i> <i>TELEPHONE NUMBER</i> <i>Areacode</i> <i>Number</i> <i>Get_telephone_number</i>

Hide Method: A method is not used by any other class. Make the method private.

Table 3.32 Hide method

<i>EMPLOYEE</i> <i>+METHOD</i>
<i>EMPLOYEE</i> <i>-METHOD</i>

Inline Method: A method's body is just as clear as its name. Put the method's body into the body of its callers and remove the method.

Table 3.33 Inline method

<pre>int getRating() { return (moreThanFiveLateDeliveries()) ? 2 : 1; } boolean moreThanFiveLateDeliveries() { return _numberOfLateDeliveries > 5; }</pre>
<pre>int getRating() { return (_numberOfLateDeliveries > 5) ? 2 : 1; }</pre>

Table 3.34 Inline temp

<i>double basePrice = anOrder.basePrice();</i> <i>return (basePrice > 1000)</i>
<i>return (anOrder.basePrice() > 1000)</i>

Introduce Assertion: A section of code assumes something about the state of the program. Make the assumption explicit with an assertion

Table 3.35 Introduce assertion

<i>double getExpenseLimit() {</i> <i>// should have either expense limit or a primary project</i> <i>return (_expenseLimit != NULL_EXPENSE) ?</i> <i> _expenseLimit:</i> <i> _primaryProject.getMemberExpenseLimit();</i> <i>}</i>
<i>double getExpenseLimit() {</i> <i>Assert.isTrue (_expenseLimit != NULL_EXPENSE _primaryProject != null);</i> <i>return (_expenseLimit != NULL_EXPENSE) ?</i> <i> _expenseLimit:</i> <i> _primaryProject.getMemberExpenseLimit();</i> <i>}</i>

Introduce Explaining Variable: Have a complicated expression. Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose.

Table 3.36 Introduce explaining variable

<i>if ((platform.toUpperCase().indexOf("MAC") > -1) &&</i> <i> (browser.toUpperCase().indexOf("IE") > -1) &&</i> <i> wasInitialized() && resize > 0)</i> <i>{</i> <i> // do something</i> <i>}</i>
<i>final boolean isMacOs = platform.toUpperCase().indexOf("MAC") > -1;</i> <i>final boolean isIEBrowser = browser.toUpperCase().indexOf("IE") > -1;</i> <i>final boolean wasResized = resize > 0;</i> <i>if (isMacOs && isIEBrowser && wasInitialized() && wasResized)</i> <i>{</i> <i> // do something</i> <i>}</i>

Introduce Parameter Object: Have a group of parameters that naturally go together. Replace them with an object.

Table 3.37 Introduce parameter object

<i>AmountInvoicedIn(start:Date,end:Date)</i>
<i>AmountInvoicedIn(DateRange)</i>

Move Class: Have a class that is in a package that contains other classes that it is not related to in function. Move the class to a more relevant package. Or create a new package if required for future use. Classes are often created in packages close to where they are being used; this can make sense until the class starts to be re-used by other parts of the product. The package in question might also have just become too big. It is often better to move to this class to a package more related to it in form or function. This can help remove complex package level dependencies and make it easier for developers to find and re-use classes.

If there are many dependencies for the class within its own package, then Extract Class could be used first to split out the relevant parts.

Table 3.38 Move class

<i>class org.davison.ui.TextThing</i> <i>class org.davison.ui.TextProcessor</i> <i>class org.davison.log.Logger</i> <i>depends on</i> <i>class org.davison.util.StringUtil</i>
--

Move field: A field is, or will be, used by another class more than the class on which it is defined. Create a new field in the target class, and change all its users

Table 3.39 Move field

<i>Class1</i> <i>AField</i>	<i>Class2</i>
<i>Class1</i>	<i>Class2</i> <i>AField</i>

Move method: A method is, or will be, used by another class more than the class on which it is defined. Create a new method with a similar body in the class it uses most. Either turn the old method into a simple delegation, or remove it altogether.

Table 3.40 Move method

<i>Class1</i> <i>Amethod()</i>	<i>Class2</i>
<i>Class1</i>	<i>Class2</i> <i>Amethod()</i>

Parameterize Method: Several methods do similar things but with different values contained in the method body. Create one method that uses a parameter for the different values.

Table 3.41 Parameterize method

<i>Employee</i> <i>FivePercentRaise()</i> <i>TenPercentRaise()</i>
<i>Employee</i> <i>Raise(percentage)</i>

Preserve Whole Object: You are getting several values from an object and passing these values as parameters in a method call. Send the whole object instead.

Table 3.42 Preserve whole object

<i>int low = daysTempRange().getLow();</i> <i>int high = daysTempRange().getHigh();</i> <i>withinPlan = plan.withinRange(low, high);</i>
<i>withinPlan = plan.withinRange(daysTempRange());</i>

Table 3.43 Pull up constructor body

<pre> class Manager extends Employee... public Manager (String name, String id, int grade) { _name = name; _id = id; _grade = grade; } </pre>
<pre> public Manager (String name, String id, int grade) { super (name, id); _grade = grade; } </pre>

Pull Up Method: Have methods with identical results on subclasses. Move them to the super class.

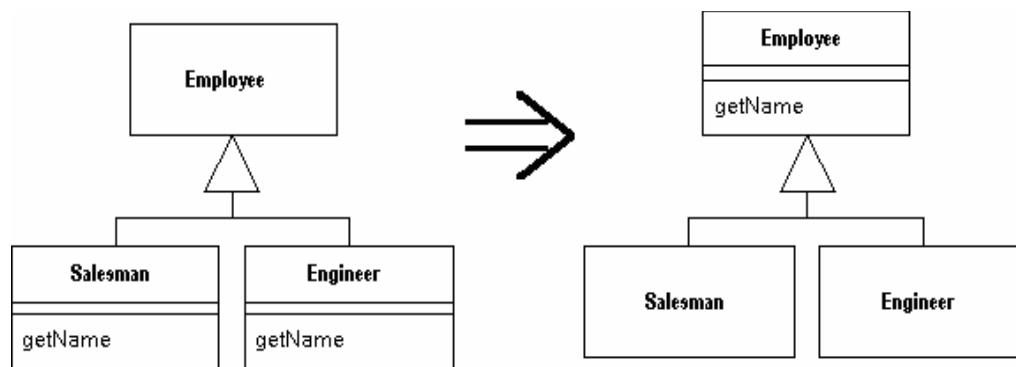


Figure 3.29 Employee, Salesman, Engineer class diagrams.

Push Down Field: A field is used only by some subclasses. Move the field to those subclasses.

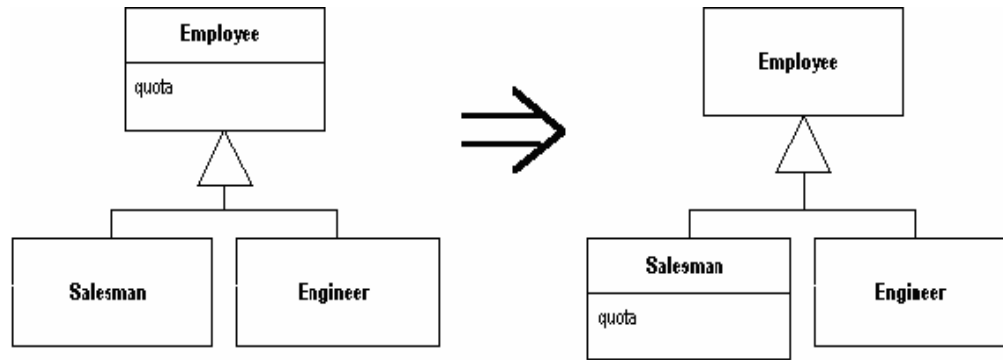


Figure 3.30 Employee, Salesman, Engineer class diagrams.

Reduce Scope of Variable: Have a local variable declared in a scope that is larger than where it is used. Reduce the scope of the variable so that it is only visible in the scope where it is used. There are several problems with local variables declared in too large scopes. One is that the variable may be declared but never used

Table 3.44 Reduce scope of variable

```

void foo()
{
    int i = 7;

    // i is not used here

    if (someCondition)
    {
        // i is used only within this block
    }

    // i is not used here
}
  
```

```

void foo()
{
    // i can not be used here

    if (someCondition)
    {
        int i = 7;

        // i is used only within this block
    }

    // i can not be used here
}
  
```

Remove Assignments to Parameters: The code assigns to a parameter. Use a temporary variable instead.

Table 3.45 Remove assignments to parameters

<i>int discount (int inputVal, int quantity, int yearToDate) { if (inputVal > 50) inputVal -= 2;</i>
<i>int discount (int inputVal, int quantity, int yearToDate) { int result = inputVal; if (inputVal > 50) result -= 2;</i>

Remove Double Negative: Have a double negative conditional. Make it a single positive conditional.

Table 3.46 Remove double negative

<i>if (!item.isNotFound())</i>
<i>if (item.isFound())</i>

Remove Setting Method: A field should be set at creation time and never altered. Remove any setting method for that field.

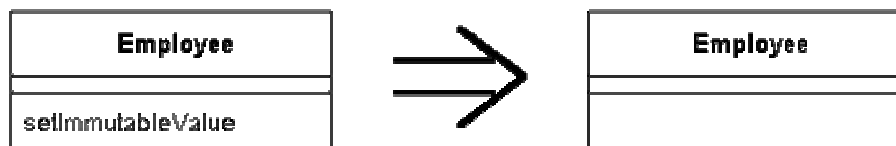


Figure 3.31 Remove setting method.

Replace Array with Object: Have an array in which certain elements mean different things. Replace the array with an object that has a field for each element.

Table 3.47 Replace array with object

<i>String[] row = new String[3]; row [0] = "Liverpool"; row [1] = "15";</i>
<i>Performance row = new Performance(); row.setName("Liverpool"); row.setWins("15");</i>

Replace Assignment with Initialization: Have code that first declares a variable and then assigns a value to it. Make it into a direct initialization instead

Table 3.48 Replace Assignment with Initialization

<pre>void foo() { int i; // ... i = 7; }</pre>
<pre>void foo() { // ... int i = 7; }</pre>

Table 3.49 Replace Constructor with Factory Method

<pre>Employee (int type) { _type = type; }</pre>
<pre>static Employee create(int type) { return new Employee(type); }</pre>

Replace Data Value with Object: Have a data item that needs additional data or behavior. Turn the data item into an object.

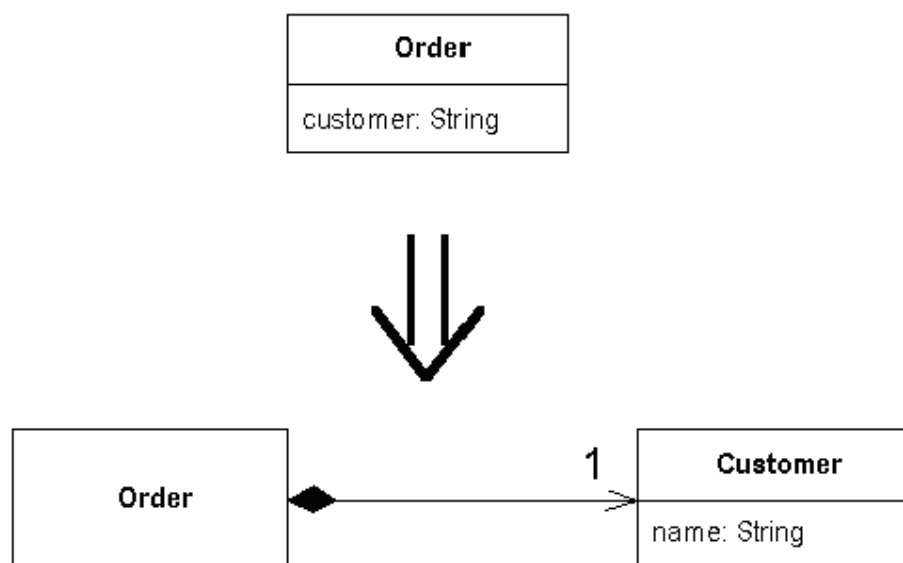


Figure 3.32 Replace data value with object.

Replace Error Code with Exception: A method returns a special code to indicate an error. Throw an exception instead.

Table 3.50 Replace Error Code with Exception

<pre>int withdraw(int amount) { if (amount > _balance) return -1; else { _balance -= amount; return 0; } }</pre>
<pre>void withdraw(int amount) throws BalanceException { if (amount > _balance) throw new BalanceException(); _balance -= amount; }</pre>

Replace Iteration with Recursion: There is a loop, and it is not obvious what iteration is doing. Replace Iteration with recursion.

Table 3.51 Replace iteration with recursion

<pre>unsigned greatest_common_divisor (unsigned a, unsigned b) { while (a != b) { if (a > b) { a -= b; } else if (b > a) { b -= a; } } }</pre>
<pre>unsigned greatest_common_divisor (unsigned a, unsigned b) { if (a > b) { return greatest_common_divisor (a-b, b); } else if (b > a) { return greatest_common_divisor (a, b-a); } else // (a == b) { return a; } }</pre>

Replace Magic Number with Symbolic Constant: There is a literal number with a particular meaning. Create a constant, name it after the meaning, and replace the number with it.

Table 3.52 Replace Magic Number with Symbolic Constant

<pre>double potentialEnergy(double mass, double height) { return mass * height * 9.81; }</pre>
<pre>double potentialEnergy(double mass, double height) { return mass * GRAVITATIONAL_CONSTANT * height; } static final double GRAVITATIONAL_CONSTANT = 9.81;</pre>

Replace Nested Conditional with Guard Clauses: A method has conditional behavior that does not make clear what the normal path of execution. Use Guard Clauses for all the special cases.

Table 3.53 Replace nested conditional with guard Clauses

<pre>double getPayAmount() { double result; if (_isDead) result = deadAmount(); else { if (_isSeparated) result = separatedAmount(); else { if (_isRetired) result = retiredAmount(); else result = normalPayAmount(); } } return result; };</pre>
<pre>double getPayAmount() { if (_isDead) return deadAmount(); if (_isSeparated) return separatedAmount(); if (_isRetired) return retiredAmount(); return normalPayAmount(); };</pre>

Replace Parameter with Explicit Methods: There is a method that runs different code depending on the values of an enumerated parameter. Create a separate method for each value of the parameter.

Table 3.54 Replace parameter with explicit methods

<pre> void setValue (String name, int value) { if (name.equals("height")) { _height = value; return; } if (name.equals("width")) { _width = value; return; } Assert.shouldNeverReachHere(); } </pre>
<pre> void setHeight(int arg) { _height = arg; } void setWidth (int arg) { _width = arg; } </pre>

Replace Parameter with Method: An object invokes a method, and then passes the result as a parameter for a method. The receiver can also invoke this method. Remove the parameter and let the receiver invoke the method.

Table 3.55 Replace parameter with method

<pre> int basePrice = _quantity * _itemPrice; discountLevel = getDiscountLevel(); double finalPrice = discountedPrice (basePrice, discountLevel); </pre>
<pre> int basePrice = _quantity * _itemPrice; double finalPrice = discountedPrice (basePrice); </pre>

Replace Recursion with Iteration: There is a code that uses Recursion and is hard to understand. Replace Recursion with Iteration. Recursion is often used without considering alternatives before using it.

Table 3.56 Replace recursion with iteration

<pre> public void countDown(int n) { if(n == 0) return; System.out.println(n + "..."); waitASecond(); countDown(n-1); } </pre>

```

public void countDown(int n) {
    while(n > 0) {
        System.out.println(n + "...");
        waitASecond ();
        n -= 1;
    }
}

```

Replace Temp with Query: You are using a temporary variable to hold the result of an expression. Extract the expression into a method. Replace all references to the temp with the expression. The new method can then be used in other methods.

Paul Haahr pointed out that you couldn't do this refactoring if the code in between the assignment to the temp and the use of the temp changes the value of the expression that calculates the temp. In these cases the code is using the temp to snapshot the value of the temp when it's assigned. The name of the temp should convey this fact.

Table 3.57 Replace temp with query

<pre> double basePrice = _quantity * _itemPrice; if (basePrice > 1000) return basePrice * 0.95; else return basePrice * 0.98; </pre>
<pre> if (basePrice() > 1000) return basePrice() * 0.95; else return basePrice() * 0.98; ... double basePrice() { return _quantity * _itemPrice; } </pre>

Reverse Conditional: There is a conditional that would be easier to understand if it is reversed its sense. Reverse the sense of the conditional and reorder the conditional's clauses.

Table 3.58 Reverse Conditional

<pre> if (!isSummer(date)) charge = winterCharge(quantity); else charge = summerCharge(quantity); </pre>
<pre> if (isSummer(date)) charge = summerCharge(quantity); else charge = winterCharge(quantity); </pre>

Self-Encapsulate Field: It is accessed a field directly, but the coupling to the field is becoming awkward. Create getting and setting methods for the field and use only those to access the field.

Table 3.59 Self-Encapsulate Field

<pre> private int _low, _high; boolean includes (int arg) { return arg >= _low && arg <= _high; } </pre>
<pre> private int _low, _high; boolean includes (int arg) { return arg >= getLow() && arg <= getHigh(); } int getLow() {return _low;} int getHigh() {return _high;} </pre>

Separate Query from Modifier: There is a method that returns a value but also changes the state of an object. Create two methods, one for the query and one for the modification.

Table 3.60 Separate Query from Modifier

<pre> GetTotalOutStandingAndSetReadyForSummaries() </pre>
<pre> GetTotalOutStanding() SetReadyForSummaries() </pre>

But like so many optimizations, doing two different things in one loop is less clear than doing them separately. It also causes problems for further refactoring as it introduces temps that get in the way of further refactorings. So while refactoring, don't be afraid to get rid of the loop.

Table 3.61 Split Loop

<pre> void printValues() { double averageAge = 0; double totalSalary = 0; for (int i = 0; i < people.length; i++) { averageAge += people[i].age; totalSalary += people[i].salary; } averageAge = averageAge / people.length; System.out.println(averageAge); System.out.println(totalSalary); } </pre>
<pre> void printValues() { double totalSalary = 0; for (int i = 0; i < people.length; i++) { totalSalary += people[i].salary; } double averageAge = 0; for (int i = 0; i < people.length; i++) { averageAge += people[i].age; } averageAge = averageAge / people.length; System.out.println(averageAge); System.out.println(totalSalary); } </pre>

Split Temporary Variable: There is a temporary variable assigned to more than once, but is not a loop variable nor a collecting temporary variable. Make a separate temporary variable for each assignment.

Table 3.62 Split Temporary Variable

<pre>double temp = 2 * (_height + _width); System.out.println (temp); temp = _height * _width; System.out.println (temp);</pre>
<pre>final double perimeter = 2 * (_height + _width); System.out.println (perimeter); final double area = _height * _width; System.out.println (area);</pre>

Substitute Algorithm: Replace an algorithm with one that is clearer. Replace the body of the method with the new algorithm.

Table 3.63 Substitute Algorithm

<pre>String foundPerson(String[] people){ for (int i = 0; i < people.length; i++) { if (people[i].equals ("Don")){ return "Don"; } if (people[i].equals ("John")){ return "John"; } if (people[i].equals ("Kent")){ return "Kent"; } } return ""; }</pre>
<pre>String foundPerson(String[] people){ List candidates = Arrays.asList(new String[] {"Don", "John", "Kent"}); for (int i=0; i<people.length; i++) if (candidates.contains(people[i])) return people[i]; return ""; }</pre>

3.28 How Often Do You Refactor Your Code?

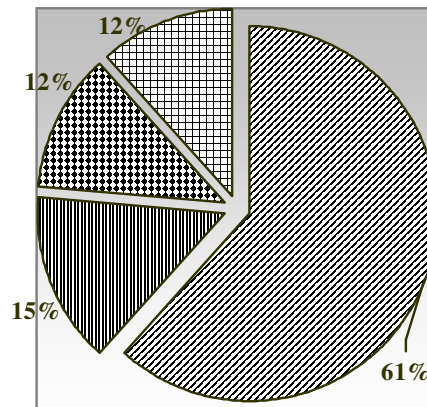


Figure 3.33 Refactor frequency.

Table 3.64 Frequency of refactor code

How often do you refactor your code?	
%61	During project
%15	Before software deliver
%12	After software deliver
%12	If work no refactor

Most of the participants state that they implement the refactoring operation all through the project.

3.29 Do You Add Comments In Your Code?

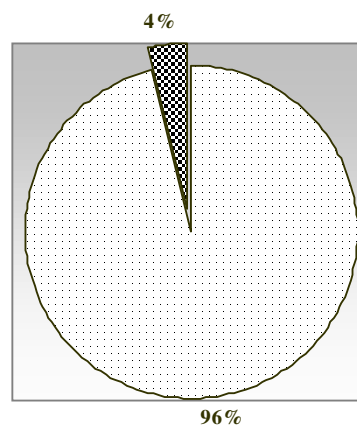


Figure 3.34 Adding code comments

Table 3.65 Adding comments ratio

Do you add comments in your code?	
%96	Yes
%4	No

In XP simplicity is very important. If the program is simple, there is no need to add code comments. It may not always be beneficial to have lots of code comments in a code. It should be added to the parts that are difficult to be understood by the developer.

3.30 Any Function That Is Done A Developer Can Be Written By Another Developer. Is This Situation Happening In Your Company?

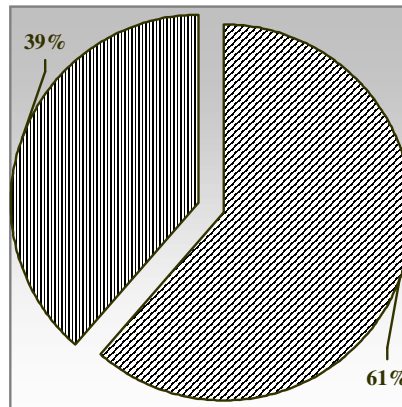


Figure 3.35 Metaphor ratio.

Table 3.66 Metaphor using ratio

Any function that is done a developer can be written by another developer,too. Is this situation happening in your company?	
%61	No
%39	Yes

Metaphor can be used for the objects in the system. It is the appearance of the project.

It is the metaphors that hold how a system operates. It includes information about the usage of classes and objects.

Metaphors decrease the learning process of the system. It provides the system to be learned more quickly.

3.31 What Is Three Factors To Success Software?

If we list the factors standing in the forefront in order:

Analysis: It is important on behalf of learning the demands of the customer.

Team: An adapted team lays the foundation of good software.

Coding Standard: It is important to make changes easily.

The measurement of Project velocity: To deliver the project on time.

Test: To debug the software bugs.

Well modeling: comprehensibility and simplicity.

Resource planning: It is important for customer satisfaction.

Communication: It is necessary to gain customer confidence.

Struggle with stress: More few bugs.

Motivation: The acceptance of work.

Work environment: More quality code, more few bugs.

Experience: To solve problem more easily.

Breaks: To find bugs more easily.

Continuous control: Quality code and delivery on time.

Documentation: The determination of demands and reduction of adaptation time.

Management: The gain of developer confidence.

Security: Customer satisfaction.

3.32 What Are The Reasons Delayed Software?

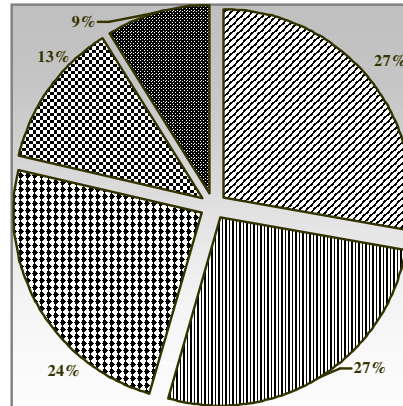


Figure 3.36 Reasons delayed software.

Table 3.67 Reasons delayed software.

What are the reasons delayed software?	
%27	Planning error
%27	Changes
%24	Analyzing error
%13	Development error
%9	Design error

In XP the project velocity can be measured at any moment. Every morning starts with a meeting. In these meetings developers inform about the work that they are going to do. By this way, everyday in which velocity the project is determined and it is understood whether these works will catch up. If it is understood that the works will not catch up, it is tried to be catches up by taking some of the works out.

During iteration planning project velocity is measured. How long every work will last is estimated, in which part of the project it is being worked is examined and it is decided according to this.

After works are done, which works should be done is asked to the customer. After customer's numbering the works, developer completes the relevant work by negotiating the technical details on his own. The measurement of project velocity or

the decision to do the works is an operation that needs experience. This is a situation can only be gained with a good developer's experiences. A friend was developing software under the management of a person who do not understand anything about software. That person asked to the friend in how many days the work could be finished. When our friend said 6 days, he said, "I was considering it to be finished in 2 days but let's take the middle course and make it 4 days." Project velocity is not an operation to be measured with bargaining.

But to decide when the project will healthily be done, can only be after some relations are established. After which technology will be used, which method will be set, what is demanded, what are the things to be done are all determined, it can be decided how long it will take to complete the project.

3.33 Do You Know Extreme Programming Practices?

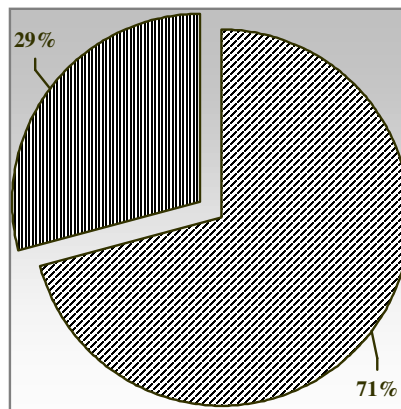


Figure 3.37 Known extreme programming practices.

Table 3.68 Known ratio of extreme programming practices

Do you know extreme programming practices?	
%71	Yes I know.
%29	No

71 % of the participants claim to know these practices. But do not think that they know them in details. The things some managers said about XP are also thought provoking.

“According to me XP can be implemented in compatible projects but I do not think some of its pratiques are very appropriate. Like pair programming and working for 40 hours week”.

“I do not think it can be applied in the big Project.”

“It is ideal for companies developing projects with private objective.”

“According to me some of its practices are beneficial.”

“It is the best methodology that can be used in Turkey.”

“It is a good method with its fast, practical and not including bug points.”

3.34 Do You Apply XP On Your Company?

In this question, I want companies to learn that want to use XP.

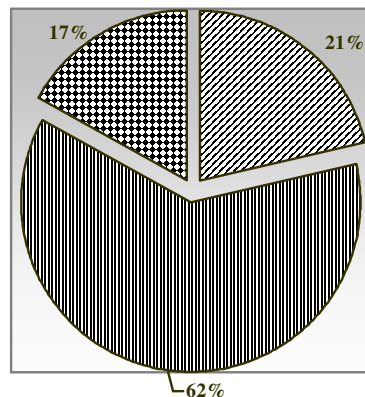


Figure 3.38 Applying XP.

Table 3.69 Table shows applying XP ratio.

Do you apply XP on your company?	
%62	No.
%21	Yes
%17	Partially.

Although most of them know the XP practices, they do not apply them in their companies.

3.35 Do You Know That Apply XP Practices On Any Company?

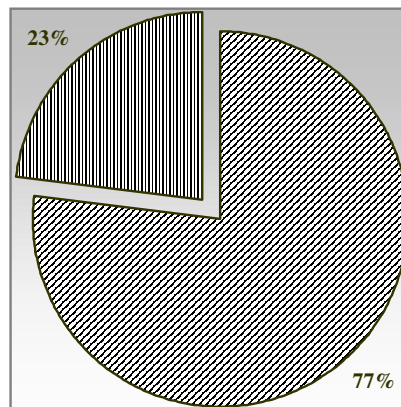


Figure 3.39 Applying XP practices on any company.

Table 3.70 XP practices on any company.

Do you know that apply XP practices on any company?	
%77	No
%23	Yes

Most of them say that they do not know a company applying XP.

CHAPTER FOUR

MANAGEMENT OF XP PROJECTS

There are some main subjects which have to be considered about XP project management. These are explained below in detail.

4.1 Communication

In XP life cycle, customers and programmers work together in order to satisfy the customers' needs. However, sometimes this process becomes uncomfortable for the programmers in many ways, such as loss of concentration or software with bugs.

In order to solve this problem, some activities should be done among software developers under control of managers. Activities like watching a movie or organizing some tournaments may improve communication among workers. By the help of these activities, they may treat more indulgent to each other.

4.2 The Management of Data Integrity and Accuracy

In the complete process of analysis and software development, data integrity and accuracy should be considered in every step. Some fields that are very essential to the customer may be ignored by the programmer. In case of misunderstanding, essential and optional fields should be defined at the beginning of the project.

4.3 Management of Documentation

Generally documentation takes more time than it should and it is really used at end of the project. So, the documentation process should be taken as short as possible. Firstly, it should focus on the customers' needs and it should be simple and easy to understand. So that customers feel more comfortable as they can express their selves.

4.4 Management of Customer Needs

XP projects deal with the customer needs by using user stories which are short and descriptive. At first sight, these user stories look like use cases. However, they differ from use cases in many ways. First of all, user stories are written by the customers and unlike use cases, they do not have a special format. Secondly, they focus on the expectations of the customer. Finally, the main difference between user stories and these kinds of documents is the details of the data given. The user stories should be clear and detailed as they define the customers' needs. By the help of these user stories, the programmers should be able to develop the project easily.

On the other hand, delaying the demands of the customer or making them consider once more by charging additional requests are other important issues about management of the customer needs.

4.5 Management of Errors

In order to correct the errors, testing should be continuous. The project has to be integrated to the existing system after error corrections. This means that, all error correction should be done before the new system is integrated.

4.6 Project Team Management

Team work is one of the important issues in XP projects. Usually various activities are organized in order to improve communication among project team members. Managers and workers mostly finish their work on time -during the work hours- and pay attention not to work at home. Most of the programmers think that working in weekend is not efficient and 40 hours of work in a week is enough.

As it has been experienced up to now, some communication problems occur in a crowded team. For a successful team management, project teams are small about 3-6 people in XP projects. By this method, the quality of the software is aimed to be improved.

Every morning, the day starts with a meeting to define the work to be done during the day. These meetings do not take much time, so they are not waste of time. However in many companies, this method is not very applicable. Mostly, it is hard to bring the team members together every morning, so meeting once a week may be a good solution to this problem.

In some cases, 3-6 people may not be enough as the project run out of time. Forty hours of work is not enough in such a case, so at the beginning of the project -when analyzing the project- the number of workers should be assigned carefully according to the scope of the project.

4.7 Pair Programming

Programmers work in pairs in XP projects which mean two people to every one machine. Pair members rotate regularly and all code is owned by the team, not by individuals. This promotes communication of technical knowledge throughout the team. When technical challenges arise, the team is more able to address the problem.

Pair programming also is excellent in matching up programmers of differing abilities. Less experienced members are constantly mentored, and the risk of less experienced code being added to the application is minimized. On the other hand, pair programming helps programmers to notice bugs and errors in a shorter time period. It provides constant code reviews and two sets of eyes on the code as it's written. Collective ownership of the code by all members of the team helps reviewing the code, increasing the amount of code review performed. So that, one of the programmers usually notices the error while others have not.

However, working in pairs becomes hard to continue sometimes. Programmers may treat intolerant to each other or they may envy their partners. Especially, if one of the programmers is more experienced than the other one, problems increase. For example, experienced programmer may develop most of the code while the other one

is unaware of it. On the other hand, two experienced developers may have problems like not coming up with a common solution. One may insist on his/her solution while the other not.

In addition to all these problems there may be physical problems between them such as a smell of a disturbing perfume or a food had in lunch.

According to the results of the survey, managers also think that pair programming is hard to put in practice because of the time element of the project. It does not make sense working in pairs by using only one computer when there is a little time to complete the project.

Finally, all these issues show us that, pair programming should be used only when it is really necessary.

4.8 YAGNI Management

In XP projects there is not much improvement of code because it is generally waste of time and labor as they are rarely used. User stories are used efficiently in order not to spend more time and money and to define the scope of the project clearly.

However, sometimes this approach may be misunderstood by the developers and the customer may not be well informed or warned about the project development. Because of this, important parts of the code should be developed by the help of the opinions of the customer.

4.9 Refactoring Management

All code should be refactored as often as possible. Refactoring is a process of improving a code's structure without changing its functionality. So it produces highly decoupled objects which makes them easy to test, easy to use, more flexible, and therefore, more changeable.

Refactoring makes the project clearer; therefore it should be done even if the project is well-built.

On the other hand, programmers should take lessons about refactoring or they should be well-informed about this subject. Most of developers still do not know the scope of refactoring concept.

4.10 Code Comments Management

Code comments take a very important place in XP projects. They make the code more apparent. However, it is not useful in some projects that change continuously as inconsistent comments remain in the code. Eventually, they should be used in complicated parts of the project.

4.11 Project Schedule Management

Time is one of the most important elements of an XP project, therefore it is considered frequently. In order to do this, there is a meeting every morning. The aim is to control the programmers and to avoid studying the same subjects on and on. The part of the code developed by a programmer should be known by the others. Thus, some object and relation based metaphors are used.

However, this kind of management is usually difficult for most of the companies. Developers generally get bored of documentation and they think that it should be done by another method, especially automatically.

4.12 Test Management

The faster change is identified, the faster it can be dealt with. XP strives for the fastest feedback possible in every aspect of the project.

In order to satisfy this, there are two types of testing:

Unit Tests are written for most every piece of production code. Unit tests must run at 100% before any code is checked in. When production code needs to be changed, side-effect problems are identified. After new code is checked in, it is immediately integrated with the latest changes from the rest of the team and again, all unit tests must be made without defect.

On the other hand, acceptance tests are written with the customer to verify the application is doing what the customer needs it to do and they are important as they control the system completely.

While testing is very essential for the customer satisfaction and project team success, it is not commonly used in the projects. Most of the managers think that testing is waste of money and they do not arrange a test group in the companies.

4.13 Forty Hours of Work Management

It is believed that forty hours of work in a week is ideal for XP projects. XP emphasizes that; developers should spend time at home with their families and avoid working at home. But it is sometimes impossible. Incorrect scheduling, insufficient planning or lacks of developers make programmers work at home at nights. In order to avoid it, managers should control and suggest not taking the work outside the company. It is not the effective solution for this problem. Besides, working at home may cause different serious problems.

On the other hand, according to the developers working on weekend is not efficient at all. Companies which do not work on weekends are more attractive for everyone. Moreover, weekend holidays refresh the developers and increase the efficiency.

4.14 Modelling Management

CRC (Class Responsibility Collaboration) cards are used in modeling the code in XP projects. Because most of the software companies think that this kind of modeling is waste of time, they are not commonly used. However, successful software should be designed by using these kinds of models. Also, UML and CRC cards make the project clearer. By using these models, programmers can adapt the project in a shorter time. The relation between objects can be easily understood by the help of the models.

4.15 Project Delivery in Time

XP aims to deliver the project in time by using time evaluation. However, the projects are usually delivered late or incomplete. This happens because of the wrong application of XP.

4.16 Database Management

Database management is done by three models named as gold, silver and bronze. Tests are done on silver database and then the project is integrated to the gold database if there is not any error. In some cases, gold database becomes silver, and the silver database becomes bronze in order to update the code.

4.17 Programmer Dependency Management

Generally XP projects aim not to depend on programmers. Also, fair distribution of coding is very important in these projects, but sometimes it is not possible. In fact, pair programming is hard to continue as the projects gain very little. Only necessary part of the project should be completed by this method. For instance, it may be useful when a programmer leaves the project.

4.18 Customer Satisfaction Management

XP projects aim to satisfy the customer by developing the project on-site customer.

Customer satisfaction is the main point of XP; therefore if the customer is not satisfied the project becomes insignificant. Because of this, projects have to be developed under control of the customers.

Customer should be informed about every step of the project. So that, customer can easily notices the needs and warn the developers. However, this approach may cause prejudgments and the customer may feel as the project does not satisfy the needs.

4.19 Technology Management

Both developers and the customer are aware of the improving technology in XP projects. Mostly the requests of the customer are solved by new methods.

4.20 Simple Coding Management

Everything should be done as simple as possible in XP. Refactoring, YAGNI and pair programming ensures this simplicity in a project. Collective code ownership satisfies simplicity as it is mentioned.

4.21 Coding Standards Management

Collective code ownership, pair programming and metaphors help coding the project in defined standards. As every member is responsible of the whole project, there must be some standards. Naming of functions and procedures, structure of the classes and methods should be well-defined and clear as the project starts.

However, it is sometimes hard to apply these standards because of the member circulation of the companies. If there is much circulation, programmers have some difficulties while adapting the project. This is still a serious problem in XP.

As a result, all software companies should make the documentation of these standards and sometimes should control the application of them.

4.22 Continuous Learning Management

Continuous learning is very important as it helps the developers to improve their knowledge of business and relationships with the customers. Different people may have different types of expertises. So, they should be shared among other developers and it only can be by pair programming. Also, collective ownership of the code makes this approach possible.

4.23 Programmer Satisfaction

Although, close relationship of the programmer and customer is the main point of XP, programmers usually do not prefer it. Unlike the customer, programmers are not satisfied.

On-site customer causes bugs and loss of concentration. That is why the call centers are more useful in order to solve customer problems.

4.24 Planning Management

Planning is divided into two parts as release planning and iteration planning. While customer determines the scope of the project in release planning, programmer plans how to do the planned work. Mostly, it is hard to decide which planning type to use. For example, while programmer decides to develop a prior part, the customer

may insist on completing a different subject. As a result, customer should take the responsibility of project planning.

Determining the tasks that are most important to the customer and accomplishing these tasks first looks like a game. In other words, planning game aims to develop a qualified project. The members of this game are programmer, customer and the manager. As the manager generally does not know the whole infrastructure, he should not be very active in the game. This prevents the manager to make mistakes or confuse the customer with inconsistent answers.

4.25 Adaptation Management

Adaptation process takes not much time in XP projects. Everything is designed as simple and clear as possible. Therefore, when a programmer is included in the project, it should not take much time to adapt .However; this approach is useful as it is applied. Generally, software companies aim to complete the project immediately and gain money. So, they do not pay attention to simplicity of the project. Because of this, new developers' struggled as they included in the project and the adaptation takes more time than it should.

4.26 Resource Requirements Management

Requirements of the XP projects are under the responsibility of the managers. Especially, programmer satisfaction is under control of them. Qualified software can only be developed by calm, peaceful and comfortable programmers. It is a fact that, limited conditions, strict rules only makes developers uncomfortable and cause bugs in the project. Having small breaks makes developers more effective and creative.

CHAPTER FIVE

CONCLUSIONS

Today, it seems that software development has become more standardized. In order to develop an efficient product, many software companies continue searching the ways of improving software development.

In this study, Extreme Programming methodology was investigated in detail. Its four values and twelve practices were determined. The differences were listed from other traditional software development methodologies.

A survey was applied to companies of all different sizes and industries. Information was given about survey and details.

Extreme programming is a lightweight software methodology and it has some fewer rules. It is constructed that these rules be applied on small software projects with small team size. Communication and collective code ownership practices could be applied on 3-6 developers efficiently. If the number of developers are greater than 3-6 then the communication path is grow.

Actually XP can be applied on all software projects. The software should be break into small releases so it is more suitable for object-oriented software than others.

Pair programming and refactoring practices are two of the most valuable practices in others. Pair programming practice seems to be the best way to detect error and bugs on software. At first sight the cost of this practice is big but error detection may earn a lot of money to you. On the other hand, refactoring seems to be the best way to simplicity. If the system is very simple then new features could be adding easily and adaptations of developers go down. Simplicity and adding new features also very valuable things on software.

Extreme programming is very important in terms of customer satisfaction. Every step of the process includes customer satisfaction and that is why it seems to become more popular when compared to other traditional methods.

Because everyone is responsible of the software in collective code ownership, programmers are aimed to like their jobs. When it comes to compare XP with other methodologies, it seems to be better as it gives importance to customer and programmer satisfaction.

Also, XP accomplishes with stress successfully when it is compared with other approaches. That is probable because efficient work is aimed instead of endless work.

As documentation takes much time in software development process, programmers do not like it. At this point, XP prevents this trouble by not taking documentation to the front. Also, this approach is may be the most popular feature of the XP among programmers.

Team work is one of the most important practices of XP. While continuous researches go on, all team members are responsible for the whole software project and this creates a project that is independent from individuals (programmers). So, it can be said that this practice is usually the most popular one among others.

As a result, this study can be a reference to others who want to learn about Extreme programming and can be used by the companies as a guide for used.

REFERENCES

- Brewer, J. (2001). Extreme Programming FAQ. Retrieved May 26, 2005, from <http://www.jera.com/techinfo/xpfaq.html>
- Burke,E.M., & Coyner,B.M. (Eds.). (2003). Java™ Extreme Programming Cookbook. O'Reilly & Associates, Inc.
- Constantine, L.L.(Ed.). (2003)., Constantine on Peopleware (Yourdon press computing series), ed. E.Yourdon. 1995, Englewood Cliffs, NJ: Yourdon Press.
- Cunningham, W. (2006). Extreme Programming RoadMap. Retrieved May 26, 2005, from <http://www.c2.com/cgi/wiki?ExtremeProgrammingRoadmap>
- Extreme Programming. (n.d.). Retrieved June 8, 2005, from <http://www.extremeprogramming.org>
- Hally, Mike (2005:79). Electronic brains/Stories from the dawn of the computer age. British Broadcasting Corporation and Granta Books, London. ISBN 1-86-207663-4
- Hightower,R., & Lesiecki, N (Eds.). (2002). *Java tools for extreme programming*. John Wiley & Sons, Inc
- Kessler, R. & Williams,L A.,(Eds.).(2000). The effects of “pair-pressure” and “pair-learning” on software engineering education, University of Utah
- Nosek, J.T. (Ed.). (2001), The case for collaborative programming, p. 105-108. , in communications of the ACM. 1998.
- N. V. Flor & E. L. Hutchins, “Analyzing distributed cognition in software teams: A case study of team programming during perfective software maintenance,” presented at Empirical Studies of Programmers: Fourth Workshop, 1991.

Refactoring (n.d.). Retrieved May 10, 2006, from <http://www.refactoring.com/Senge>, P. The Fifth Discipline: The art & practice of the learning organization. Currency Doubleday, 1990, 4.

Talbott, N.,(Ed). (2001).Conversant pairing. John Wiley & Sons, Inc

Wake,W.C., (Ed.). (2001). Extreme programming explored. XP series, John Wiley & Sons, Inc.

Wesley,A.,(Ed). (2002). Planning XP. John Wiley & Sons, Inc

Wiki programming in pairs in Portland Pattern Repository. (n.d.). Retrieved August 8, 2003, from <http://c2.com/cgi/wiki?ProgrammingInPairs>.

Williams, L. (Ed.). (1999), Pair programming questionnaire . Retrieved May 26, 2005, from <http://limes.cs.utah.edu/questionnaire/questionnaire.htm>.

APPENDICES

XP Anket Formu

TC
DOKUZ EYLÜL ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ
BÖLÜMÜ

XP ANKET YERLEŞİM
FORMU

Hazırlayan: **Ali BAKAN**

Şirketinizin merkez ofisi hangi şehirdedir?
Bir projede birden çok ofis birarada çalışıyor mu?

Konumunuz:

- | | | | | |
|-------------|-----------------------|------------------------|--------------|------------|
| a) Yönetici | b) Sistem
Analisti | c) Proje
Yöneticisi | d) Kodlayıcı | e) Öğrenci |
|-------------|-----------------------|------------------------|--------------|------------|

Şirkette ne tip yazılımlar geliştirilmektedir?

- a) Müşteriye özel yazılım geliştirilmektedir.
- b) Genel satış amaçlı yazılımlar geliştirilmektedir.
- c) Bağlı bulunduğumuz holdinge özel yazılım geliştirilmektedir.

Şirketinize resmi kaynaklı,askeri ve yurtdışı kaynaklı işler alınıyor mu?

- a) Tüm projelerimiz bu şekildedir.
- b) Bu tip bazı projelerimiz var
- c) Bu tip projelerimiz yok.

Şirketinizde yazılımla ilgilenen toplam kaç çalışanınız var?

Şirketinizde bir projeye ortalama kaç kişi atanıyor?

Bir projede müşteri ile görüşme sıklığınız ne kadardır?

- | | | | |
|---|---|---|---|
| a) İhtiyaç
olduğu
durumlarda
görüşme
yapılabilir. | b) Haftada bir iki
kez görüşme
yapılır. | c) Ayda bir
iki kez
görüşme
yapılır. | d) Analiz süresince
görüşülür,raporlanır,proje
bitinceye kadar görüşme
gerekmez. |
|---|---|---|---|

Aşağıdaki maddelerden hangilerini şirketinizde uygulamaktasınız?

- a) Proje bitmeden projenin ara sürümlerinin müşteriye gösterilmesi.
- b) Müşteriden sürekli geri bildirim alınması.
- c) Yazılan kodların günlük olarak birleştirilmesi.
- d) Proje ekibinin konusunda deneyimli personellerden oluşması.

e) Projenin birbirinden bağımsız bileşenlere ayrılması.

f) Şirket içerisinde organizasyon hiyerarşisinin olması.

g) Ekip içi sözlü iletişimin sağlanması.

Görevi yalnızca hazırlanmış yazılımı test eden bir biriminiz var mı?

a) Evet b) Hayır

**Bir projedeki raporlama ihtiyaçları sizin için sorun teşkil ediyor mu?
Cevabınız evet ise bu sorunları çözme adına ne gibi yollar izlersiniz?**

- a) Çalışma sürecini değiştirmek.
- b) Sadece raporlama için özel ekip kurmak.
- c) Danışman firmadan profesyonel destek almak.
- d) Diğer

Şirket işlerini hangi sıklıkta evinize götürmektесiniz?

a) Her gün b) İki-Üç günde bir c) Haftada bir d) Çok nadir e) Götürmem

**Şirketinizde işin yetiştirilmesi için çok fazla mesaiye kalıyor musunuz?
Kalıyorsanız bunların nedenleri sizce nedir?(Tasarım eksikliği, iyi planlayamama vb...)**

Modelleme,UML ve CASE araçlarını projelerinizde kullanıyor musunuz?

a) Sıklıkla kullanırım b) Zaman kaybı olarak görürüm c) Faydalı olduğunu düşünüyorum ancak ben kullanmıyorum.

Şirketinizde çözümleyici(sistem analist) var mı?

a) Evet b) Hayır

Projelerde tasarım yapan ile kodlama yapan kişiler farklı mı?

a) Evet b) Hayır

Kodlayıcıların müşteriler ile görüşmesini nasıl buluyorsunuz?

a) Doğru buluyorum Çünkü ...

b) Doğru bulmuyorum Çünkü...

Proje geliştirme hızının ölçülmesi ve zaman planlamasının yapılması için şirket içi toplantıları hangi sıklıkta yaparsınız?

- a) Her sabah toplantı ile başlar.
- b) İki-üç günde bir toplantı yapılır.
- c) Haftada bir yapılır.
- d) Yapılmasına gerek kalmaz. İlk toplantıda görev ataması yapılır sonra teslim süresi kontrol edilir.

Müşterilerinizin ihtiyaçlarını öğrenme adına neler yaparsınız? Varsa diğer yöntemleri de ekleyiniz.

- a) Toplantılar yapılır.
- b) Raporlar hazırlatılır. Hazırlanan raporlar incelenir.
- c) Yerde analiz yapılır, potansiyel kullanıcılarla görüşülür.
- d) Diğer

Bir projenin müşteriye gösterilmesi ne zaman gerçekleşir?

- a) Proje tamamlanınca gösterilir.
- b) Her hafta sonu/her ay sonu yapılan işler müşteriye gösterilir.

Proje tesliminden sonra bir hata (bug) çıktığında bu hatanın giderilmesi adına uyguladığınız süreç nedir?

Şirketinizde deneyimli bir kişinin “proje süresi içerisinde” işten ayrılması sizleri ne kadar etkilemektedir?

- a) Kişiye bağımlı projeler yapılmadığı için böyle bir durumda sıkıntı yaşanmaz.
- b) Her ne kadar kişiye bağımlı işler yapılsa da kişinin ayrılması iş durumunu olumsuz etkiler.
- c) Kişiler birbirlerini yetiştirdiği için her an yerine bakabilecek çalışan vardır.

Şirketinizde deneyimli bir kişinin “proje tesliminden sonra” işten ayrılması sizleri ne kadar etkilemektedir?

- a) Kişiye bağımlı projeler yapılmadığı için böyle bir durumda sıkıntı yaşanmaz.
- b) Her ne kadar kişiye bağımlı işler yapılsa da kişinin ayrılması iş durumunu olumsuz etkiler.
- c) Kişiler birbirlerini yetiştirdiği için her an yerine bakabilecek çalışan vardır.

Yeni başlayan bir kişinin şirketinizde projelere tam zamanlı olarak katılacak şekilde adapte olması ne kadar sürüyor?

- a) 1-3 ay b) 3-6 ay c) 9-12 ay d) 12 aydan fazla

Sizce bir projede ideal takım kaç kişiden oluşmalıdır? Sayıyı neye göre belirlersiniz?

- a) 1-3 b) 3-6 c) 9-12 d) 12 den fazla

Kriterleriniz:

Bazı kurumlarda çalışanın performansını artırma adına çeşitli kısıtlamalara(Örneğin çay,kahve,sigara molalarına kısıtlama) gidilmektedir. Bu uygulamanın faydasına inanıyor musunuz?

- a) Evet b) Hayır

Haftada kaç saatlik bir çalışmanın ideal olduğunu düşünüyorsunuz?

- a) 30 b) 35 c) 40 d) 45 e) 45 ve üstü

Şirketinizde ekip içi iletişimi ve motivasyonu artırma adına neler yapılmaktadır?

Sizce ekip içi iletişimi ve motivasyonu artırma adına neler yapılabilir?

Kodlama ile ilgili bir standartınız var mı?

- a) Var b) Yok

Yeni başlayan bir çalışanın kodlara müdahale edebilmesi için neler yaparsınız?

- a) Kodlama standartları ile ilgili yazılmış dökümanı incelemesi istenir. b) Belli saatlerde eğitim verilir. c) Kodları incelemesi istenir

Sizin yazdığımız koda başkası kolayca müdahale edebiliyor mu?

- a) Rahatlıkla müdahale edebiliyor. Çünkü şirket içerisinde belli standartlarda kod yazılıyor. Kodu inceleyen kişi rahatlıkla anlayabiliyor.
b) Çoğunlukla anlayabiliyor ancak anlaşılmayan durumlarda bana danışıyor.
c) Temel olarak anlaşılıyor.

Başkasının yazdığınız koda müdahalesi sizi rahatsız eder mi?

- a) Rahatsız eder çünkü o benim yaratıcılığımın bir ürünüdür.
- b) Rahatsız eder çünkü yazılan koddan ben sorumluyum.
- c) Rahatsız etmez.

Kodlamada “Kodun Sahibi” fikri ile ilgili ne düşünüyorsunuz?

- a) Kodların sahibi olmamalıdır. Herkes her koddan sorumlu olmalıdır.
- b) Her yordam ve alt yordamın bir sahibi olmalıdır. Değişiklikler onların bilgisi dahilinde olmalıdır.
- c) Farklı birim tarafından bu tip genel yordam, alt yordamlar yazılmalıdır. Bunların haricinde kimse müdahale edememelidirler.
- d) Diğer ...

İkili kodlama fikrini nasıl buluyorsunuz? (İkili kodlama : Bir bilgisayar başında bir modülün iki yazılımcı tarafından hazırlanması.)

- a) Faydalı bulmuyorum çünkü uygulanmasını zor buluyorum.
- b) Faydalı bulmuyorum çünkü tek kişi ile yazılan kod her zaman için iyidir.
- c) Faydalı bulmuyorum çünkü zaman kaybı olarak görüyorum.
- d) Duymamıştım ancak faydalı görünüyor.
- e) Duymamıştım ancak faydalı görülüyor.
- f) Diğer

YAGNI(You Aren't Going to Need It) düşüncesi hakkında ne düşünüyorsunuz?(YAGNI : Proje geliştirirken müşterinin istekleri doğrultusunda kodlama yapmak gerekir, ihtiyaç duyulmadıkça kaynak harcanmamalıdır.)

Sizce çalışan karmaşık bir koda müdahale etmek gerekir mi?

- a) Evet
- b) Hayır

Yukarıdaki soruya cevabınız Evet ise hangi sıklıkta kodlarınızı gözden geçirip tasarım iyileştirmesi yapıyorsunuz?

- a) Evet proje boyunca tasarım iyileştirmesi yapıyoruz.
- b) Evet proje tesliminden önce tasarım iyileştirmesi yapıyoruz.
- c) Evet proje tesliminden sonra tasarım iyileştirmesi yapıyoruz.
- d) Proje çalışıyorsa tasarım iyileştirmesi yapmıyoruz.

Program içerisine açıklama(comment) satırları ekliyor musunuz?

- a) Hayır, kodlama belli standartlarda yazıldığı için açıklama satırlarına gerek kalmıyor.
- b) Evet,açıklama satırları ekleniyor.

c) Hayır,açıklama satırları eklenmiyor.

Bir işlevi gerçekleştiren bir yordamın başka bir kodlayıcı tarafından birinciden habersiz olarak ikinci kez tekrar yazıldığı durumlar olabiliyor mu?

a) Evet

b) Hayır

Kodlayıcılar bir projede yazılan yordam , alt yordamlardan nasıl haberdar oluyorlar?

a) Yazılan yordam, alt yordamlar veritabanında tutulur.

b) Sözlü olarak diğer kodlayıcılara bu tip yordam ve alt yordamların varlığı sorulur.

c) Bu tip yordamlar belirli kişilerden istenir. O yüzden böyle bir duruma gerek kalmaz.

d) Diğer.

Başarılı projelerinizde sizce bu başarıyı sağlayan 3 önemli neden neydi?

a)

b)

c)

Projelerinizi zamanında teslim edemediğiniz oluyor mu? Nedenleri neler olabilir?

a) Müşterilerin ihtiyaçlarında değişiklik olması.

b) İhtiyaç analizinin eksik yapılması.

c) Tasarımın eksik yapılması.

d) Kodlayıcıların istenilen sürede kodları yetiştirememeleri

e) Zaman planlamasının yanlış yapılmış olması(fazla iyimser/fazla kötümser)

f) Diğer

Ekstrem programlama sürecini duyduunuz mu?

a) Evet

b) Hayır

Cevabınız evet ise yararına inanıyor musunuz?

Şirketinizde uyguluyor musunuz?

a) Evet

b) Hayır

Ekstrem programlamayı uygulayan bir şirketle karşılaştınız mı?