

# **A TOOL FOR SUPPORTING TEAMWORK IN EARLY SOFTWARE DESIGN MEETINGS**

**A Thesis Submitted to the  
Graduate School of Natural and Applied Sciences of  
Dokuz Eylül University**

**In Partial Fulfilment of the Requirements for  
the Degree of Master of Science in Computer Engineering, Computer Program**

**90587**

**by**

**Gamze SARMAŞIK**

**July ,1999**

**İZMİR**

**T.C. YÜKSEKÖĞRETİM KURULU  
DOKÜMENTASYON BİRİMİ**

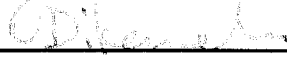
## M.Sc. THESIS EXAMINATION RESULT FORM

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as thesis for the degree of Master of Science.



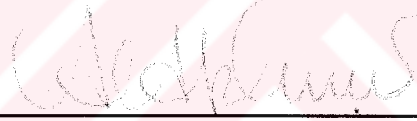
**Assoc. Prof. Dr. Alp KUT**

(Advisor)



**Assist. Prof. Dr. Oğuz DİKENELLİ**

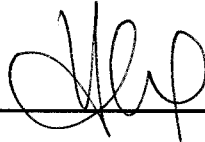
(Committee Member)



**Assist. Prof. Dr. Adil ALPKOÇAK**

(Committee Member)

Approved by the  
Graduate School of Natural and Applied Sciences



**Prof. Dr. Cahit HELVACI**

Director

---

## ACKNOWLEDGMENTS

---

We have always produced profitable studies in a short time within the enjoyable team atmosphere that has been created by my lecturers Elif Demirörs and Onur Demirörs who provided us with the every necessary resources and facilities. I would like to express my gratitude to them for the encouragement and support, their friendly and motivated they gave in order to discover our abilities and force our limits. With their studies and achievements they left on us unforgettable traces which we will follow all through our lives.

I would like to thank my advisor Alp Kut for his help. I also would like to thank my thesis committee members Oğuz Dikenelli and Adil Alpkoçak.

My special thanks go to Ali Yıldız in our team for his help and cooperation throughout this research study.

At least but not least, I thank my family for their patience and support, which enabled my master program to reach to an end.

---

## ABSTRACT

---

Software development is a complex activity that requires a group of individuals working effectively as a team. Studies have shown that the performance of effective teams can exceed that of individuals acting independently. However achieving effectiveness is a challenging task that needs the investment and commitment. In the second chapter of this study, we investigate the characteristics that differentiate effective teams from all others. To observe these characteristics within real software teams, we have chosen Microsoft company as a case study and analyzed Microsoft teams with respect to the characteristics that we define.

In the third chapter of this thesis, we also investigated why some teams fail to perform effectively. We have observed that conflicts among team members' expectations, undefined roles and commitments are the primary factors that reduce effectiveness of teams. Several approaches have been suggested to overcome these difficulties and building effective teams. These approaches, called team design methodologies, generally include an information structure that can be adapted by the teams as well as a set of guidelines on how to apply this information structure.

Based on the activities suggested by different team design methodologies as well as on our observations on software teams we have investigated the various mechanisms that would support effective team functioning. All of the tools as CASE tools, project management tools, workflow management tools provide assistance to different aspects of software development when used appropriately will help a team to define its own information structure and then operate with respect to this structure. However, a major activity of software teams, design meetings, is not addressed by any of these tools. During design meetings team members discuss the requirements, project plan and develop solutions. In order to support building of a team memory and effective communication the decisions made during these sessions should be recorded and data should be passed to appropriate tools within the development cycle. In this thesis, we introduce a software support tool we developed for a team of software program developers so that can enter information from each workshop meeting and develop early project design.

---

## ÖZET

---

Yazılım geliřtirmek, bir grup bireyin etkin bir takım olarak alıřmasını gerektiren karmařık bir aktivitedir. Arařtırmalar, etkin takım performansına bireylerin bağımsız olarak alıřmasıyla ulařılabileceğini göstermiřtir. Bununla beraber, etkinlięi saęlamak, yatırımı ve kendini adamayı gerektiren, emek isteyen bir iřtir. Biz, bu tezin ikinci bölümünde, etkin takımları dięer tüm takımlardan ayıran özellikleri arařtırdık. Bu özellikleri gerek yazılım takımlarında gözlemek için, Microsoft řirketini uygulama alıřması olarak aldık ve Microsoft takımlarını bizim tanımladıęımız özelliklere göre analiz ettik.

Bu alıřmanın üçüncü bölümünde, bazı takımların etkin alıřmalarına raęmen neden başarısız olduklarını arařtırdık. Üyeler arası beklentilerdeki atıřmaların, belirlenmemiř rollerin ve vaatlerin, takımın etkinliğini düşüren birincil etken olduęunu gözlemledik. Bu zorlukları ařmak ve etkin takımlar kurmak için bir kaç yaklařım önerilmektedir. Takım dizayn yöntemleri diye adlandırılan bu yaklařımlar genellikle takımlarca uyarlanabilen bilgiyi ve aynı zamanda bu bilginin nasıl uygulanacaęı konusundaki önerileri içerirler.

Farklı takım dizayn yöntemlerinin önerdikleri aktiviteleri ve yazılım takımları üzerindeki gözlemlerimizi temel alan, etkin takım fonksiyonlarını destekleyecek eřitli mekanizmaları arařtırdık. Yazılım geliřtirmenin farklı alanlarını destekleyen, bütün araçlar CASE araçları, proje yönetim araçları, iř akıřı yönetim araçları yerinde kullanıldığında, takımın kendi bilgi yapısını tanımlamasını ve daha sonra bu yapıya uygun alıřmasını saęlar. Ancak yazılım takımlarının ana aktivitesi olan dizayn toplantıları bu araçların hiçbirini tarafından desteklenmez. Dizayn toplantıları sırasında takım elemanları, gereksinimleri, proje planını tartıřırlar ve özümmler geliřtirirler. Takım belleęinin kurulması ve etkin iletiřimin desteklenmesi için bu ařamalarda alınan kararlar kaydedilmeli ve veriler uygun araçlarla geliřim döngüsüne aktarılmalıdır. Biz bu tezde, yazılım takımlarının, proje toplantılarındaki bilgileri girebilmeleri ve ön dizayn yapabilmeleri için geliřtirdięimiz yazılım destek aracını tanıttık.

---

# CONTENTS

---

	<b>Page</b>
Contents.....	6
List of Tables .....	9
List of Figures .....	10

## Chapter One INTRODUCTION

1.1 Effective Team and Their Characteristics.....	11
1.2 The Importance of Design on Software Development.....	12
1.3 Team Design Methods.....	12
1.4 Early Design Meetings and A Suggested Support Tool .....	13

## Chapter Two EFFECTIVE TEAMS AND THEIR CHARACTERISTICS

2.1 Group Dynamics.....	15
2.1.1 Software Groups as Group Dynamics.....	15
2.1.2 Factors Affecting Group Performance .....	16

	<b>Page</b>
2.2 Effective Team and Their Characteristics.....	22
2.2.1 Size.....	23
2.2.2 Structure.....	23
2.2.3 Composition.....	24
2.2.4 Process.....	24
2.3 Building Effective Teams .....	25
2.4 A Case Study: Microsoft Software Development Teams .....	25
2.4.1 Size.....	26
2.4.2 Structure.....	26
2.4.3 Composition.....	27
2.4.4 Process.....	29
2.4.5 Evaluation of the Microsoft Teams Characteristics .....	30
2.5 The Role of Communication Media on Teamwork.....	32

## Chapter Three

### METHODOLOGIES SUPPORTING TEAMWORK

3.1 The Difficulties of Software Design.....	36
3.2 The Functions of Early Design Meetings in a Teamwork.....	37
3.3 Team Design Methods.....	38
3.3.1 Joint Application Design (JAD) .....	38
3.3.2 Rapid Application Development (RAD) .....	44
3.3.3 Participatory Design (PD) .....	45
3.3.3.1 Comparison between JAD and PD.....	47
3.3.4 Team Design (TD) .....	48
3.3.4.1 Team Design Phases: Planning and Initiating.....	51
3.3.4.2 Team Design Preworkshop Planning and Initiating Phase Methods	53
3.3.4.3 Team Design Formats.....	56

## Chapter Four

### A TOOL FOR EARLY SOFTWARE DESIGN MEETINGS (EDT)

	Page
4.1 Early Design Meetings.....	59
4.2 The Issues of Design .....	60
4.3 Methods in Early Design Meetings.....	61
4.3.1 Brainstorming.....	61
4.3.2 Scoping.....	62
4.3.3 Context Diagram.....	63
4.4 The positive effects of the tool on software design complexity.....	64
4.5 Sample Scenario on Early Design Meetings.....	65
4.6 Early Design Tool (EDT) .....	71
4.7 Suggestions for Further Research on the Tool EDT.....	76
 <b>CONCLUSION</b> .....	77
 <b>REFERENCES</b> .....	79
 <b>APPENDICES</b>	
A. Early Design Tool (EDT) Class Diagram.....	83
B. Installing the EDT.....	87
C. EDT Program Code.....	93



---

## LIST OF TABLES

---

	<b>Page</b>
Table 2.1. Characteristics of Effective Teams.....	24
Table 2.2. Characteristics of Microsoft Teams.....	31
Table 3.1 Model Team Design Format .....	49
Table 3.2 Planning .....	51
Table 3.3 Initiating.....	52
Table 3.4 Team Design Format .....	56
Table 4.1 Team Design Format .....	59
Table 4.2 Requirement Analysis .....	60
Table 4.3 Requirement Gathering .....	61
Table 4.4 Requirement Analysis .....	64

## LIST OF FIGURES

	Page
Figure 2.1 Communication Structures Between Individuals.....	17
Figure 2.2 Decision-making Structures.....	18
Figure 2.3 Communication Structures.....	19
Figure 2.4 Messages sent and received across media types.....	33
Figure 2.5 Duration per contact by media type.....	34
Figure 3.1 JAD Phases.....	39
Figure 3.2 JAD Phase 1.....	40
Figure 3.3 JAD Phase 2.....	41
Figure 3.4 JAD Phase 3.....	42
Figure 3.5 JAD Phase 4.....	43
Figure 3.6 JAD Phase 5.....	44
Figure 3.7 Techniques Applicable in Participatory Design.....	46
Figure 3.8 Sample workshop agenda.....	55
Figure 4.1 Brainstorming.....	61
Figure 4.2 Scoping.....	62
Figure 4.3 Context Diagram.....	63
Figure 4.4 Relationship among the Requirements Gathering Activities.....	64
Figure 4.5 Brainstorming.....	67
Figure 4.6 Data Transfer from Brainstorming to Scoping Phase.....	68
Figure 4.7 Data Transfer from Scoping to Context Diagram.....	69
Figure 4.8 Data Transfer among Brainstorming, Scoping and Context Diagram.....	70

---

## CHAPTER ONE

# INTRODUCTION

---

### 1.1 Effective Teams and Their Characteristics

Developing large-scale software systems requires individuals working together with diverse backgrounds and expertise. The outcome of software projects depends not only on technical aspects, but also on human factors and team dynamics (Curtis, 1990). Successful projects are those where team members work effectively with each other. In order to achieve effective team functioning, an understanding of the factors affecting team performance is needed. Suitable physical and social environments are also required so that individuals can work together for a common goal. Investment in teamwork brings major benefits since teams have the potential to outperform individuals acting alone particularly in complex problem domains such as software development (Katzenbach and Smith, 1993).

Studies have shown that the performance of effective teams can exceed that of individuals acting independently. In the second chapter of this thesis, we investigate the characteristics that differentiate effective teams from all others. We first define what it means for a team to be effective, and provide a set of characteristics. Then we concentrate on software teams, and summarise the results of several investigations in software field. Finally, we analyse Microsoft with respect to how effective its teams are working.

## 1.2 The Importance of Design on Software Development

Working as a team means sharing the information. Team members need to discuss the project, write down these ideas and keep up with each other's improvements. Through an effective information exchange each member's individual mental model of the product will be unified with others forming a collective team memory. In order to have an effective information exchange, the "information structure" that defines the team structure, its communication mechanisms, and its software development processes should be explicitly defined.

We have observed that conflicts among team members' expectations, undefined roles and commitments are the primary factors that reduce effectiveness of teams. A good design also prevents the misunderstanding among team members. Several approaches which are called team design methods have been suggested to overcome these difficulties and building effective teams.

## 1.3 Team Design Methods

Team design methods manage team activities in detail such as software design and group planning which includes the information structure. These methods help to organise team workshops, processes and offer a comprehensive set of methods from which teams can choose depending on their needs. Existing models are Joint Application Design (JAD), Rapid Application Development (RAD), Participatory Design (PD) and Team Design (TD).

**Joint Application Design (JAD):** JAD is the most common design process in the North American business culture since 1977. JAD approach presents the basic plan and development issues to entire team in a project.

**Rapid Application Development (RAD):** When a project requires a truly rapid design and development, you may find it useful to plan a single extended JAD workshop which is called RAD. For a smaller RAD project, one workshop of two or three days may cover the needs for all phases of the project.

**Participatory Design (PD):** PD, is a model for user context which is easily understood by organisations. The responsibility for the product is shared by the users and designers. It has been used in Europe for over a decade, only more recently been used in North America.

**Team Design (TD):** Team Design gives organisations a framework from which to build their own custom design practices. TD includes JAD and PD component.

#### **1.4 Early Design Meetings and A Suggested Support Tool**

Early Design Meetings are very delicate process as the team starts to form, members get to know each other, discuss the project plan and requirements, suggest alternative design and develop solutions. Based on the activities suggested by different team design methodologies as well as on our observations on software teams we have investigated the various mechanisms that would support effective team functioning.

In order to support building of a team memory and effective communication the decisions made during Early Design Meeting sessions should be recorded and data should be passed to appropriate tools within the development cycle. In the fourth chapter, we analyse the requirements of a support tool for software design meetings and provide an architecture for such a tool. Then we introduce the tool we developed for supporting teamwork in Early Software Design Meetings.

---

## CHAPTER TWO

# THE ROLE OF TEAMWORK IN SOFTWARE DEVELOPMENT

---

Large scale software development is a complex problem-solving activity that requires coordination and cooperation of several individuals with diverse backgrounds. Success of a project depends not only on technical aspects but also on how effective these individuals function as a team. Effective teams are observed to outperform individual members and accomplish their goals to the satisfaction of all involved. However, forming effective teams and providing the necessary conditions for their functioning is a difficult task that needs to be investigated apart from the technical aspects of software development (Demirörs, 1995).

Investigation of teams and the factors affecting their performance has been a major research area in social sciences for more than three decades (Cartwright & Zander 1960, Ross 1989, Shaw 1976). These studies, classified as "group dynamics", have provided major feedback for teams and for organisations within which teams function. A common observation in group dynamics is that there is no single factor that determines the performance of teams (Hackman, 1990). Performance depends on a complex set of factors that are interdependent and need to be considered together.

In this chapter, we identify the factors influencing the performance of software teams. With this goal as a starting point, we investigated the results of group dynamics research as well as studies conducted on software teams. Our investigation uncovered a set of characteristics, which differentiate effective teams from others. We then performed a study on Microsoft Company and analysed its software development teams with respect to these characteristics.

## 2.1 Group Dynamics

Human behaviour plays a significant role in teamwork thus becomes important in software development. Especially today, there are only a few software engineers who work alone because software systems are too complex to be built alone. Most of the programmers work in teams. Therefore the productivity is related to the effectiveness of the team environment. That is why building effective, harmonious and well balanced teams is the target of today's software organisations.

Most managers are aware that they have more people related problems than technical problems. If they have got little management experience and no meaningful practice they spend most of their time thinking technology and no time thinking about the people side of the problem. This is because the human side of the work is more difficult (DeMarco & Lister, 1987).

As the first and the most crucial condition for successful software development by a team is peopleware, we have investigated group dynamics and the factors that affect the performance of a group.

### 2.1.1 Software Groups as Group Dynamics

In group dynamics, groups are classified depending on their goals such as problem solving groups, interactive groups, educative groups and self evaluating groups.

Groups that have the goal of developing software systems are classified as "problem solving groups" which are generally unnatural since their members are chosen by someone from outside such as a company manager. "Problem solving groups" come together to solve a specific problem. In software world, the problem they are expected to solve is to define and implement software components that will satisfy user requirements.

Bales and Strodtbeck (1950) list the development steps of problem solving groups as follows:

1. Orientation step: Group members are oriented towards the problem domain. This can also be called exploratory step.
2. Evaluation step: Group members try to evaluate the whole problem. They collect information and find alternative solutions.
3. Control step: In the last step group members organise environmental conditions related to the problem and distribute the responsibilities among themselves. They realise their missions.

### **2.1.2 Factors Affecting Group Performance**

There are a number of factors that affect group productivity and quality of group work. Among those factors we can list size, structure, interpersonal skills, leadership, consensus, and motivation. Collaboration and coordination improve group success. It is not easy to establish a team with the optimum blend of these factors and achieve high performance because these are all multi dimensional properties and relationships among members are not stable. We hope the factors we mention here draw the attention of organisations and guide them to achieve high performance and improve software development.

#### **Number of members:**

The group size is of primary importance to team success. There have been a lot of research on the size a group should be (Kerr 1986, Latane 1981, Harkins & Szymanski 1987, Katzenbach & Smith 1993). Teams have been studied in two categories as big or small according to the number of members and studies also tried to determine how group size affects team performance. The results showed that small groups are more productive than big groups. The reasons are as follows:

- The communication among individuals is better because they can easily get together.
- Different viewpoints are less likely to cause differentiation and individuals are less likely to fall of.



On the other hand, big groups have following disadvantages:

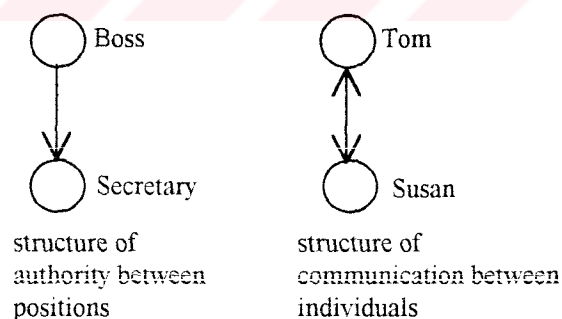
- Coordination of the regular attendance of all members to the meetings and activities is difficult.
- Individuals get less satisfaction from group membership.
- Members are less similar to each other and show less attention to other members' needs and requirements, which leads to lack of cooperation.
- The group breaks into smaller groups.

As a result productivity decreases because of these motivation problems in big groups.

### Team Structure:

Team structure reflects how individual members of a team can be located in relation to other members according to some criterion of placement (Cartwright & Zander, 1960). The relative positions of group members are important in understanding the group dynamics.

Different criteria can be used to describe team structure such as flow of information (communication), flow of work, and flow of authority. These criteria provide relationships between individuals or positions within a team. Examples are given in Figure 2.1.



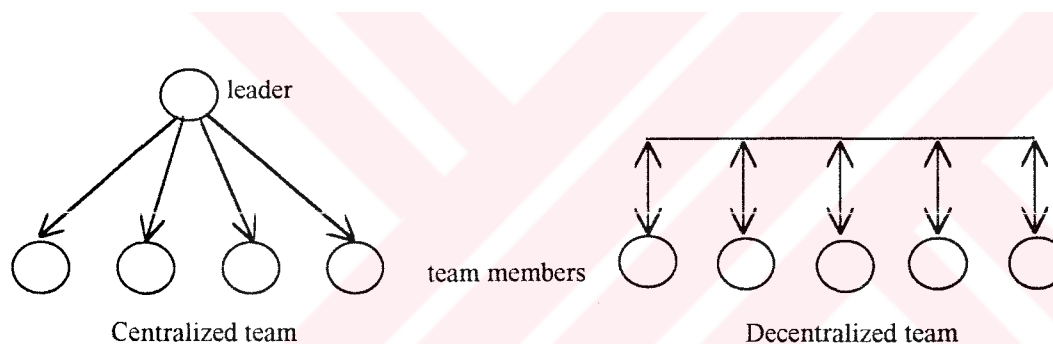
**Figure 2.1 Communication Structures between Individuals**

A structure can be imposed on a team by the organisation the team is working within, or it can evolve during the joint teamwork. In either case, it can be expressed formally in terms of written rules or it can be informal. Whether formally defined or not, team members are aware of the structure and they feel a need to conform.

In this section, we focus on two criteria that define team structure, flow of decision-making, and communication patterns between the team members (Demirörs, 1995, p:31).

**Decision-making structure:** Teams can be classified into two groups with respect to the decision-making structure they adopt or are imposed on :

- Centralised team: decision-making is performed by a single individual who is also responsible from the overall performance and the outcome.
- Decentralised team: decision-making is distributed among members. A consensus-based approach is used for teamwork. Performance and product become the joint responsibility of the team. Figure 2.2 depicts these two types of teams.

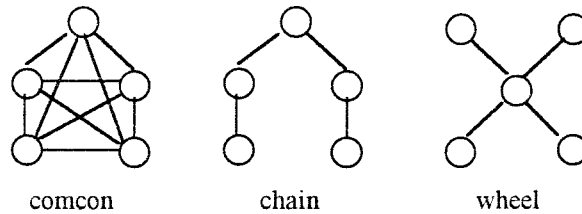


**Figure 2.2 Decision-making Structures**

If the problem is clear and the roles of the members are defined, centralised team structure can be appropriate for such a team. On the other hand, if the problem domain is complicated, goals and approaches are not clear, decentralised team structure can be ideal.

**Communication structure:** Communication structure of a team reflects the physical arrangement of communication channels among members such as who can communicate with whom, whether the communication is direct or via another member. This structure may or may not be the same with the decision-making structure discussed above.

Different patterns of communication can be adapted or imposed on the team some of which are shown in Figure 2.3 (Shaw, 1976).



**Figure 2.3 Communication structures (lines represent communication channels)**

Relative effectiveness of a particular communication structure depends on complexity of the task, goals of the team, and the environment (Ross, 1989). Shaw (1976) points out that decentralised communication network (such as common) is most efficient when the task is complex. He also mentions that in decentralised structures the morale is higher than centralised structures.

#### **Skills of the members:**

When building a team, the individuals' skills should be a key criterion in their involvement to the program. This is especially true if you want the group to reach to the goal in a fast and effective way.

The majority of research related to individual skills showed that highly skilled people had higher performance (Tziner & Eden, 1985). Shaw (1976) also observed that members who have special skills relative to the team task are usually more active, contribute more to the team work, and have more influence on team decision. For example, Curtis (1990) observed that having a member with domain knowledge within the team contributed to the success considerably. Hiring better people is the best way to improve the productivity and quality of systems in an organisation.

## Consensus:

Consensus among the members of a team is a crucial factor affecting the group performance. Although some studies (Patterson, 1986) claim that group composition (members having the same occupation) is not a dominant factor during the decision making process, effective groups are generally composed of members who resemble each other because people that have similar characteristics find each other more attractive and closer (Feld, 1982). When group members have similar ideas, they easily reach a consensus on a subject and get on well each other (Stasser & his friends 1989). Consensus in group dynamics is studied with respect to three dimensions as cultural, social and sex. Cultural, social and sex differences affect relationships of the group members in the following ways:

**Culture:** Cultural differences among individuals determine their relationships. For example, people immigrated from villages to cities or people living abroad may have different viewpoints.

**Social status:** Status means income, education, occupation etc. The more the status of people in a society resembles each other, the more they get on well each other.

**Sex:** Sex is also another factor determining the relationships among members but this type of evaluation is still uncertain and complex so it could be discarded. According to the investigations, men are interested in power, status and income whereas women are more sincere, cooperative, supporting and natural (Porter & his friends 1985, Patterson & Schaffer 1977).

## Leadership:

Leadership function is the universal dimension of groups. It is the leader who organises, directs and motivates a group. The basic responsibilities of a leader are:

- Determining the aims of the group and keeping the team morale high.
- Establishing the coordination, dividing tasks, planing the mission and creating harmony among members (Aktaş, 1997, p74).

Leader's role changes according to the structure of the team. In a centralised team the leader plays a very important role in group performance. In a decentralised team, however, leadership responsibilities are shared by team members. In order to be successful and exist for a long time all the members should have a leaderlike approach. Besides, communication structure should be democratic and all members should be encouraged to participate in teamwork. As a conclusion, teams where responsibilities are shared not only by a leader but also by other members are more successful and productive.

### **Goals and Approaches:**

The goal of a group can be defined as the final result that the members aim to reach. A group may have only one goal or more. The heart of the matter is the clear and complete definition of the goals. This simplifies the cooperation and coordination among members and improves the feelings of satisfaction by combining the members of the group.

In order to have an effective group work, members should commit themselves to a common purpose and to a common approach that explains what the team needs to do to achieve its goals. The common purpose is formed by the team as a collaborative effort that ensures the awareness of each member. Common approach describes how the team is going to work together towards a common purpose. A team defines its own approach considering both administrative and social aspects of its work.

Deciding together on how, when and by whom the activities related to the goals and approaches will be given is important because this kind of interaction among members increases the group performance (Aktaş, 1997). Besides, when the intention for success increases among group members, they have a tendency to choose a task with medium difficulty, on the other hand, groups aim extremely hard goals when they have a risk to fail (Trape, 1975).

## **Motivation and Cooperation:**

It is striking that motivation of the group members directly affects team performance. When we say motivation, we mean the mutual consensus and cooperation of the members, the willingness to produce task together, the existence of a team spirit. Team members should act as “One for all, all for one”.

In order to improve individual behaviour, regular meetings are organised. Members outline their goals and approaches during those meetings. In addition, social meetings like parties, dinners help the members get closer. Team members feel the need to conform to the standards of the team in order to coordinate and communicate with each other. After some time, members learn about their strong and weak points and compensate each other when necessary. This brings commitment to each other and work.

Cooperation gets easier when the group members believe that group resources like status, reward are divided fairly. When the cooperation level of a group is high this protects the group against the outside interference and makes the group more powerful. However the presence of cooperation does not mean that disagreement never occurs in a group. Disagreements and differences within a group are an inevitable and dynamic part of it. These conflicts when solved positively ensure the survival of the team.

## **2.2 Effective Teams and Their Characteristics**

We define teams that outperform individual members and accomplish their goals to the satisfaction of all involved as effective teams. In order to achieve high productivity and quality in complex problem domains such as software, teams should be functioning effectively (Curtis, 1990).

Effective teams possess a number of characteristics that distinguish them and increase their performance levels above the teams without those properties. These characteristics can be investigated under four headings: size, structure, composition, and process. Table 1 summarises the characteristics that are identified as essential for effective functioning (Cartwright & Zander 1960, Demirörs 1995, Hackman, 1990, Katzenbach & Smith 1993, Shaw 1976).

### 2.2.1 Size

Size of a team has been observed to have an impact on the performance (Shaw, 1976). In general small teams are believed to function better since coordination becomes a problem as the team gets larger. As a result, small teams are more effective and it is believed that team size should be kept under certain limits, usually between 2 to 12 people (Katzenbach & Smith, 1993). “Workshops with 12 to 15 people are generally manageable, but with more than 20 participants the venue becomes more appropriate for facilitated meeting rather than a productive design session” (Jones, 1998, p.160).

### 2.2.2 Structure

Team structure reflects how individual members of a team can be located in relation to other members according to some criterion such as flow of information (communication structure), and flow of authority (decision making structure) (Cartwright & Zander, 1960). Relative effectiveness of a particular structure depends on complexity of the task, goals of the team and the environment. For example, centralised decision-making structure where decision-making is performed by a single individual works well when that individual is an expert in the problem domain. Observation on effective teams have shown that in general leadership is shared among members, and there is no single individual who is responsible for everything (Katzenbach & Smith, 1993). Even if a leader exists, the decision-making within the team is democratic, and every member has a chance to raise his/her voice. On the other hand, the ease and efficiency of communication structure has shown to be a critical factor on the effective functioning of the team (Ross 1989, Shaw 1976). A communication structure where every member can communicate directly with the other is the one generally adapted by effective teams.



### 2.2.3 Composition

Team composition reflects the behavioral factors affecting the performance of individual team members such as skills, experience, background, and problem-solving strategy. It has been observed that members who have special skills relative to the team task are usually more active, contribute more to the teamwork, and have more influence on team decisions (Shaw, 1976). As a result, teams that consist of such skilled members have higher performance records (Tziner & Eden, 1985). Katzenbach and Smith (1993) argue that in addition to technical and functional expertise, teams also require problem solving, decision making, and interpersonal skills for effective functioning. Members of effective teams have been observed to be highly motivated and committed both to their work and to the team. “One for all, all for one” feeling exists among team members. As a result, members enjoy working together and doing their job (Katzenbach & Smith, 1993).

### 2.2.4 Process

Team process reflects the way team task is accomplished. It includes team approach, set of activities to be performed, and evaluation criteria for success. Effective teams form a set of common goals that define what the team is going to achieve depending on the environment and on the team task. Such teams also define a common approach to attack these goals. Definition of the process by the team members encourages collaboration, increase adaptability within the team, and ensures awareness of each member (Ellis, Gibbs & Rein, 1991). Effective teams all work towards achieving high quality and meeting their customer’s needs.

**Table 2.1. Characteristics of Effective Teams**

SIZE	Small
STRUCTURE	Democratic decision-making Direct communication among all members
COMPOSITION	Highly skilled Highly motivated Committed to work and team Enjoyment
PROCESS	Common goals and approach High quality



### 2.3 Building Effective Teams

Different approaches have been suggested for building effective teams ranging from selecting the best people to retreating to a mountain resort with all members and their families (DeMarco & Lister, 1987). Katzenbach and Smith (1993) provide a list of practices that may lead to an effective team. The main idea behind their suggestions is that performance should be the key focus for effective team formation. These practices are:

- Select the team members based on skills and skill potential, not personalities.
- Pay particular attention to first meetings and actions.
- Set some clear rules of behavior.
- Set and seize upon a few immediate performance-oriented tasks and goals.
- Challenge the group regularly with fresh facts and information.
- Spend lots of time together, especially at the beginning.
- Exploit the power of positive feedback, recognition, and reward.

Although different practices can be followed to build effective teams, generally a considerable amount of time is needed to actually build one. Also it should be noted out that, even if you follow all the suggestions on creating the preferable conditions, and facilitating teamwork, achieving effectiveness depends on the individuals forming the team. Managers and organizations, after establishing an environment to work in, need to let the teams find their own style. There is always a chance of a team failing to work and break down before accomplishing its goal.

### 2.4 A Case Study: Microsoft Software Development Teams

Nowadays the name “Microsoft” is widely spoken among people. Nearly everyone who uses a computer has a Microsoft product. Even those who don’t have a computer or anything to do with computers heard a lot about Microsoft especially its business success. In this section, we analyze whether the software development teams within Microsoft Company have the effective team characteristics given in the previous section (Demirörs E., Demirörs O., & Sarmaşık, 1997).

### 2.4.1. Size

Although Microsoft is a big company with nearly 20000 employees, it still keeps its structure that consists of small teams. Microsoft strategy is “to organize small teams of overlapping functional specialists”. The evidence of Microsoft’s small team structure can be found in the following paragraph:

“Within the product units, program managers, developers, and testers work side by side in small ‘feature teams.’ These typically consist of one program manager (who generally works on more than one feature) and three to eight developers...”(Cusumano & Selby, 1995, p.74).

### 2.4.2 Structure

Microsoft culture has evolved around the leadership of Bill Gates as observed by Yourdon:

“My sense, in my contacts with Microsoft developers, is that this is the quintessential heroic programmer culture, with Bill as the obvious spiritual leader” (Yourdon, 1996, p.271).

The leader works with a group of managers who are senior technical people and many of them are veterans working for Microsoft for long years. They know the technology and how to make money with this knowledge. This top management team runs key product areas and new initiatives, and gives its decisions independently:

*“...the centurions go off on their own and report back only occasionally. But they roam within certain limits, and the leader can rest assured that these centurions-and their troops-are fighting for the good of the whole organization...Microsoft clearly has a leader, a top management team, and an army of employees who deeply understand both the technology and the business of PC software. They also know how to win.”* (Cusumano & Selby, 1995,p.22)

Apart from this top level management, the company has, in general, two more levels of management for software development. Each product unit has a manager, and within product units there exist small teams leaded by program managers. However this hierarchical structure of the company does not reflect the flow of authority. For example, program managers have little formal authority over the groups that will get things done. The following sentence defines the role of program managers within teams clearly:

“The program manager is a leader; facilitator; and coordinator; but is not the boss.”  
(Cusumano & Selby, 1995, p.77)

Also in order to get everyone involved and foster creativity, software teams members are encouraged to raise their voice and state their ideas frankly whether positive or negative. This practice is called “push-back” and empowers the employees to change things within projects.

The communication structure in Microsoft is decentralized, and fully connected since nearly everyone can communicate with each other. In general employees feel that the management practices an open door policy and their ideas are seriously considered (Cusumano & Selby, 1995). Also in order to increase communication among team members nearly all new product development is done on one site. As Gates states “Our all being with very minor exception, here on one site, so that whatever interdependencies exist you can go see that person face to face...” (Cusumano & Selby, 1995, p.26).

#### **2.4.3. Composition**

One of the basic strategies of Microsoft Company is to hire smart people with qualities such as ambition, high IQ, expertise on a technical subject and business knowledge. Since the company gets 10000 unsolicited resumes a month it has a chance to pick the best. (Yourdon, 1996). In total, Microsoft hires between 2 and 3 percent of the people it interviews. Their famous question in their interviews for new hires is: “volume of water following down the Mississippi River and the number of gas stations in the United States” (Cusumano & Selby, 1995). The applicant’s approach for analyzing the problem is important, not the answer. We can also see the skill of Microsoft team members in the explanation of Steve Ballmer:

*“We are lucky to be as successful as we have been; the great people we have working for us has been the reason we’ve achieved this success. As we continue to develop new products, research new technologies and make our existing products faster, smarter and more user friendly, we need to hire a lot more developers. We’re always thinking about new ways to attract the best in the industry to us. The flyer we mailed to you is just one of the things we’re doing to let people in the industry know that we are looking for top-notch technical people.”*  
(Yourdon, 1996, p.266)

Software developers in Microsoft are generally workaholics, who follow traditional Microsoft way: “Wake up, go to work, do some work. ‘Oh, I’m hungry.’ Go down and eat some breakfast. Do some work. ‘Oh, I’m hungry.’ Eat some lunch. Work until you drop. Drive home. Sleep.” (Cusumano & Selby, 1995, p.93) The developers who remain in the company are ambitious, and are willing to work long hours for long-term financial benefits. The work environment provides a continual series of challenges motivating these people to do their best.

Development team members in general feel good about working as a team. They believe that everyone does his/her share and also help each other during peak workload periods (Cusumano & Selby, 1995). However, the commitment of software developers to their teams in Microsoft can be best exemplified with their “mentor” program for new employees. Each new comer is assigned to an experienced developer within the project team, the “mentor”, who is responsible for monitoring everything new employee does including reading every line of code he/she writes. The relationship between the mentor and the new comer creates an atmosphere of friendship and sincerity, they achieve personnel commitment to each other.

The most important practice of Microsoft managers that ensures commitment of team members is to transfer the responsibility of scheduling and work management to individual developers and testers. In other words, software developers make their own schedule given the fixed delivery deadlines, and resources. As Cusumano and Selby points out “This ensures that everyone takes individual responsibility in addition to acting as a part of the team.” (Cusumano & Selby, 1995, p.251) Estimating their own schedules also creates a sense of ownership among the developers and generates a pressure to stay on track since those are their own dates.

Although they work long hours under stress, software developers in Microsoft enjoy their work, which they consider interesting, and varied. Most of them believe that Microsoft is one of the best places to work in the high-tech industry (Cusumano & Selby, 1995).

#### 2.4.4. Process

The software development teams in Microsoft are responsible for defining the specifications of the product, setting the schedules and managing the work. This practice when combined with the “push-back” principle helps the team members to form a set of common goals and approaches. The following story exemplifies how a development team managed to change its focus and found a set of common goals and approaches.

*“ ... Denis polled all the managers and the various leaders in the group, asking each to rank our projects. It became clear that everyone thought a single project, code-named Caviar, was the single most critical to our success. We concluded that if we could get Caviar out in good shape at the right moment, we would live to fight another day. The second most important project was code - named Barracuda, essentially Caviar ported to Windows NT.*

*All the rest of our projects, we decided, would have to go, and go now. We would be lucky to achieve these two. And, if need be, we would simply focus on one, Caviar. We would put everything we had behind shipping Caviar and winning in the market...*

*Caviar would become Visual C++ 1.0 for Windows 3.1.” (McCarthy, 1995, p.15-17)*

Microsoft became more serious about quality control in mid-1980s. Since then quality has become an issue of rising importance to Gates and other managers primarily because of pressure from customers and competitors. Microsoft people still tend to view quality differently. For example, Dave Moore explains quality within Microsoft as follows:

*“Our viewpoint is that an obsession with quality has to map to customers. There has to be utility. There is a certain amount of quality in being flexible, being adaptable to changing customer needs, changing market conditions. And so your standards for quality must not hinder that.” (Cusumano & Selby, 1995, p.325)*

Microsoft as a company has installed a number of practices to achieve higher quality. These are:

1. Early customer involvement: Microsoft collects and utilizes customer feedback before marketing a product. In fact customer feedback is an integral part of the development process in the form of detailed weekly reports on customer inquiries. Bill Gates states:

*“Most people don’t get millions of people giving them feedback about their software products...We have this whole group of over [two] thousand people in the US alone that takes*

phone calls about our products and logs everything that's done. So we have a better feedback loop, including the market." (Cusumano & Selby, 1995, p.26)

2. Postmortem reports: Microsoft began documenting their experiences in projects through written postmortems, reflecting the belief that people could do a much better job at learning from mistakes. Bill Gates, explains postmortem's importance like this:

"We do good postmortems after the projects and look at what we were the source of bugs, how could the design generate less bugs, [and] how could the tools generate less bugs?" (Cusumano & Selby, 1995, p.26)

3. Daily builds: In order to keep several different parts of the same project synchronized Microsoft practices "daily build" approach. The main idea is to create a working copy of the product frequently and perform a number of automated tests on this version. Daily build approach provides rapid feedback to the project team, keeps them coordinated and aware of the progress of the product.

The first Microsoft product that use these quality practices was Excel 3.0 that was shipped only eleven days late (Cusumano & Selby, 1995).

The concept of quality also includes the ability to steer the customer that can be possible through continuously creating new products. In order to encourage new ideas, Microsoft depends on its "push-back principle". It is believed that when software developers are provided a chance to raise their voice, positive or negative, the chances of finding out something new increases.

#### **2.4.5 Evaluation of the Microsoft Teams Characteristics**

As a starting point in this study we investigated various resources analysing Microsoft teams and their practices. We have identified that nearly all of the characteristics of effective teams do exist in Microsoft software development teams (See Table 2.2). The only exceptions are "democratic decision-making" and "high quality".



Although decision-making in Microsoft seems to be democratic with every software developer has a chance to raise his/her voice, top managers can always make the final decision sometimes contrary to team's common wisdom. In a company where strong leadership of a single person exists democracy may not work all the time.

There are different arguments on the quality of Microsoft products in the market. The general belief is that Microsoft produces low quality products. However, observations of various researchers point out that the company has a set of quality related practices although different than those of the field's experts. But the fact that Microsoft products have sold millions of copies around the world may be taken as a proof that its quality definition, "good enough", works for the customers the company targets.

It should be noted that our investigation of Microsoft teams depends on other researchers' work, and their observations. These observations involve not a single team but several teams that the authors had a chance to interview. So it is actually not very clear whether a single team within Microsoft incorporates all these characteristics. However it is clear from our study that within Microsoft there are a number of very good practices that help people to form effective teams and there are a number of high performance teams.

**Table 2.2. Characteristics of Microsoft Teams**

(+ : valid; - : not valid; + - : can't decide)

EFFECTIVE TEAMS IN GENERAL		MICROSOFT TEAMS
SIZE	Small	+
STRUCTURE	Democratic decision-making	+ -
	Direct communication among all members	+
COMPOSITION	Highly skilled	+
	Highly motivated	+
	Committed to work and team	+
	Enjoyment	+
PROCESS	Common goals and approach	+
	High quality	+ -

## 2. 5 The Role of Communication Media on Teamwork

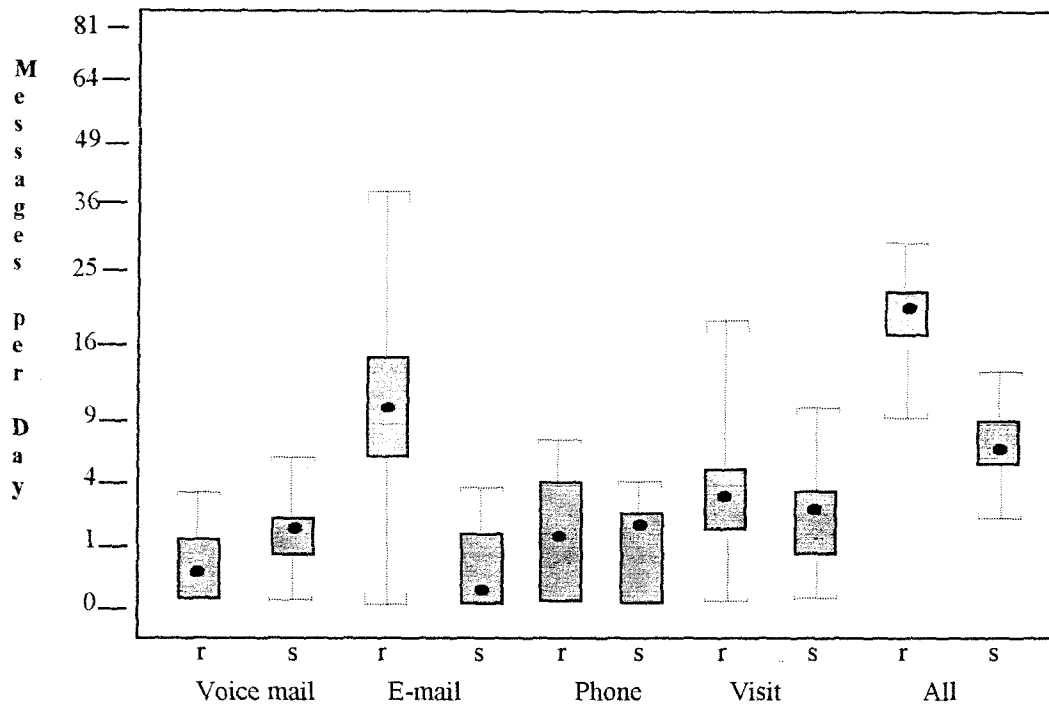
In the beginning of this chapter we mentioned how good communication among team members increases productivity. Therefore, communication tools play a very significant role for a team effort to succeed, because it is essential that the programmers exchange reference materials and advice as well as be able to request modifications and explain the reasons for changes. Most commonly used tools are telephone, e-mail, and fax and voice mail. In this context, communication tools that are preferred and frequently used by members, besides the affects of their usage on teamwork are being explained in this section.

The studies on the use of communication media by development teams are quite few. Perry and Votta (1994) investigated how often team members use communication media and studied the daily contacts of seven people for five days. They included e-mail, voice mail, telephone and visits into their research and the results are noteworthy. Although e-mail was thought to be the most favoured one, others were more frequently used and the most preferred one was visits. Detailed information is given in Figure 2.4. Each box represents a different data set. The heights of the boxes show the spread of the central 50 percent of the data. Data median is shown by the dot in the boxes. The tails of the distribution are denoted by the vertical dashed lines.

Figure 2.4 shows the number of messages being sent and received each day across different media. The distribution of sent and received phone messages and visits are normal. However, the results of e-mail usage are astonishing. The numbers of received e-mail messages are quite high but sent e-mail messages are low and almost none. Though the number of received e-mail messages are high, the contents are rarely technical subjects but mostly about organizational news, announcements, recent product sales and congratulatory messages. We can give several reasons for this.

- It is difficult and time consuming to prepare an e-mail message. People prefer either visiting or calling the person. It is easier and quicker.
- E-mail is used like broadcasting.
- E-mail is not suitable with the iterative problem solving of software technology. Because, people prefer completely finished messages to send not half ones.
- Software CASE tools do not include e-mail and therefore user leaves the tool he is in and enters into another tool. This decreases e-mail usage as it is time consuming.





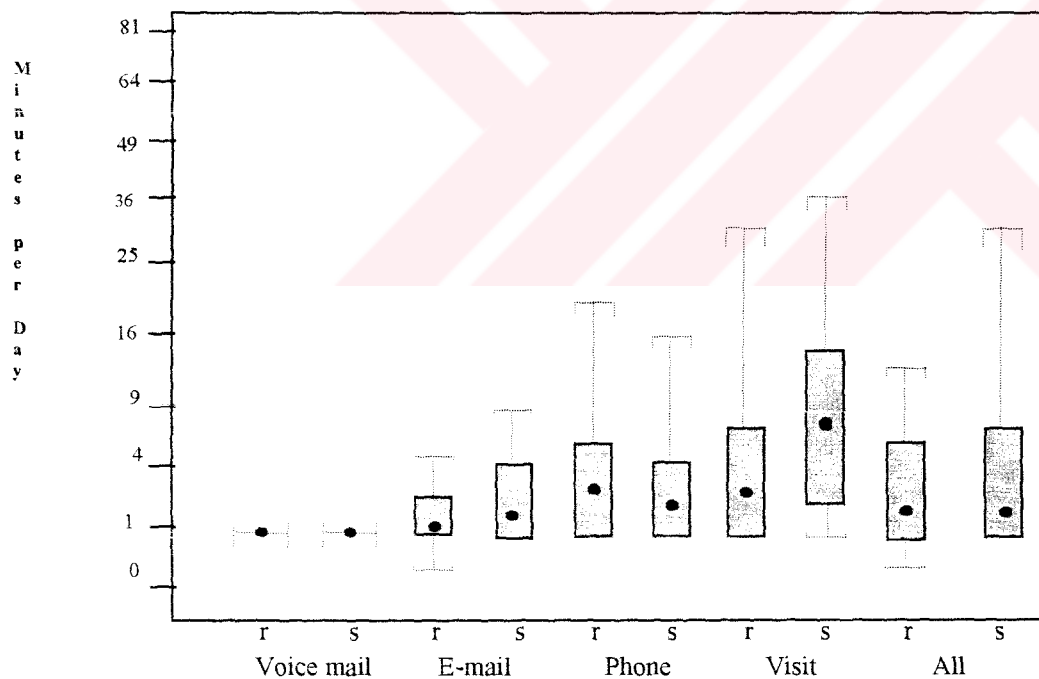
**Figure 2.4 Messages sent and received across four media types. The figures shows the number of messages sent and received, by media type and according to whether they were received (r) or initiated (s).**

The data on the length of communication in the same research is shown in Figure 2.5. The results show that voice mail messages are very brief, one minute. Telephone conversations are also brief two or three minutes. Both receiving and sending messages require the same amount of time in telephone talks and voice mail. However, e-mail is shorter than the telephone talks. People need less time to read an e-mail item than to prepare and send one but this is not surprising as composing a message requires more time. The longest one is visits naturally because you spend time on going and coming.

Perry and Votta's research show that individuals do not use communication media effectively. Even though it is time consuming, they prefer visits because telephone and e-mail are insufficient for transmitting delicate nuances as diagrams, figures, and graphs. As a result, we can say that simpler, user friendly and more powerful tools providing advanced telecommunications should be developed. Suggestions are made below about the addition of voice, vision and drawing to the e-mail technology.

- **Drawing:** It is like a white board on the desktop. All members share access and can write and draw diagrams as they like. Information can be saved and screens can be displayed on a board to facilitate discussion.
- **Voice:** Microphones can be attached to members' computers. Thus subtle nuances of voice and facial expressions are faithfully transmitted and the need to compose long e-mail messages is eliminated.
- **Vision:** Electronic mail and voice mail have facilitated staff communications but there are still times when face-to-face discussions are required. Video cameras can be attached to computers providing simultaneous data transmission for exchange opinions in detail.

Advanced telecommunication tools can reduce product development costs and minimize the need for movement between locations. Furthermore, they help the members continuously contact with each other out of work. All these motivate the commitment of members towards teamwork and increase the performance of the team.



**Figure 2.5 Duration per contact by media type.** The duration per contact is broken down by media channel and according to whether the message was received (r) or initiated (s).

---

## CHAPTER THREE

# METHODOLOGIES SUPPORTING TEAMWORK

---

Effective software team starts with knowledge sharing, broad communication and coordination. Even in the teams composed of highly skilled members, communication among members could break down. This leads team failure and big loss. The reasons of this miscommunication and misinterpretation can be summarised as follows:

- The team does not have a certain project plan
- Goals are not defined
- Functions to be designed are not clearly defined
- Scope of the project is not clear
- The roles of team members are uncertain
- Discussions are not documented well and information is lost
- Commitments are oral, not written, therefore members forget their promises

The necessity of coordination has been emphasized by the writer H. P. Jones as:

*Without coordination, the best group members will fail to work as a team and the project will suffer. With coordination, even a workaday group of designer and developers can parcel out necessary tasks and produce something special (Jones, 1998, p:381).*

Therefore in this chapter, we investigate team design methodologies in order to suggest alternative plans for team activities and software process.

### 3.1 The Difficulties of Software Design

The first prerequisite for a successful software product is a good design. Since a good design increases the quality of the software and reduces the cost by defining the possible problem areas, it makes maintenance easier and provides reuse. However, designing software is not easy since software is naturally complex in essence. This complexity occurs due to the abstract entities of software which can not be verified experimentally. The essence of software entities as data sets, relationships among data items, algorithms, and functions are all related with each other and also they are too detailed. In his paper, Brooks underlines the difficulties of design as:

*"I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. "(Brooks, 1987, p.10)*

The complexity rises as the system gets larger and this creates problems even for a person developing a program individually. However, complexity enlarges more within a teamwork. Everybody has a different idea, approach and design model for the problem domain. In order to minimise this fuzzy atmosphere, entities should be concretized somehow. We can achieve this if we can clearly define each phase of our design.

For an effective development of software, a number of software processes (life cycle models) have been defined and new research are continuing. Today, the below mentioned models exist:

- Waterfall (Royce, 1970)
- Spiral, iterative or evolutionary life cycle (Boehm, 1981)
- Rapid application development lifecycle (Arthur, 1992)
- Object -oriented development lifecycles (Coad & Yourdon 1991, Jacobson et al 1992, Booch 1994)

Life cycles define work activities in each phase of software development. Common phases defined in most of the models are (Jones, 1998):

- Planning
- Requirement definition
- Solution Design
- Implementation

Using the life cycles is a very important step that can not be overloaded. They provide great help for software design by decreasing its natural complexity and as a result the quality of software increases. On the other hand, life cycles do not include the activities which we call Early Design Meetings. With these meetings a very useful teamwork atmosphere is created thus an immediate motivation is accomplished. Members get to know each other and the roles and the task of the members are also defined.

### **3.2 The Functions of Early Design Meetings in a Teamwork**

Early Design Meetings correlate with the phases of Planning and Requirement definition in standard life cycle. During Early Design Meeting session, team members discuss the requirements, work on the project plan and develop solutions. Early Design Meetings are like the bricks of an effective team wall because during these sessions below mentioned steps are achieved:

- The team is set up.
- The members get to know each other.
- The goal of the team is defined.
- The requirements are clarified.
- Members are motivated towards the work.
- The team spirit is felt.
- Members learn to act as a real team.

However, traditional lifecycle methodologies do not address above mentioned trends. As Jones (1998) stated bringing productive and creative individuals together in a room with assigned team members is not by itself likely to create great works. When a group of people are brought together important things to be considered are how to engage them to do work as a team, achieve collaboration among them, document decisions and start design. For this reason some team design methods have been developed addressing both social factors and design solutions.

### 3.3 Team Design Methods

Team design methods manage team activities such as software design and group planning. They not only include the life cycle models but also explain in detail the activities during teamwork. These methods help to organize team workshops, processes and offer a comprehensive set of methods from which teams can choose depending on their needs. Existing models are Joint Application Design (JAD), Rapid Application Development (RAD), Participatory Design (PD) and Team Design (TD). Below, we summarize these methods previously studied by Jones (1998), Damian, Hong, Li, Pan (1998).

#### 3.3.1 Joint Application Design (JAD)

Joint Application Development (JAD) was originated in IBM in the late 1970's. Traditionally, JAD has been a joint venture between users and data processing professionals. In recent years, it has expanded among people who need to make decisions affecting multiple areas of an organisation. It is a structured workshop (usually called JAD session) where people come together to plan projects, design systems, or make business decisions. Each JAD session has a detailed agenda, visual aids, a facilitator who moderates the session, and a scribe who records the agreed-upon requirements. A final document containing all the decisions made by the group is generated after each session.

JAD was originally designed to address information system development. It used to involve some aspects of system design, or, at least, development. But now, the use of JAD techniques has expanded to handle a broader range of challenges. Nevertheless, all the sessions are still facilitated sessions which are led by the facilitators.

JAD consists of five phases. Each of them has its own emphasis and tasks. The diagrams used in this subsection are taken from Wood and Silver (1995) (Figure 3.1).

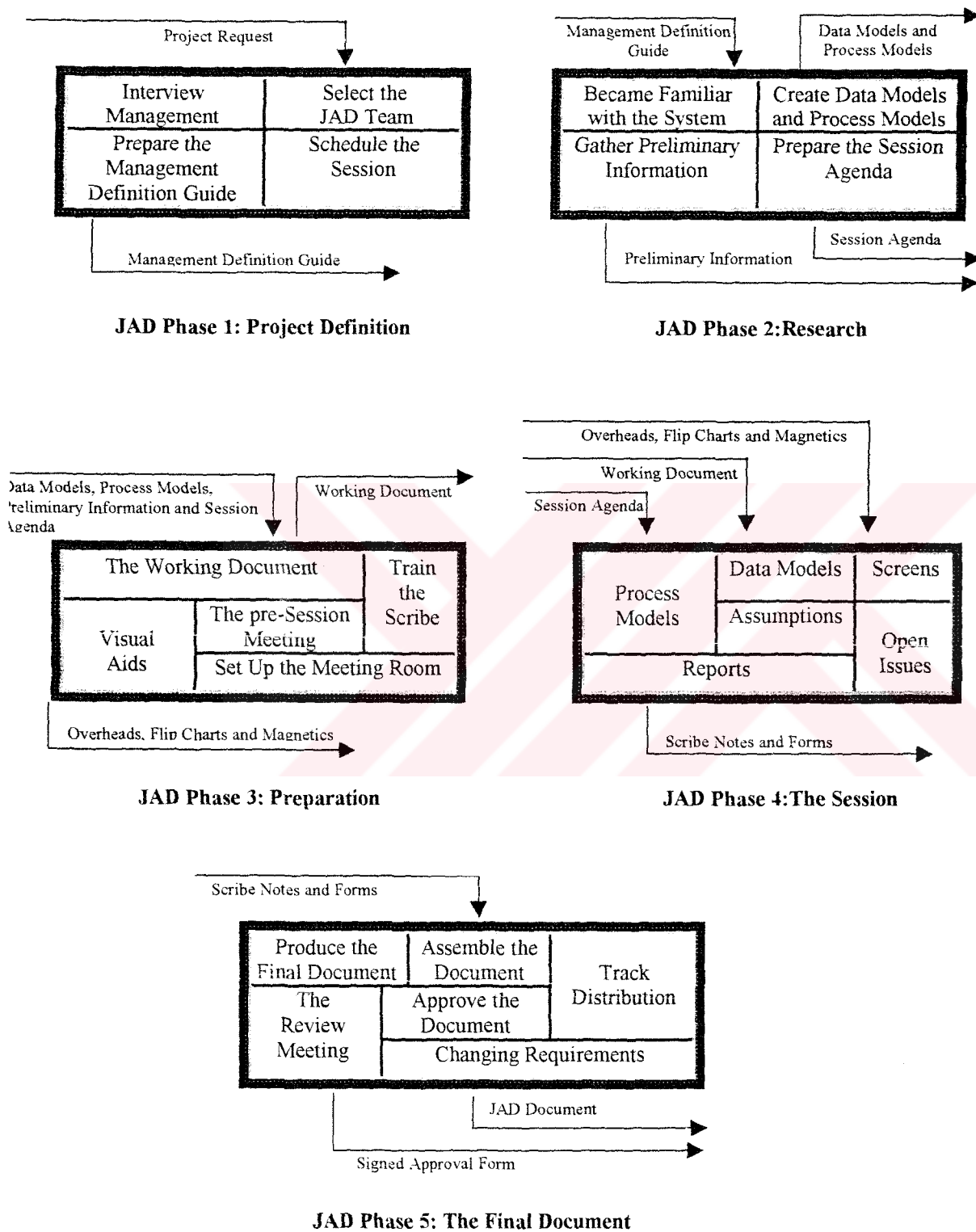


Figure 3.1 JAD Phases (Wood & Silver, 1995)

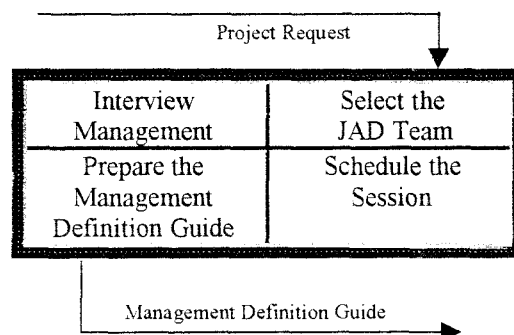
### Joint Application Development Phases:

Five phases of JAD (Figure 3.1) evolve from a definite start to a clear finish. These phases are:

- 1. Project Definition.** This phase involves defining project purpose, scope, and objectives, identifying JAD team members, setting schedules, etc.
- 2. Research.** This phase involves gathering more details about the user requirements, exploring problem domain, considering design issues, etc. Based on the research, an agenda is prepared listing what needs to be decided in the session.
- 3. Preparation.** This phase involves preparing everything you need for the session, such as visual aids, working document, flip charts and overhead transparencies.
- 4. The Session.** This is the actual workshop session. JAD team members define business process, project requirements, etc. Agreed-upon decisions are documented for the final document.
- 5. The Final Document.** The information captured in the session is used to produce the final document. This is the final products of JAD session.

### JAD Phase 1: Project Definition

The process of phase 1 is illustrated in figure 3.2.



**Figure 3.2 JAD Phase 1: Project Definition (Wood & Silver, 1995)**

The main tasks to be accomplished, as shown in the above figure, include:

- Interview management - first high-level interview to identify what management wants from the project, that is to say the purpose, scope and objectives of the project.



- Produce the Management Definition Guide
  - Contents of the Document
  - Send the completed Management Definition Guide to the contributors for their review.
- Schedule the session
 

The session schedule is influenced by the scope of the project and the time constraints.

## JAD Phase 2: Research

The main tasks to be accomplished, as shown in the figure 3.3, include:

- Become familiar with the business
- Document data requirements : Create data models
- Document business requirements: Create process models

Defines the rules for using the data.

- Gather preliminary information

Gather information about the business requirements. The kind of information to gather depends on what you want to accomplish in the session.

- Prepare the session agenda

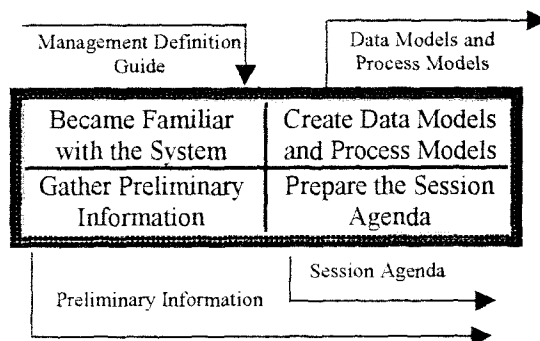


Figure 3.3 Phase 2: Research (Wood & Silver, 1995)

### JAD Phase 3: Preparation

The main tasks to be accomplished, as shown in the figure 3.4, include:

- Produce the Working Document

All the information gathered from the above phases is compiled into the Working Document.

- Send out the Working Document

Send the document to all participants at least one week before the session begins so that they have time to review the document and do any research or preparation necessary.

- Train the scribe

- Produce visual aids

Visual aids help keep the participants focused and can clarify the decisions being made.

- Hold the pre-session meeting

The purpose is to establish management commitment, summarize the JAD process, and distribute and discuss the Working Document. Also, this is the first time all the participants will be together and have a chance to establish group rapport.

- Set up the meeting room

- Using checklists

Use a couple of checklists for JAD tasks and for JAD supplies.

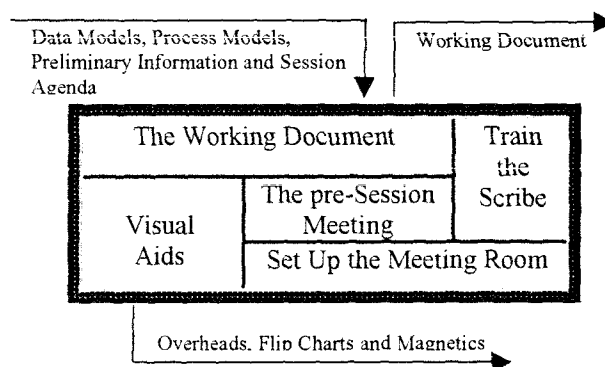
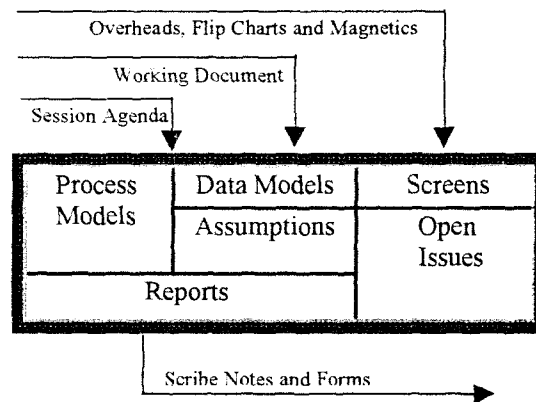


Figure 3.4. Phase 3: Preparation (Wood & Silver, 1995)

## Phase 4: The Session

The process of phase 4 is illustrated in figure 3.5



**Figure 3.5. Phase 4: The Session (Wood & Silver, 1995)**

The main tasks to be accomplished, as shown in the above figure, include:

- Opening the session: facilitator start the session.
- Discuss Assumptions
- Define Data Requirements
- Define Business Processes
- Design Screens
- Design Reports

Define all output from the system. Besides standard reports, this includes any other print outs generated in the process, such as invoices, statements, checks, and labels.

- Other agenda items

Depending on the kind of JAD you are running, there are all kinds of other items your agenda might include.

- Resolve Open Issues

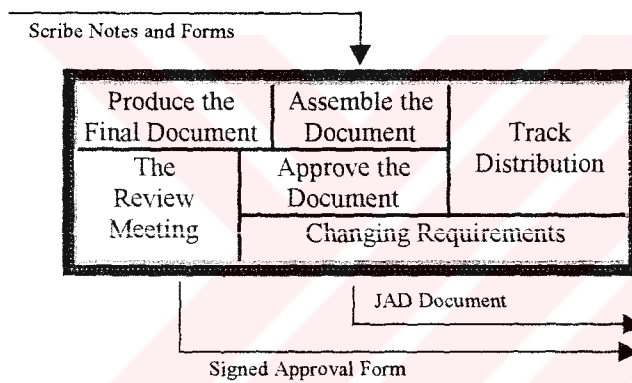
Open issues are added throughout the session. At the end of the session, all those open questions are addressed.

- The scribe takes notes whenever a prompt is received from the facilitator.
- Evaluate the Session
- Close the Session

### JAD Phase 5: The Final Document

In this phase (Figure 3.6) all the agreements made in the session are transferred into the final document and the final document is assembled and distributed to the participants for review. Finally, signatures approving the final document and for systems development projects are taken, the development team is authorised to begin the next phase of the life cycle.

The final document is a comprehensive synthesis of agreements made in the session. It is the one resulting document, the one final product that represents JAD's role in the process. For the people who were not participants but have a line of responsibility for that project, the final document may be the only evidence they have to judge the status of the project after the session.



**Figure 3.6 Phase 5: The Final Document (Wood & Silver, 1995)**

#### 3.3.2 Rapid Application Development (RAD)

Rapid application development (RAD) refers to a development lifecycle designed to give much faster development and higher-quality results than those achieved with the traditional lifecycle (Booknews, 1991). RAD is a series of iterative working sessions used to create a working prototype of the proposed system.

When a project requires a truly rapid design and development, it is useful to plan a single extended JAD workshop. Used together with JAD, it creates systems users want very rapidly (Reilly, 1997). For a smaller RAD project, one workshop of two or three days may cover the needs for all phases of the project.

### 3.3.3 Participatory Design (PD):

The primary goal of Participatory Design (PD) is the involvement of the individuals who do work in a design process. The responsibility for the product is shared by the users and designers.

The socio-technical approach which began in England by Enid Mumford, and the collective resource approach developed in Scandinavia by Pelle Ehn and Morten Kyng challenge this design tradition (Bjerknes, Ehn, Kyng, 1987). The result of applying the socio-technical approach in Scandinavia is the developing of Participatory Design (PD) in late 1970's. Now the area of PD has been growing rapidly, in terms of numbers of practices, extent of theoretical development, numbers of practitioners, and geographical and institutional diversity of practice. It has been only more recently been used in North America (Jones, 1998).

#### **PD Advice:**

- to attract the interest of users, it is important that the focus be on addressing their immediate needs.
- the project group is likely to function better in an environment away from everyday pressures so participants can focus on learning from each other, practicing skills, and developing systems.
- have management support
- specify in a contract how much time the users can /shall spend on the project
- have a steering group in which conflicts can be discussed.
- be sure the required equipment is available for systems experts and users.
- listen to the user, but do not do everything the user proposed.

The advantages of using PD are (Harris and Taylor, 1996):

- involving users of different organizational groups and provide a forum in which the participants can better understand
- boundaries and explore ways to overcome them
- shared understanding and knowledge of the process enables individuals to anticipate the consequences of their actions.
- increasing worker's influence on technological change

- participation in the design process also prepares people for changes
- participation is used as a tool to open up conflicts and negotiated throughout the design processes

PD is a complex process involving technology and multiple levels of organization. It is also highly dependent on specific organizational contexts. For project participants, this means there are no programmatic solutions. To the extent that PD projects have been documented to date, there is little evidence that a "standard" set or ordering of practices has been decided. However, PD efforts in recent years have led to repertoire of flexible practices and general guideline. Figure 3.7 (Muller, Wildman, and White, 1993) shows a brief guide to PD practices for practitioners.

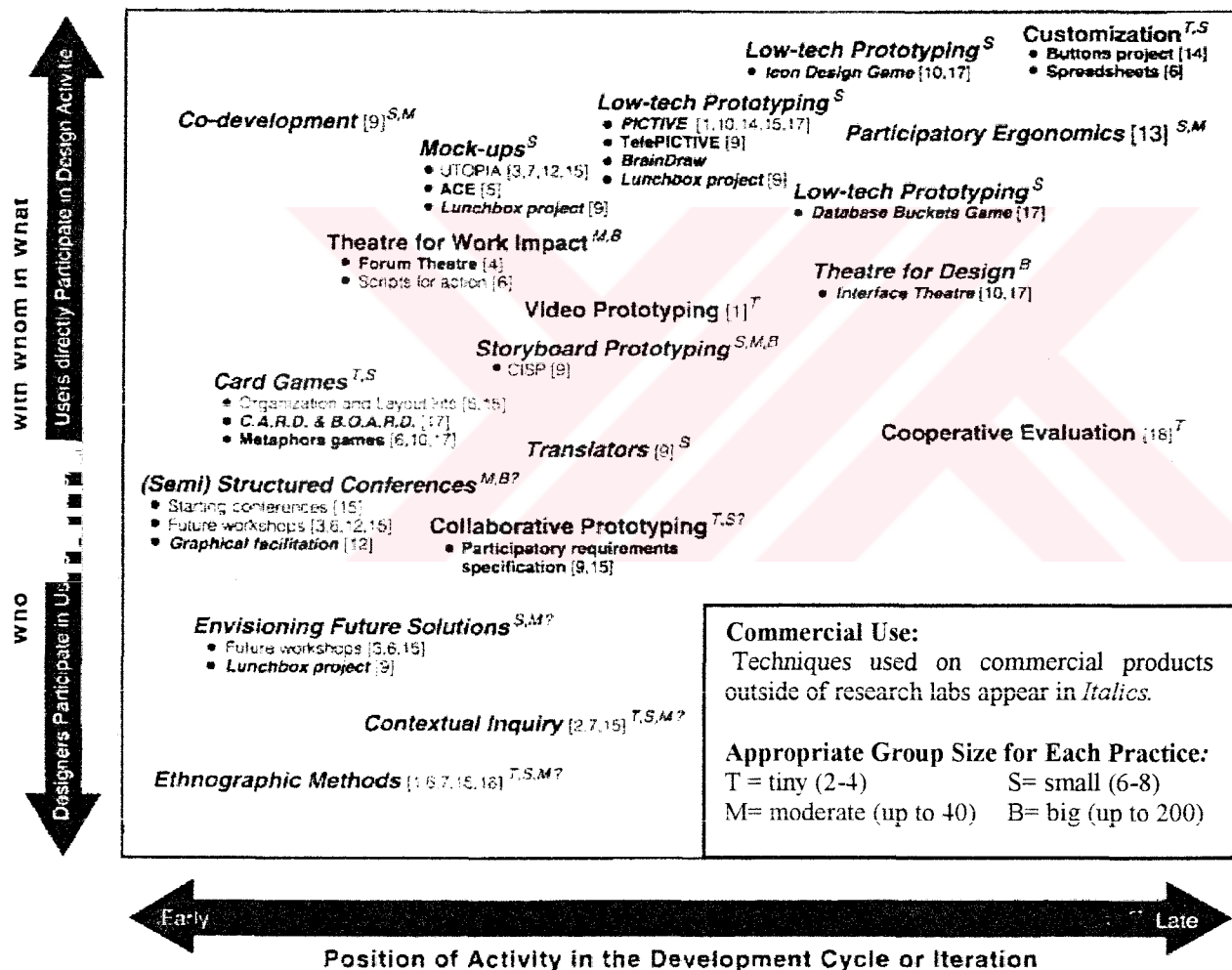


Figure 3.7: Techniques Applicable in Participatory Design

Notions in the above figure are described below:

**Time during the development life cycle:** some practices appear to be more appropriate at certain points within the development life cycle or iteration. The horizontal axis of the figure provides a very approximate guide to points within the life cycle at which each practice may be useful.

**Who participates with whom in what?** The concept of participation is open to multiple interpretations. The vertical axis of the figure spans one way of organising the various approaches, the software professionals can participate in the users' world (lower on the axis), or users can participate in the software professionals' world (higher on the axis).

**Appropriate group size for the practice:** Different practices are designed to work with groups of different sizes. Appropriate group sizes are indicated by superscript letters for each category of practice: T (tiny, 2-4 participants), S (small, 6-8 participants), M (moderate, up to 40 participants), and B (big, up to 200 participants). The group size recommendations are in some cases approximate.

### 3.3.3.1 Comparison between JAD and PD

#### *User Involvement*

The main common characteristic of PD and JAD is that they advocate a strong user involvement in system design, in which workers actively engage in designing the computer system they will eventually use (Carmel et al., 1993).

In JAD, both operational workers and managers are considered "users". The term "user" from the JAD perspective does not indicate rank or position, but simply organizational affiliation.

In PD approach user participation is mandatory because users are viewed as primary source of knowledge. The focus is on lower-level, operational users - often excluding



management from the process. PD practitioners presume that operational users are the most qualified authorities on improving their workplaces (Carmel, Whitaker, and George, 1993).

Both approaches face the same obstacles in a successful implementation: managerial resistance, user conservatism, lackluster workshops and poor facilitators. Getting user participation may create troubles as users themselves can be uncooperative and unmotivated.

### *Structure*

JAD is a very structured approach, in which manuals and guides are like a cookbook. The JAD approach where everything is explained in great detail emphasizes structure and agenda.

PD emphasises creativity. PD practitioners do not advocate of prescriptive methods, however, they do use a number of techniques such as: Future Workshops, Cooperative Prototyping, Design Mock-ups, Future Games, etc.

### **3.3.4 Team Design (TD):**

Team Design incorporates structure and practices from several types of facilitated workshops and planning sessions for analysis and design, including Joint Application Design (JAD), Participatory Design (PD), facilitated planning, prototyping workshops and total quality management processes. JAD approach is highly structured in terms of the control of the meeting and the guidance of participants and expects a consensus in a team and PD expects user-centred democratic process. These methods require the facilitator to help to establish and maintain the team context. The process of working together towards those solutions is based in the notion of teamwork. As the teamwork is not given, in JAD and PD, it should be developed by the team, usually with a leadership and guidance. The Team Design approach addresses this process of working with creative groups and integrates the best practices of researchers and facilitators within a team model and the facilitation could be shared among the participants. Besides, less structured is necessary to provide a working environment where team members can be allowed to think and develop solutions creatively and also less structured helps organisations to relax the cultural constraints and to generate ideas and proposals from a more creative perspective. Team Design method gives organisations a framework from which to build their own custom design practices. A sample model format for Team Design (Jones, 1998) is shown in Table 3.1



**Table 3.1 Model Team Design Format (Jones, 1998)**

Agenda Activities	Methods	Inputs and Outputs
<i>Planning: Preworkshop</i>		
Advance planning for workshop	Project planning, team planning, workshop planning	In: Business case, schedules Out: Workshop plan (initial)
Preparation for workshop	Team and workshop planning, coordination interviewing and surveys, discussions	In: Workshop plan (initial) Out: Workshop plan (initial)
<i>Initiating: Workshop</i>		
Welcome	Agenda, ground rules	In: Workshop plan Out: Agenda
Team Building	Warm-ups, introductions, exercises	In: Agenda Out: Team identity
The workshop process	Agenda, discussions	In: Workshop plan, agenda Out: Project goals
<i>Scoping: Identification</i>	Brainstorming, scoping diagrams	In: Mission statement , project goals Out: Purposes
<i>Visualizing: Analysis</i>	Brainstorming, system diagrams	In: Purposes, scope Out: System and process vision
<i>Usage: Application</i>	Scenarios, prototypes	In: System and process vision Out: Scenarios, prototypes
<i>Packaging: Completion</i>	Design diagrams, mapping, final documentation	In: Scenarios, prototypes Out: Design package
<i>Validating :Evaluation</i>	Design evaluation, test planning	In: Prototype design package Out: Validate design

Three columns in each table list the following:

- *Agenda Activities*: Phases of design (shown as the italic listings) are indicated for each major set of agenda activities.
- *Methods*: Methods are associated with activities within a phase.
- *Inputs and outputs*: Inputs are required deliverables or conditions to enable a given activity. Outputs are recommended products developed as a result of completing an activity.

The phases shown in the model format a full-extended set of activities, representing an end to end design process. These phases are described as follows:

- **Planing: Preworkshop.** The preworkshop is used as the planning period for the workshop organiser, the facilitator and the project manager to plan and prepare for the session. The idea is to set aside enough time in the preworkshop period to sufficiently prepare for the workshop.
- **Initiating: Workshop.** The initiating phase starts the workshop and integrates all of the introductory sessions and most of the team-building processes. Initiating is used at the start of all projects and series of workshops.
- **Scoping: Identification.** Identification of scope and process. Scoping phase involves identifying the components and boundaries of the problem or requirements addressed by design.
- **Visualising: Analysis.** Analysis of the process. This phase bridges the design from current analysis to the new requirements or vision. Numerous front-end analysis methods are used in visualising to enable teams to construct appropriate design models.
- **Usage: Application.** Application of the process in context. Usage integrates operational models, scenarios and use cases, and prototypes in order to surface issues, test assumptions and iterate the design within an identifiable context.
- **Packaging: Completion.** Completion of the design process. Packaging is the phase where a design model or specification is produced from the analyses, iteration of design, and learning of the prior phases.
- **Validating: Evaluation.** Evaluation of the design and process. Validating is a phase recommended for review and evaluation, which is typically used in analysis and design formats.

### 3.3.4.1 Team Design Phases: Planing and Initiating

#### Planing: Preworkshop.

In this phase, the idea is to set aside enough time in the preworkshop period to adequately prepare for the workshop. For a two-day workshop, a week or more of advance work might be required to ensure that the workshop is well planned and productive. Also participating team members must have enough advance time to digest advance information and make preparations too be capable of collaborating in the workshop.

*Advance planning for workshop:* Advance planning for workshop starts with the project and workshops planning factors. These activities involve the general methods shown in Table 3.2, including project, team, and workshop planning. For a new project or a new workshop series, the first inputs to planning will likely be a project description or business case and any schedules that have been prepared. The essential output for the workshop will be the workshop plan, which includes project and team planning relevant to the workshop.

*Preparation for workshop:* Preparation for workshop might require continued planning, with participation of sponsors or management and discussions and interviews with participants. The methods for preparation can include any activity necessary to establish the groundwork for the workshop. This activity includes coordinating with stakeholders, discussing roles and participation with the entire team and gathering information required to better understand the processes or systems to be designed. Interviews are used in advance to gather specific detailed information from participants. Surveys can be used as a means of tapping into the larger organisation and gathering feedback on a number of issues relevant to the workshop. The initial workshop plan is the primary input and the revised and final workshop plan is the output.

**Table 3.2 Team Design: Preworkshop Planning (Jones, 1998)**

Planning: Preworkshop		
Agenda Activities	Methods	Inputs and Outputs
Advance planning for workshop	Project planning, team planning workshop planning	In: Business case, schedules Out: Workshop plan (initial)
Preparation for workshop	Team and workshop planning, coordination interviewing and surveys, discussions	In: Workshop plan (initial) Out: Workshop plan (initial)

### Initiating: Workshop.

A mini-initiating session should be held at the start of every new meeting as well, to check in with participants, obtain feedback about the process, and take informal measures of perceived progress.

Team Building Activities: Team building activities follow the welcoming and these can take the form of many different approaches and exercises. The goal or output of this activity is a group with a team identity, ready to work together over the project or at least, the workshop period. Introductions are typically made, and warm-ups and small group exercises are used for members to learn about each other. If the team uses a team building method, such as Team Spirit (Heermann, 1997) or methods from the Team Handbook (Scholtes, 1988), would be started at this point.

The workshop process: The workshop process activity provides the opportunity to discuss the workshop agenda, and consensus can be reached on agenda topics, ground rules and scheduled items. Discussions are held with the team to craft policies and agreements for working together. The goals for the project can be discussed openly, and agreements and proposals affecting the workshop. The outputs for this activity is project goals that have been shared, discussed and understood among the team.

**Table 3.3 Team Design: Initiating Phase (Jones, 1998)**

Initiating: Workshop		
Agenda Activities	Methods	Inputs and Outputs
Welcome	Agenda, ground rules	In: Workshop plan Out: Agenda
Team Building	Warm-ups, Team Handbook, Team Spirit, introductions, exercises	In: Agenda Out: Team identity
The workshop process	Agenda, discussions	In: Workshop plan, agenda Out: Project goals

### 3.3.4.2 Team Design Preworkshop Planning and Initiating Phase Methods

#### Workshop Planning

Any workshop requires planning and coordination that reflect the amount of advance preparation. Several significant planning considerations determine your workshop structure, duration and style. These are described in the following:

***Size of the project: small, large-scale or phased.*** Project size guides the breadth versus the depth of the workshop. With a smaller project, a single workshop may be all that's required, at least for each stage of the life cycle (analysis, design and development). A large project might involve numerous stakeholders and more team members and workshops from three days to a week should be scheduled for the start up period.

***Complexity of the design: simple, complex or unknown.*** For quick planning purposes, consider complexity as a rough guess based on the number and interrelatedness of requirements and the team's relative experience level in the area. If the design is considered simple, be cautious and plan the workshop as if it were of medium complexity.

***Project deadlines and milestones:*** For near-term deadlines, the team will be pressured to conduct a rapid workshop, settle decisions by compromise and cut corners on meeting requirements. Midterm deadlines are any milestones that are not under extreme pressure for immediate delivery.

#### Project Planning

Project planning, involving team planning for the project and its phases, presents a major opportunity for team facilitation. Facilitated project planning creates a productive atmosphere for working through technical and organisational issues. At the project planning workshop, clients will have a project definition, resource plan and bar chart schedule for the project's lifespan. Several aspects of project planning:

- Project scoping and definition
- Project scheduling and resourcing
- Development planning
- Organisational issues resolution

## Team Building:

Two approaches for team building are The Team Handbook (Scholtes, 1988) and Team Spirit (Heermann, 1997).

The Team Handbook (Scholtes, 1988), provides a cookbook of methods and guidelines practitioners and participants:

- *Forming*: Social relationships are established and the group initially begins to work with the leader.
- *Storming*: Conflict begins to emerge between members as they recognise differences in goals and perspectives.
- *Norming*: Group members recognise commonalities and shared interests in the team and establish process for communication.
- *Performing*: Team members generate insight s into the team's processes and work cooperatively through group problems.

Team Spirit (Heermann, 1997), presents a holistic view of teams that integrates thinking from organisational, scientific and spiritual foundations. It has five phases that characterised in order as follows:

- *Initiating*: Introducing and learning about team members and their interests, disclosing and sharing and creating a common sense of belonging.
- *Visioning*: Developing a shared vision or purpose as a group developing mutual interests, becoming closer through sharing ideas and establishing a team presence.
- *Claiming*: Identifying and aligning with roles and goals available in the team, developing organisational support and identifying competencies and personal goals.
- *Celebrating*: Bringing about team celebration through recognition and individual accomplishment.
- *Letting Go*: Allowing for constructive feedback among team members, providing disclosure and straightforward communication.

## Agenda:

Workshop agendas identify each major topic of discussions and define the timeframe allocated for work. The basic elements of any agenda are described on the example from shown in figure 3.8.

*Following the agenda:* Select an appropriate workshop template and create an initial agenda. Be sure to ask workshop participants for their input to the agenda in advance of the session.

Team Design Workshop Agenda		
<b>Project:</b>	<b>Date:</b>	<b>Room:</b>
<b>From:</b>	<b>Start time:</b>	<b>End time:</b>
<b>Attendies/team:</b>		
<b>Purpose of workshop or meeting:</b>		
<b>Facilitator:</b>	<b>Please be prepared to:</b>	
<b>Workshop activities:</b>		
Activity	Time	Person responsible
1. Introduction:	8:15 - 8:45	Facilitator
Ground rules and procedures		
2. Project overview	8:45 - 9:30	Project leader
3. Requirements discussion	9:30 - 10:15	Project leader
Review document in advance		
Break	10:15-10:30	

**Figure 3.8 Sample workshop agenda (Jones, 1998)**

## Introductions:

Introductions can be handled by the organiser, facilitator or participants. Have people describe their work and their role on the team, since this is the place to establish some context for working together. Members answer several simple questions about themselves (collage attended, favourite sport, last vacation spot, hobbies and interest) or introduce their neighbour.



### Ground Rules:

Ground rules are a must and should always be presented at the first meeting of a new group.

1. Start of the ground rules discussions by listing a partial set of rules that most members will agree with, such as: Arrive and end on time and follow the ground rules.
2. Explain to the group the importance of ground rules and how they support holding a smooth and productive workshop.
3. Ask the group to add any other rule to which they want to hold.
4. After five to seven ground rules have been listed, offer to close the discussion. Ask whether all are satisfied with the rules and whether everyone can live with them.

#### 3.3.4.3 Team Design Formats:

Team Design method (Table 3.4) offers five formats which guide the practice of design techniques. Each format provides a structure to follow for *Scoping*, *Visualising*, *Usage*, *Packaging*, and *Validating* and includes the following:

- A baseline agenda to use as a starting point
- A set of group activities to generate the deliverables for the workshop
- Descriptions of practices for using the appropriate development methods.

**Table 3.4 Team Design Format (Jones, 1998)**

Activities
<i>Planning: Preworkshop</i>
<i>Initiating: Workshop</i>
<i>Scoping: Identification</i>
<i>Visualising: Analysis</i>
<i>Usage: Application</i>
<i>Packaging: Completion</i>
<i>Validating: Evaluation</i>



The five formats of Team Design Method are:

- **Business Process Design:** This is a front-end process which formats team in starting with an objective and completing with a redesigned business process.
- **Requirement Definition:** This is another front-end format supporting teams in evaluating the initial business requirements and working through to completion of them. This can be considered the design of requirements.
- **Application Design:** This format integrates application analysis and design activities into a team workshop format. Unlike the other formats, Application Design integrates a front-end format as input - either requirements or process design.
- **Team Planning:** Planning is typically a front-end activity, but this format can be used during workshops, for project planning, scheduling and business planning.
- **Decision Making:** The Decision Making format supports team-based decision processes. This format can be used at any point within the development or in nondevelopment decisions. Decision Making follows the same design cycle as the other formats, but in a much shorter time period.

Development teams might use these five analysis and design formats independently, sequentially or cumulatively.

---

## CHAPTER FOUR

# A TOOL FOR EARLY SOFTWARE DESIGN MEETINGS (EDT)

---

Early Design Tool (EDT) is a software support tool designed to be used for Early Design Meetings and workshop activities in a teamwork. Early Design Meeting activities include planning and designing stages that occur before coding in a software development project. During Early Design Meeting sessions, team members discuss the project plan, requirements and develop solutions.

All the meetings hold in a team are important, however, early meetings are the most delicate as the team starts to form, people get to know each other and they suggest alternative solutions for design. Unfortunately, software teams today are not aware of the importance of Early Design Meeting activities and therefore most of the teams face the following problems:

- Loss of information which coming from the discussions due to manual documentation
- Loss of time and obligation to enter the information to CASE tools by hand
- Difficulties of transferring information with an appropriate format to the CASE tools
- Adaptation problems with the system support tools

It is astonishing that there is no tool in the computer world to support these activities. So we decided to develop a tool as a support system for Early Design Meetings. This tool addresses to the above mentioned problems and has following advantages:

- Decreasing the loss of information
- Transferring data safely
- Saving time

- Transferring the data into CASE tools easily
- Supporting effective team work by being user friendly
- Lessening the problem of software complexity

In summary, in developing such a tool we aimed firstly to help the teams to get rid of the struggles they have, secondly, to improve the quality of software with the advantages mentioned above.

#### 4.1 Early Design Meetings

For the tool EDT, we investigated Early Design Meeting Activities according to Team Design Method. The activities in the Early Design Meeting are *Project Planing*, *Team Planning*, *Scoping* (Table 4.1).

**Table 4.1 Team Design Format (Jones, 1998)**

Activities
<i>Planning: Preworkshop</i>
Advance planning for workshop
Preparation for workshop
<i>Initiating: Workshop</i>
Welcome
Team Building
The workshop process
<i>Scoping: Identification</i>
<i>Visualizing: Analysis</i>
<i>Usage: Application</i>
<i>Packaging: Completion</i>
<i>Validating :Evaluation</i>



**EARLY DESIGN MEETING  
ACTIVITIES**

## 4.2 The Issues of Design

We investigated the *Scoping phase* of **Requirement Definition Format** (Jones, 1998). This phase is called **Requirements Gathering**. We took **Requirements Gathering** activities as a frame and developed it as a support tool of a team (Table 4.2).

The requirement analysis is the foundation for developing effective team design workshop. Requirement process begins with the problem definition, initial gathering and results in understanding of the user and product requirements. The activities in this phase are *Exploring the purpose*, *Defining the scope* and *User and system goals* (Jones, 1998, pp:338-340).

- **Exploring the purpose:** The team identifies the goals, assumptions and business needs for the system in the very beginning of the analysis. The scope and the overall purposes are considered for requirements definition during this phase.
- **Defining the scope:** Scoping the project is the beginning stage of work when the team is building a shared understanding of the domain. Defining the scope is useful for allocating resources and determining priorities, purposes and objectives. It also provides multiple perspectives.
- **User and System goals:** Defining user and system goals is a useful way to construct the high level frame work for requirements definition. Understanding the goals of users supports the team in making functionality decisions and helps to find the right balance between the user requirements and system goals.

Table 4.2 Requirement Analysis (Jones, 1998)

Requirements Gathering		
Activities	Methods	Inputs and outputs
Exploring the purpose	Dialogue, brainstorming methods, Breakthrough Thinking methods	In: Project Goals Out: System purpose
Defining the scope	Scoping diagram, context diagram process hierarchy	In: Purposes, initial scope Out: Scope definition
User and system goals	Brainstorming methods, Breakthrough Thinking methods, affinity diagram	In: Scope definition Out: User and Scope goals

### 4.3 Methods in Early Design Meetings

The Requirement Analysis activities (Table 4.3) -Exploring the purpose, Defining the scope, User and system goals - are realised with the methods next to them. In our tool, we used three methods namely: Brainstorming, Scoping and Context Diagram.

**Table 4.3 Requirements Gathering (Jones, 1998)**

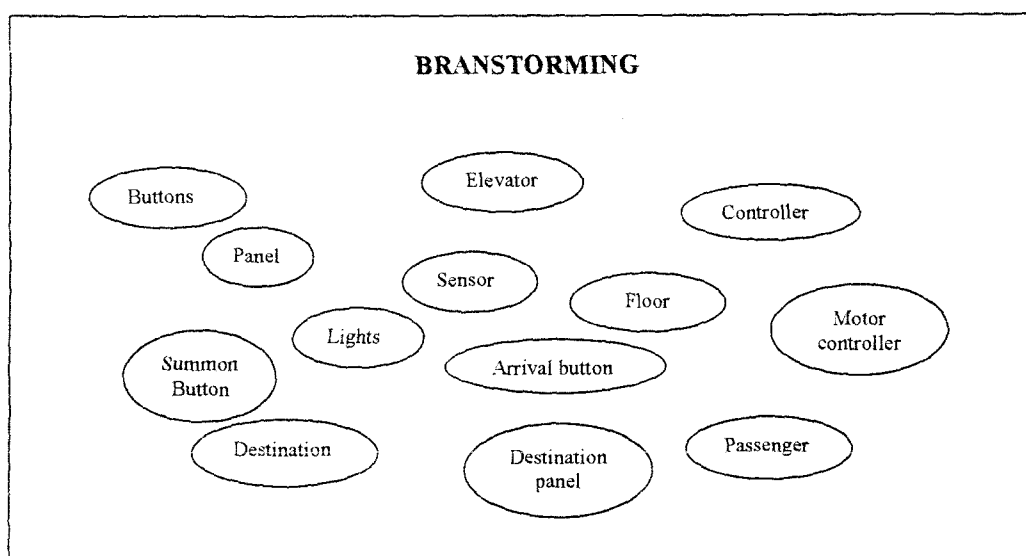
Requirements Gathering	
Activities	Methods
Exploring the purpose	Dialogue, brainstorming methods, Breakthrough Thinking methods
Defining the scope	Scoping diagram, context diagram process hierarchy
User and system goals	Brainstorming methods, Breakthrough Thinking methods, affinity diagram

#### 4.3.1 Brainstorming

Brainstorming is used for generating or elaborating requirements with the users (Jones, 1998). During Brainstorming, potential viewpoints, system services, data inputs, non-functional requirements, control events and exceptions are identified. Any possible way of looking at the project is written down (Sommerville, 1996).

##### *The use of Brainstorming:*

- Contributions from everyone are encouraged.
- Brainstorming inputs are requested from the group one person at a time.
- Each item is entered in one bubble.
- The process is continued until no further inputs are forthcoming.



**Figure 4.1 Brainstorming**

### 4.3.2 Scoping

In this session boundaries for the project and system are defined. Scoping diagrams bring discussions back to the relevant topics and provide a focus for the team.

**The use of Scoping:** The team members are asked to reflect on whether new issues fit into scope or lie outside its bounds.

- Major topics that fit within the scope are identified and written inside the biggest circle.
- Related topics are represented inside the small circles.

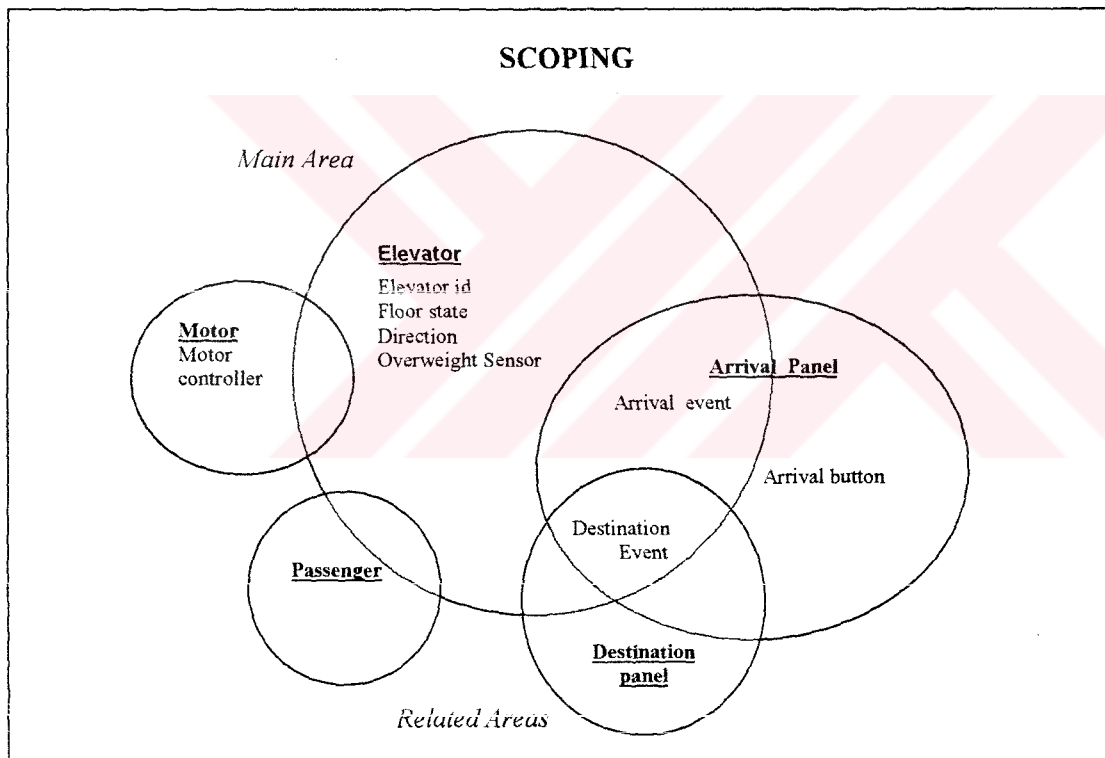
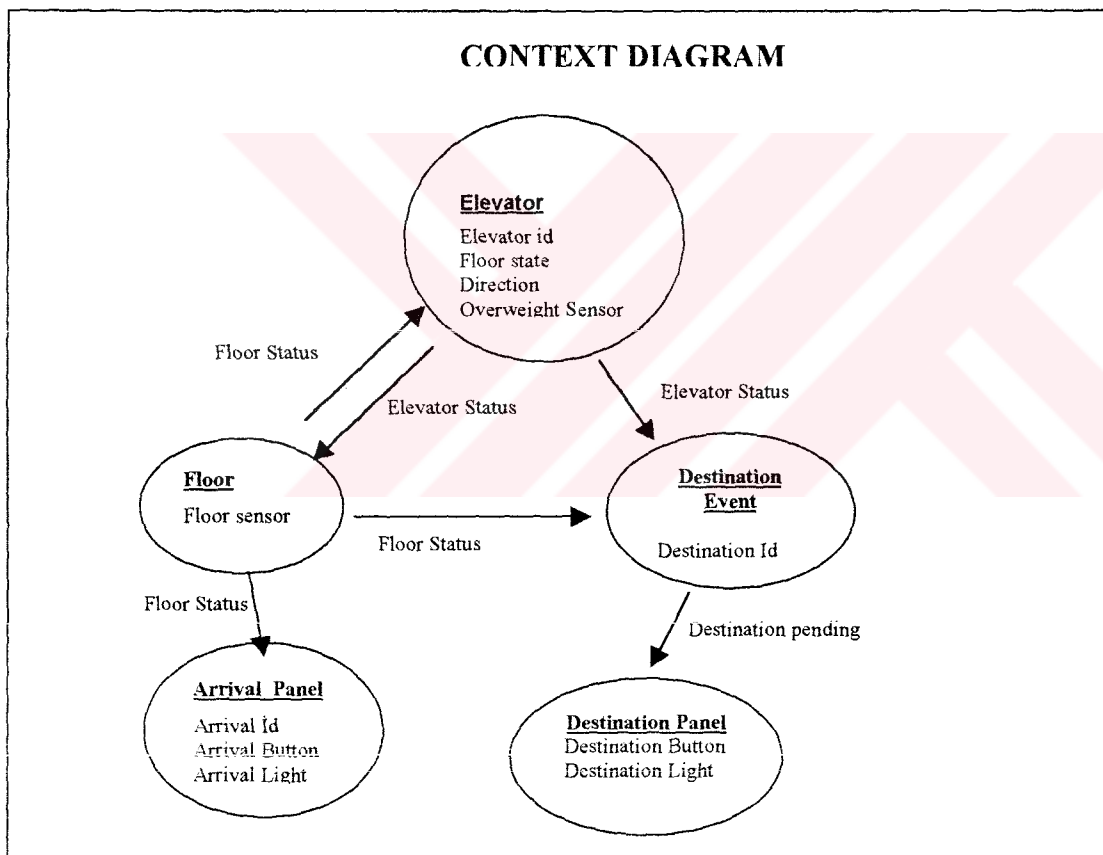


Figure 4.2 Scoping

### 4.3.3 Context Diagram:

Context diagram provides a useful overview of the process and generates the common understanding of work (Jones, 1998). With a context diagram the team can communicate the impact of process design alternatives using a common model as a baseline.

**The use of Context Diagram:** The selected entities which are previously chosen from scoping diagram are transferred to the related areas. The relations among the diagrams are shown with the arrows.



**Figure 4.3 Context Diagram**

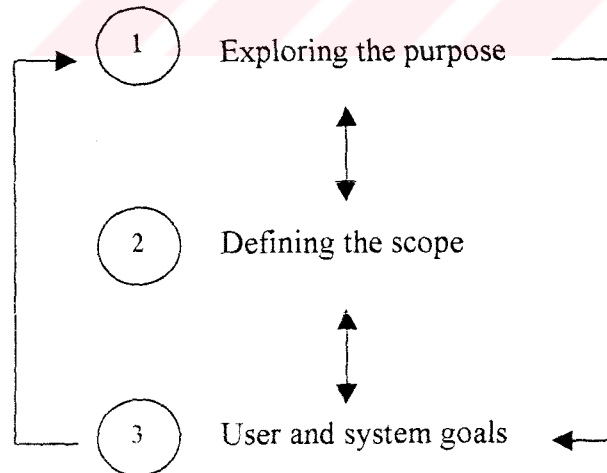
#### 4.4 The positive effects of the tool on software design complexity

Designing software is difficult as software has a natural complexity in essence. The reason of this complexity comes not only from the abstract entities of the software which can not be verified experiment but also from the interrelated relationships among the entities.

**Table 4.4 Requirement Analysis (Jones, 1998)**

Requirements Gathering Activities
Exploring the purpose
Defining the scope
User and system goals

In order to simplify the complexity of software design, each activity should be defined clearly and made concrete (Table 4.4). This is the ideal and desired solution. However, this is not so easy. For example; during the step1 "*Exploring the purpose*", "*User and system goals*" can be taken into account together (Figure 4.1). Because if user and system goals are completely discarded, our purpose can not be attained. Moreover, during the step2 "*Defining the scope*", new ideas may arise and going back to previous step may become a necessity. The same need of going back may occur in step3 and thus the step1 and the step2 might be reviewed again. These are the typical problems team members will struggle with.



**Figure 4.4 Relationship among the Requirements Gathering Activities**

We attempt to solve this problem with the tool we present. The tool has been designed in such a way that you can go back to the previous steps if need be.



## 4.5 Sample Scenario on Early Design Meetings

The team of program developers has regular meetings to develop software design solutions on an existing project using Brainstorming, Scoping and Context Diagram Methods. Each meeting of the team is called “Workshop”. Early Design Meetings consists of five steps.

### Step1: Initiating

Initiating phase starts the workshop, integrates all of the introductory sessions and team building process. The goal of the team building activity is forming a group with a team identity and ready to work together over the project. Introductions are typically made, warm-ups and small group exercises are used for members to learn about each other. (Heerman 1997, Scholtes 1998).

In all workshops, information - such as *Date, Time, Purposes and Participants*- has to be written down.

### Step2: Problem Domain

After the workshops in initiating step team is ready to work together over the project the team begins to study on a Problem Domain which chosen by them or by someone (manager) out of the team. In order to explain the usage of the methods better, a problem domain is given as an example. The following steps and the tool explained in the next section are modelled over this problem domain.

#### **Problem Domain: Elevator Control System**

The general requirement is to design and implement a program to schedule and control four elevators in a building with 40 floors. The elevator will be used to carry people from one floor to another in the conventional way.

**Efficiency:** The program should schedule the elevators efficiently and reasonably. For example, if someone summons an elevator by pushing the down button on the fourth floor, the next elevator that reaches the fourth floor travelling down should stop at the fourth floor to accept the passenger(s). On the other hand, if an elevator has no passengers ( no out-standing destination requests), it should park at the last floor it visited until it is needed again.

**Destination Button:** The interior of each elevator is furnished with a panel containing an array of 40 buttons, one button for each floor, marked with the floor numbers (1 to 40). These destination buttons can be illuminated by signals sent from the computer to the panel. When a passenger presses a destination button not already lit, the circuitry behind the panel sends an interrupt to the computer (there is a separate interrupt for each elevator). When the computer receives one of these (vectored) interrupts, its program can read the appropriate memory mapped eight-bit input registers (there is one for each interrupt, hence for each elevator) that contains the floor number corresponding to the destination button that caused the interrupt. Of course, the circuitry behind the panel writes the floor number into the appropriate memory-mapped input register when it causes the vectored interrupt.

**Destination button lights:** As mentioned earlier, the destination buttons can be illuminated (by bulbs behind the panels). When the interrupt service routine in the program receives a destination button interrupt, it should send a signal to the appropriate panel to illuminate the appropriate button. This signal is sent by the program's loading the number of the button into the appropriate memory-mapped output register (there is such one register for each elevator).

**Floor sensors:** There is a floor sensor switch for each floor for each elevator shaft. When an elevator is within eight inches of a floor, a wheel on the elevator closes the switch of switches in each elevator shaft). When the computer receives one of these (vectored) interrupts, its program can read the appropriate memory-mapped eight bit input register corresponding to the floor sensor switch that caused the interrupt.

**Arrival lights:** The interior of each elevator is furnished with a panel containing one illuminable indicator for each floor number. This panel is located just above the doors. The purpose of this panel is to tell the passengers in the elevator the number of the floor at which the elevator is arriving (and at which it may be stopping). The program should illuminate the indicator for a floor when it arrives at the floor and extinguish the indicator for a floor when it leaves a floor or arrives at a different floor. This signal is sent by the program's loading the number of the floor indicator into the appropriate memory-mapped output register (there is one register for each elevator).

### Step3: Brainstorming

In this step, the team starts design solutions. Brainstorming is done on the problem domain above mentioned. Requirements, control events, exceptions, potential viewpoints and any possible data inputs related with the problem domain are jotted down.

A model result of the Brainstorming according to Elevator Control System is represented in figure 4.5.

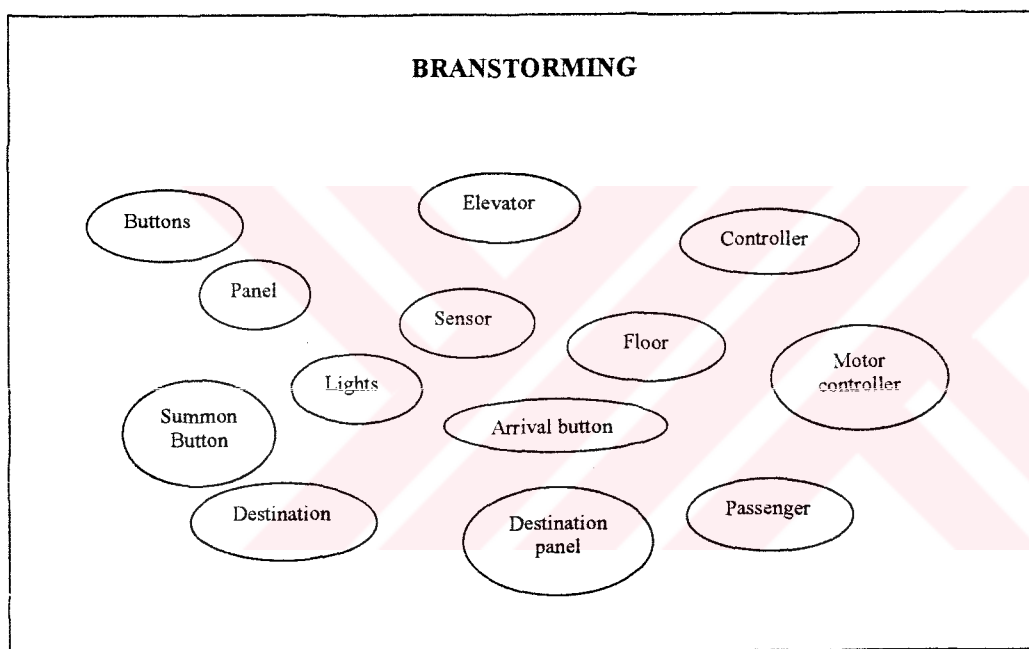


Figure 4.5 Brainstorming

### Step 4: Scoping:

In this step, the data found out in Brainstorming step are transferred to the Scoping areas and boundaries of the problem domain are tried to be defined. Scoping diagrams bring discussions back to the relevant topics and provide a focus for the team.

The team members try to fit the data to three areas in the scope namely: major area, related areas and unrelated areas or let the data lie outside the boundary.

- Major topics that fit within the scope are identified and written inside the biggest circle.
- Related topics are represented inside the small circles.

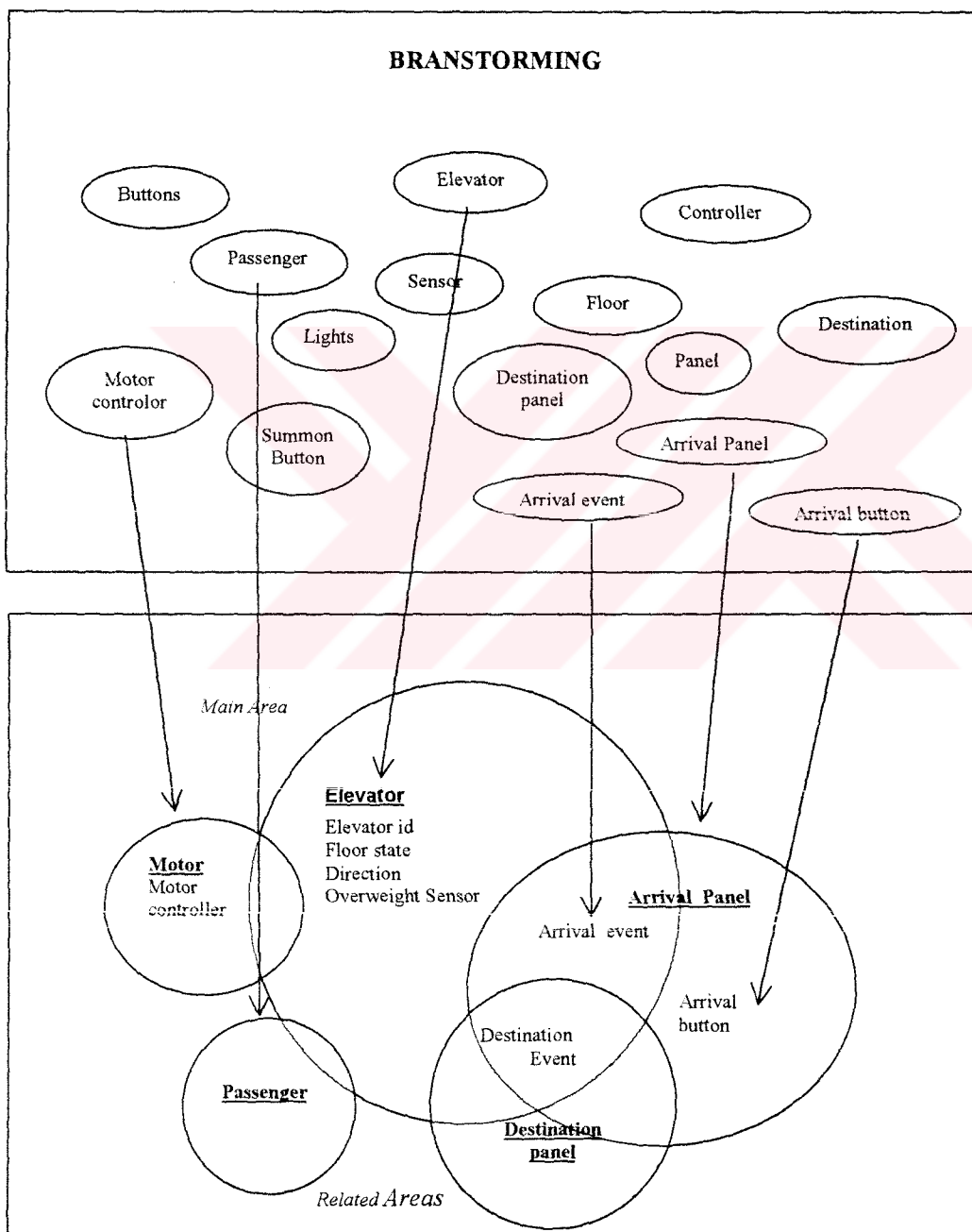


Figure 4.6 Data Transfer from Brainstorming to Scoping phase

### Step 5: Context Diagram.

Data previously chosen from the Scoping diagrams are transferred to the related areas. Transactions between diagrams are shown by labelled unidirectional arrows as between “Floor” and “Elevator”. The “basic elevator process” is described by the steps listed. For example the relationship of the “Elevator” with “Floor” is such as “Elevator Direction” is sent to “Floor” from “Elevator” and as a result “Floor Status” is sent to “Elevator” from “Floor”.

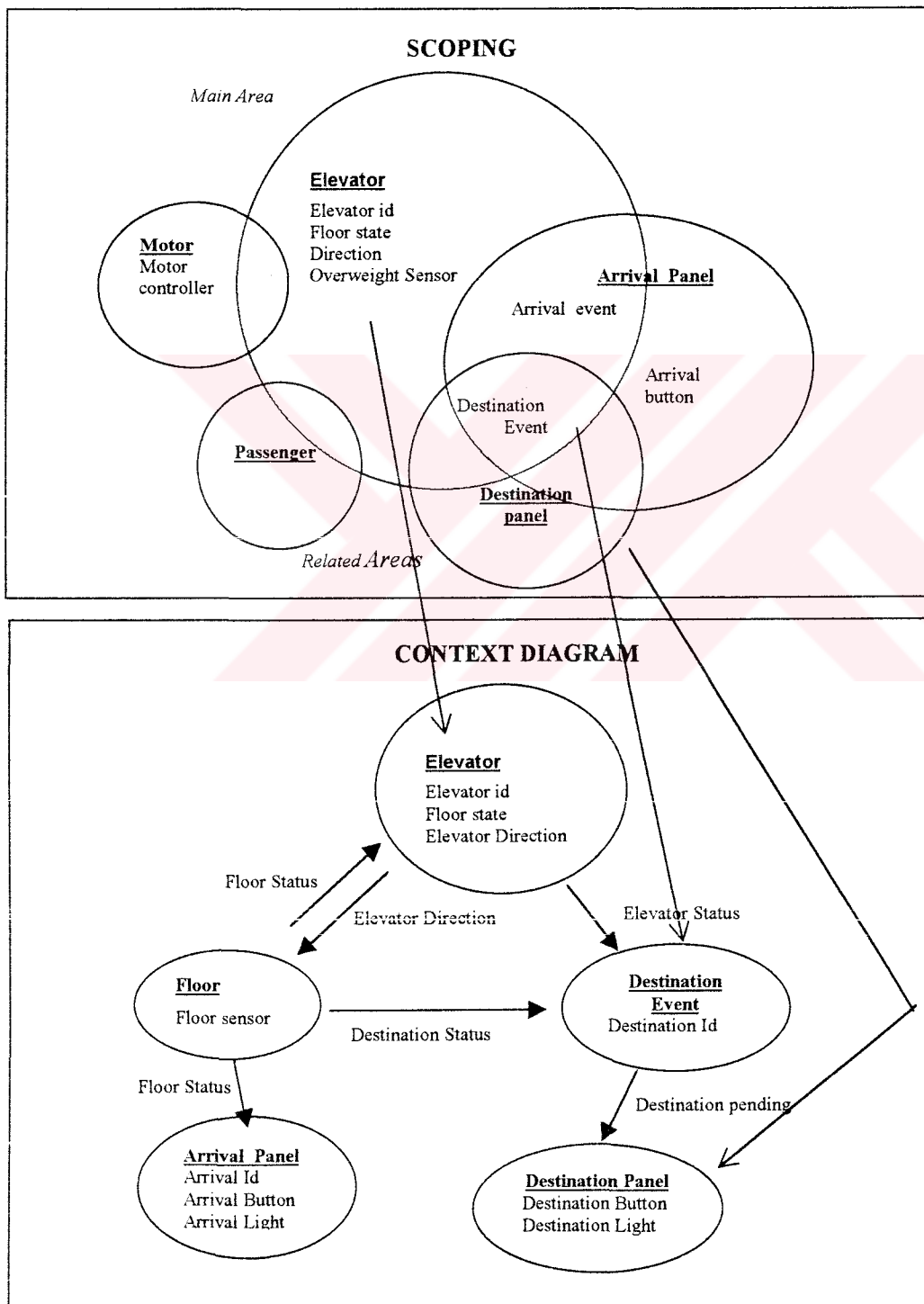


Figure 4.7 Data Transfer from Scoping to Context Diagram Phase

Selection of the entities on the diagrams illustrate the primary subjects of analysis, transactions reflect the team's top considerations for the design. In a way, they interpret the focus of analysis and design. Data transfer among Brainstorming, Scoping and Context Diagram phases are shown in Figure 4.8

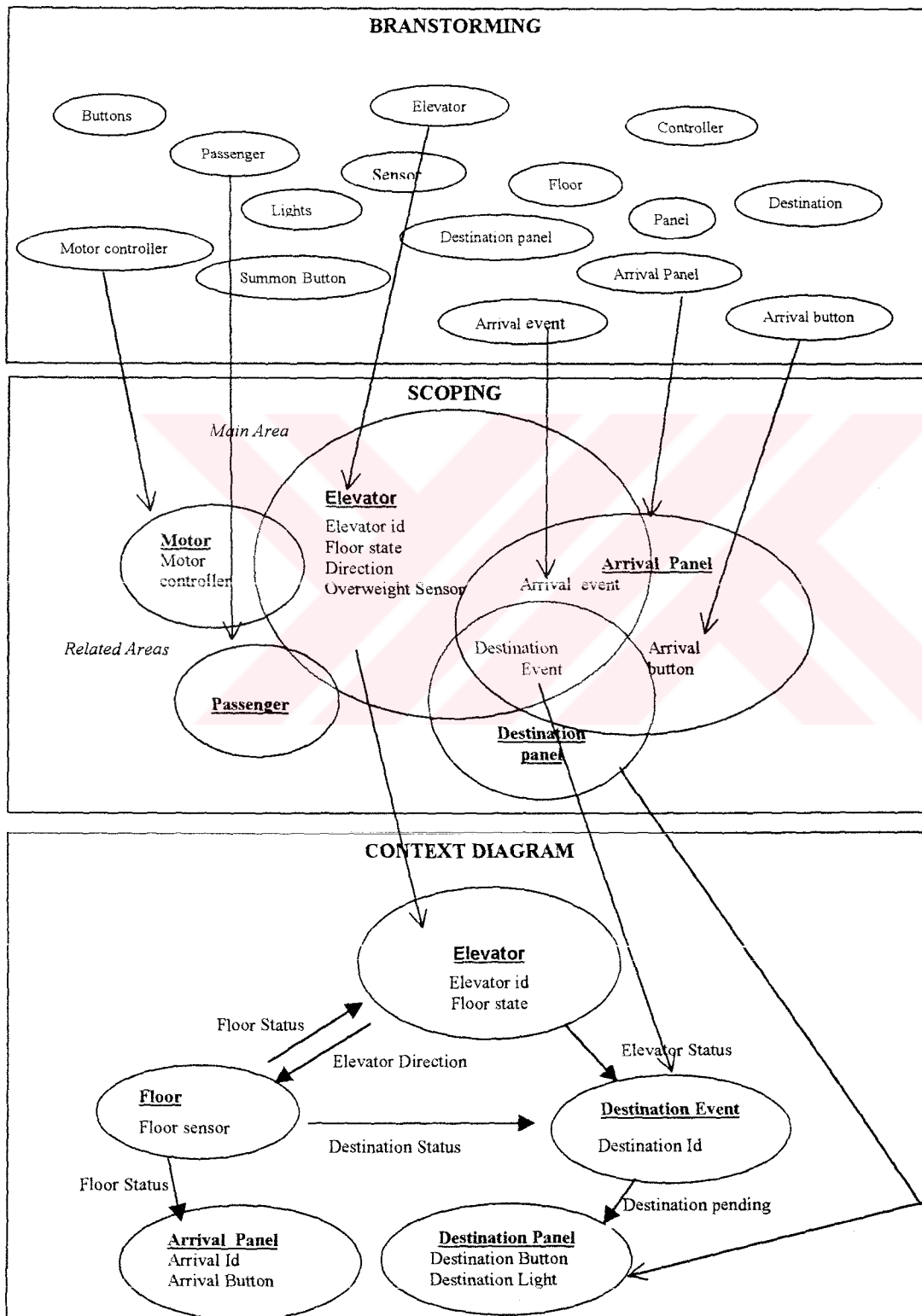


Figure 4.8 Data Transfer among Brainstorming, Scoping and Context Diagram

## 4.6 EARLY DESIGN TOOL (EDT)

EDT has been developed for a team of software program developers so that they can enter information from each workshop meeting and develop early project design. It is modelled [Appendix A] by the graphical representation of the Unified Modelling Language (Booch, 1997) and developed by the Programming Language of Visual Age for Java ver2.0 [Appendix C].

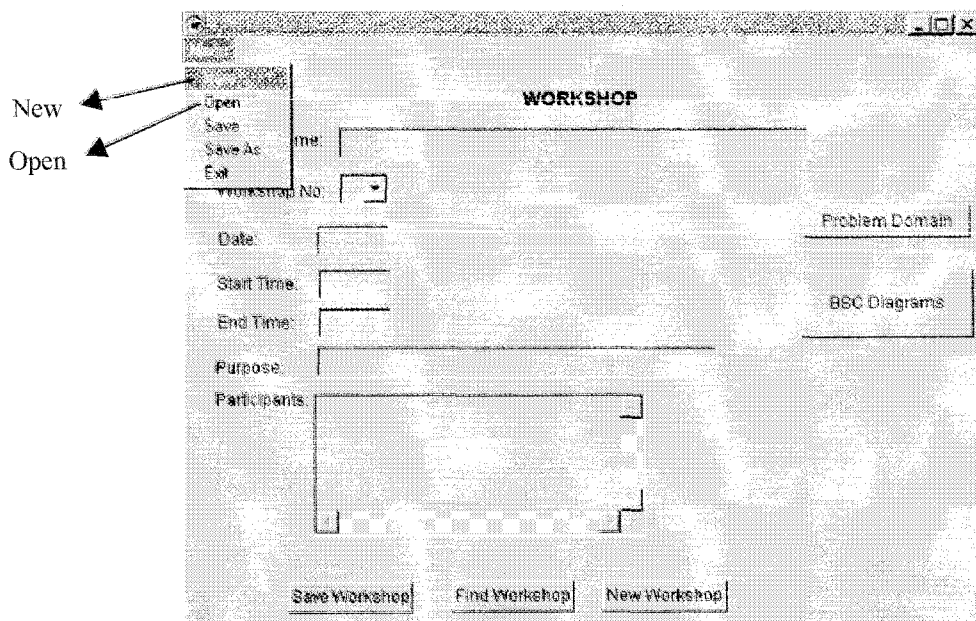
### Step1: Initiating

Let's imagine our team consists of 5 people. The information is entered into the tool by one member. The program is followed on the display projector or monitor projection panel by the other members.

#### Starting the Program

The main window is called "Workshop". Main window is not enabled at the start of the session. A project name must be entered by using the menu items. You can enter data into the main window afterwards.

- If a new project is going to be created, "New" item is selected.
- If an existing project is to be continued, the item "Open" is selected.





### Starting the Workshop

1. To start the Workshop the “New Workshop” right button is clicked the bottom of the screen. When the New workshop button is clicked “Workshop No” will automatically be assigned a value of 1. Then the other workshop information can be entered: *Date*, *Start Time*, *End Time*, *Purpose*, *Participants*. Finally the workshop is saved by the “Save Workshop” button which is the one at the bottom left of the screen.

The screenshot shows a window titled "WORKSHOP". On the left, there is a "Menu" label. The main area contains several input fields: "Project Name" (a text box), "Workshop No:" (a dropdown menu), "Date:" (a text box), "Start Time:" (a text box), "End Time:" (a text box), "Purpose:" (a text box), and "Participants:" (a large text area). On the right side, there are two stacked text boxes labeled "Problem Domain" and "BSC Diagrams". At the bottom, there are three buttons: "Save Workshop" on the left, "Find Workshop" in the middle, and "New Workshop" on the right. Hand-drawn arrows point to the "Workshop No:" dropdown and the "New Workshop" button.

2. If another new workshop is wanted to be start, “New Workshop” button is clicked again. This time “Workshop No” will be assigned a value of 2.
3. To display the existing workshop, the number of the workshop is chosen from the choice box and then the “Find Workshop” button is clicked. The workshop information is then displayed.



## Step2: Problem Domain

1. The Problem domain button is clicked on the Main Window (Workshop) in order to enter the Problem Domain Window. The data is entered by the same way in the Workshop Window. Firstly the “New Domain” button is clicked and then the choice box is assigned a value of 100. (In the Problem Domain Window numbers start with 100). After the data entry is finished, it is saved.

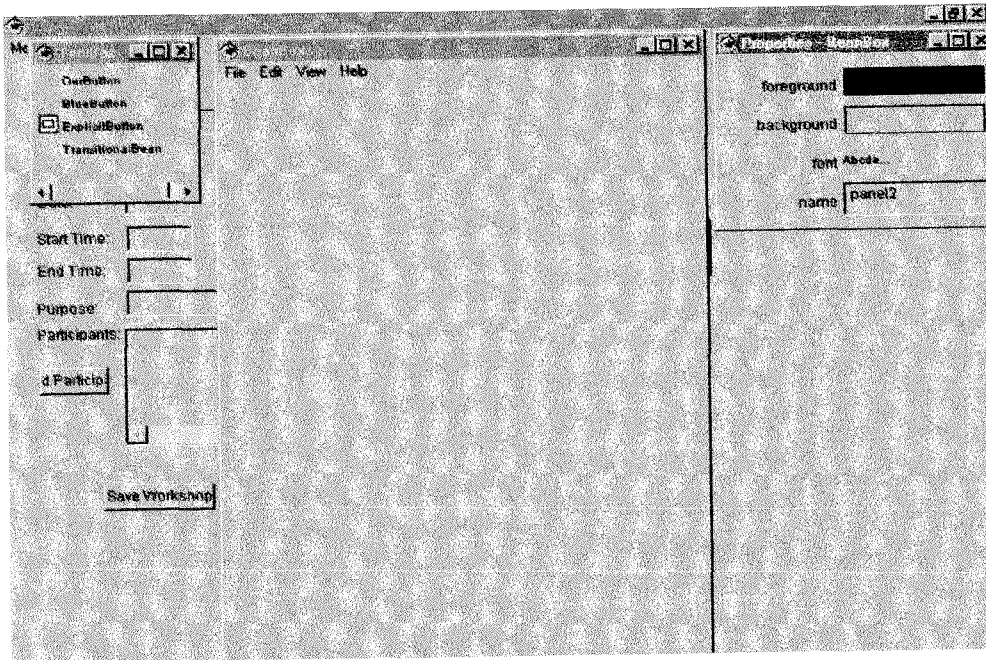
The screenshot shows a window titled "Problem Domain" with the following fields and controls:

- No:** A dropdown menu currently showing "100".
- Name:** A text input field containing the word "Elevator".
- Domain:** A large text area containing the text: "The general requirement is to design and implement a program to schedule and control four elevators in a building with 40 floors." Below the text area is a horizontal scrollbar.
- Buttons:** At the bottom of the window are three buttons: "Save", "Find", and "New Domain".

## Step 3,4,5: Brainstorming, Scoping and Context Diagram Window (BSC Diagrams)

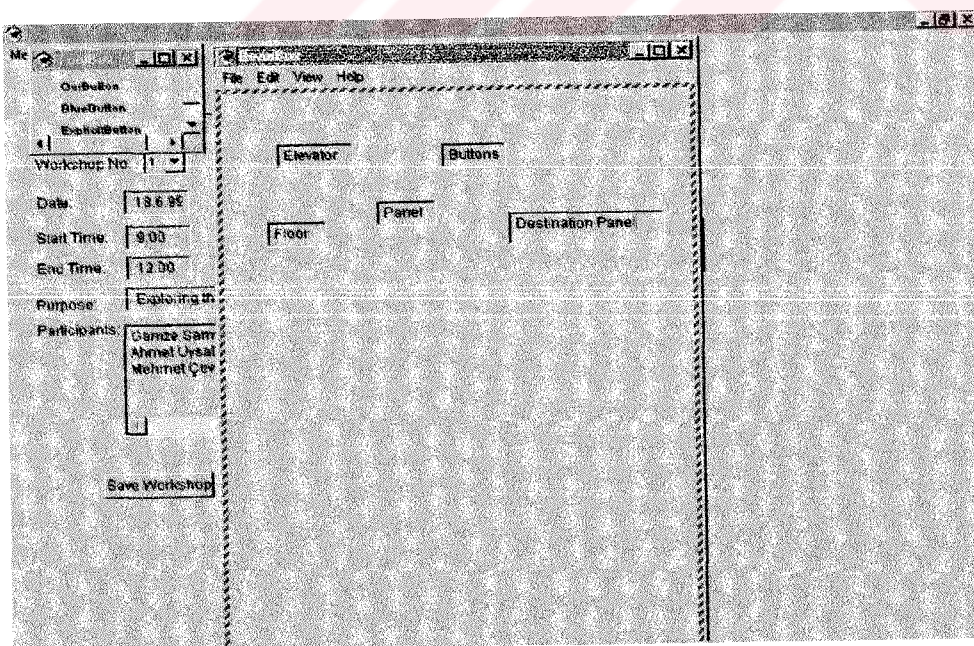
1. The “BSC Diagrams Button” on the Workshop Window is clicked in order to start the “BCS Window”.
2. The beans we would like to drag are chosen from the tool box located on the upper far left corner of your screen.
  - To create a text field - click our button
  - To create a text area - click Explicit button
  - To create arrows – click Transitional Bean





### Step3: Brainstorming

1. Brainstorming bubbles are represented by the text fields in the program.
2. For creating a Text Field, the "Our Button" is clicked on the ToolBox Window and the mouse is clicked once again on the BeanBox Window.
3. The Text Fields can be moved around the BeanBox Window.





### Step 4: Scoping

1. The data in the Brainstorming text fields is transferred to the Scoping text area.
2. In the Scoping phase related and unrelated areas are colour coded. Related boxes are the same colour:

Major area: Red

Related areas: Blue

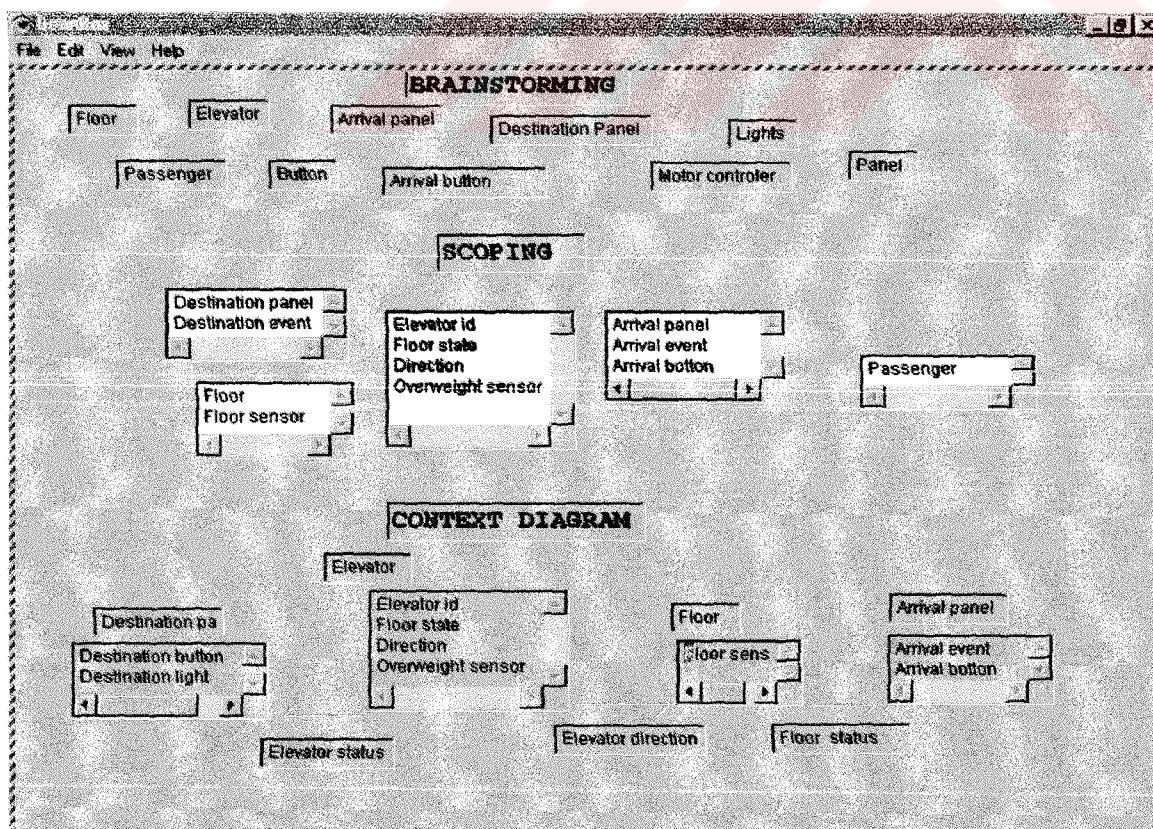
Unrelated areas: Yellow

The colours of the bean are changed on Properties Bean Box Window.

3. Under Scoping, each data written under Brainstorming may be used again in more than one box.

### Step 5: Context Diagram.

1. Entities which are previously chosen from the Scoping diagrams are transferred to the related areas in the Context Diagram phase.
2. In this phase, the data written in the Scoping phase can be transferred into only one box.
3. The text areas in the Context Diagram phase are identical and are the original colour.
4. The Relationships between Context Diagram text areas are represented by arrows.



#### **4.7 Suggestions for Further Research on the Tool EDT**

In this study, we tried to provide an insight into the ways of managing teamwork and developed a support tool on which the teams can develop a software design. The tool offers you a path from Analysis to Design. Final diagrams in our tool EDT illustrate the primary subjects of analysis and the transactions reflect the teams' top consideration for the design. In a way, they interpret the focus of analysis.

Future work of the tool:

- The tool can help to find out classes and objects in Object Oriented Programming Language.
- Transferring the final diagrams of our tool to CASE tools can play a very significant role for the Object Oriented development.
- In most of the Team Design Methods a facilitator is a prerequisite, because they are so complicated that a specialist is needed for application. Our suggestion is to add the roles of the facilitator to the tool. Thus, the tool can guide the teams and the teamworks. As a result, the absolute need for the facilitator is vanished.

---

## CHAPTER FIVE

## CONCLUSION

---

Software development is a complex activity that requires a group of individuals working together effectively as a team. In our study on effective teams and their role in software development, we have identified a number of characteristics that such teams possess. These characteristics differentiate effective teams from the others and cause their individual members to outperform. Effective team features are having a small team size, common approach and goals, a team spirit, and having highly motivated, skilled and committed members working in harmony. To observe the characteristics within real software teams, we have chosen Microsoft Company as a case study.

Achieving effectiveness is a challenging task. As computer systems become more complex and business emphasis is more on quality and productivity, discussions are not documented properly, roles are not clearly defined, forgotten or misunderstood commitments cause conflicts within communication structure and this interrupts healthy teamwork. Even though some teams possess effective team properties, they may fail. When a group of people are brought together important things to be considered are how to engage them to do work as a team, achieve collaboration among them, document decisions and start design (Jones, 1998). Another prerequisite for a successful software product is a good design. A good design increases the quality of the software and reduces the cost by defining the possible problem areas.

For an effective software design, a number of life cycle models have been defined. Common phases in most of the life cycle models are Planning, Requirement Definition, Design and Implementation. However, life cycles do not provide the activities called Early Design Meetings where the team members discuss the project plan, requirements and develop solutions. With these meetings a very useful teamwork atmosphere is created and thus an immediate motivation is accomplished. Members get to know each other and also the roles and the tasks of the members are defined. Therefore we studied existing team design methodologies addressing both social factors and design solutions. These are Joint Application Design (JAD), Rapid Application Development (RAD), Participatory Design (PD) and Team Design (TD).

We observed that today's computer world is not aware of the importance of Early Design Meeting activities and also there is no tool supporting Early Design Meetings. Therefore we decided to develop a tool as a support system for Early Design Meetings. We designed our tool for Early Team Design Meetings according to Team Design (TD) Method. We chose TD method as a base because this method includes JAD and PD. This tool addresses the following advantages:

- Decreasing the loss of information
- Transferring data safely
- Saving time
- Transferring the data into CASE tools easily
- Supporting effective team work by being user friendly
- Lessening the problem of software complexity
- Providing a path to program developers from Analysis to Design.

In this thesis, we tried to provide an insight into the importance of managing teamwork and software design and then introduced the Early Design Tool (EDT). In developing such a tool we aimed firstly to help the teams to get rid of the struggles they have, secondly, to improve the quality of software with the advantages mentioned above.

Future development of EDT will include class and object selection in Object Oriented Programming Languages and the transfer of final diagrams to CASE tools to prevent information loss.



---

## REFERENCES

---

- Aktaş, A.M. (1997). Grup Süreci ve Grup Dinamikleri. İstanbul: Sistem Yayıncılık
- Arthur, L.J. (1992). Rapid Evolutionary Development :Requirements, Prototyping & Software Creation. New York: John Wiley & Sons.
- Bales, R.f. (1950). Interaction Process Analysis: A method for study of smll groups. Reading, MA: Addison-Wesley.
- Bjerknes, G., Ehn, P., & Kyng, M. (1987). Computer and Democracy: A Scandinavian Challenge. Aldershot, Avebury, Great Britain.
- Boehm, B. (1981). Software Engineering Economics. Englewood Cliffs NJ: Prentice-Hall.
- Booch, G. (1994). Object-oriented Analysis and Design with Applications. Redwood City CA: Benjamin Cummings.
- Brooks, F.P. (1987). Essence and Accidents of Software Engineering. IEEE Computer, 20, 10-19.
- Carmel, E., Whitaker, R. D., & George, J.F. (1993). PD and Joint Application Design: A transatlantic Comparison. Communications of ACM, 36, 4
- Cartwright, D., & Zander, A. (1960). Group dynamics: research and theory. (2<sup>nd</sup> ed). Evanston, IL: Row, Peterson and Company.

- Coad, P., & Jourdan, E. (1991). Object Oriented Analysis. Englewood Cliffs NJ: Prentice-Hall.
- Curtis, B. (1990). Managing the real leverage. American Programmer, 3, 4-14.
- Cusumano, M.A., & Selby, R.W. (1995). Microsoft secrets. New York, NY: The Free Press.
- Damian, A., & Hong, D., & Li, H., & Pan, D. (1998). Canada: University of Calgary.
- DeMarco & Lister, (1987). Peopleware: Productive projects and teams. New York, NY: Dorset House.
- Demirörs, E. (1995). A Blackboard framework for supporting teams in software development. PhD Dissertation, Southern Methodist University, Dallas, Tx.
- Demirörs, E., Sarmaşık, G., & Demirörs, O. (1997). The Role of Teamwork in Software Development: Microsoft Case Study. IEEE: Euromicro'97 Conference.
- Ellis, C.A., Gibbs S. J., & Rein G. L. (1991). Groupware: some issues and experiences. Communications of the ACM, 34, 38-58.
- Feld, S.L. (1982). Structural determination of similarity among associates. American Sociological Review. 47. pp:797-801
- Hackman, J. R. (1990). Groups that work (and Those That Don't). San Fransisco, CA: Jossey-Bass Publishers.
- Harkins, S.G., & Szymanski, K. (1987). Social loafing and social facilitation: New wine in old bottles. Review of Personality and Social Psychology: Group Process and intergroup relations. 9, pp:167-188



- Harris, G. B., & Taylor S. (1996). Participatory Design Using the Action Workflow Model. Workshop on Strategies for Collaborative Modelling and Simulation, Conference on Computer-Supported Cooperative Work, Boston, MA.
- Heermann, B. (1997). Building Team Spirit: Activities for Inspiring and Energising Teams. New York: McGraw-Hill.
- Jacobson, I., Cristerson, M., Jonsson, P., & Overgaard, G. (1992). Object-Oriented Software Engineering: A Use Case Driven Approach. New York: Addison-Wesley.
- Jones, P.H. (1998). Handbook of Team Design. New York: McGraw-Hill.
- Katzenbach, J. R., & Smith, D. K. (1993). The Wisdom of teams. Boston, MA: Harvard Business School Press.
- Kerr, N.L. (1986). Motivational choices in task groups: A paradigm for social dilemma research. W.Ike. pp:1-27
- Latene, B. (1981). The psychology of social impact. American Psychologist, 36, 343-356
- McCarthy, J. (1995). Dynamics of software development. Redmond, WA: Microsoft Press.
- Muller, M. J., Wildman, D. M., & White, E. A. (1993). Taxonomy of PD Practices: A Brief Practitioner's Guide. Communications of ACM, 36, 4
- Patterson, M.L., & Schaffer, R.E. (1977). Effects of size and sex composition in interaction distance, participation and satisfaction in small groups. Small Group Behaviour, 8, pp. 433-442
- Patterson, A.H. (1986). Scientific jury selection: The need for a case specific approach. Social Action Law, 11, pp: 433-442

- Perry, E., Votta, L. G. & Staudenmayer, N. (1994, July). People, organizations, and process improvement. IEEE Software: pp.38-44
- Porter, N., Geis, F.L., Cooper E., & Newman, E. (1985). Androny and Leadership in mixed sex groups. Journal of Person Social Psyclogy. 49, pp: 808-823
- Ross, R. S. (1989). Small groups in organizational settings. Englewood Cliffs, NJ: Prentice-Hall.
- Royce, W.W. (1970). Managing the development of large software systems: concepts and techniques. IEEE WESTCON, 9, 1-9.
- Shaw, M. E. (1976). Group dynamics: the psychlogy of small group behavior. (2<sup>nd</sup> ed.). New York: McGraw Hill Inc.
- Scholtes, P. (1988). The Team Handbook. Madison, Wis.: Joiner Associates.
- Sommerville, I. (1995). Software Engineering. (5<sup>th</sup> ed.). Addison - Wesley Publishing Company Inc.
- Stasser, G., Kerr, N.L., & Davis, J.H. (1989). Influence process and consensus models decision-making groups. Paulus, pp: 279-326.
- Trape, Y. (1975). Seeking information about one's own ability as a determinant of choice among tasks. Journal of Personality and Social Psychology. 92, pp:1004-1013
- Tziner, A., & Eden, D. (1985). Does the whole equal to the sum of its parts?. Journal of Applied Psychology, 67, 769-775.
- Yourdon, E. (1996). Rise and resurrection of the American programmer. Upon Saddle River, NJ: Prentice-Hall.
- Wood, J. & Silver, D. (1995). Joint Application Development. New York: John Wiley & sons.

---

**APPENDIX A**

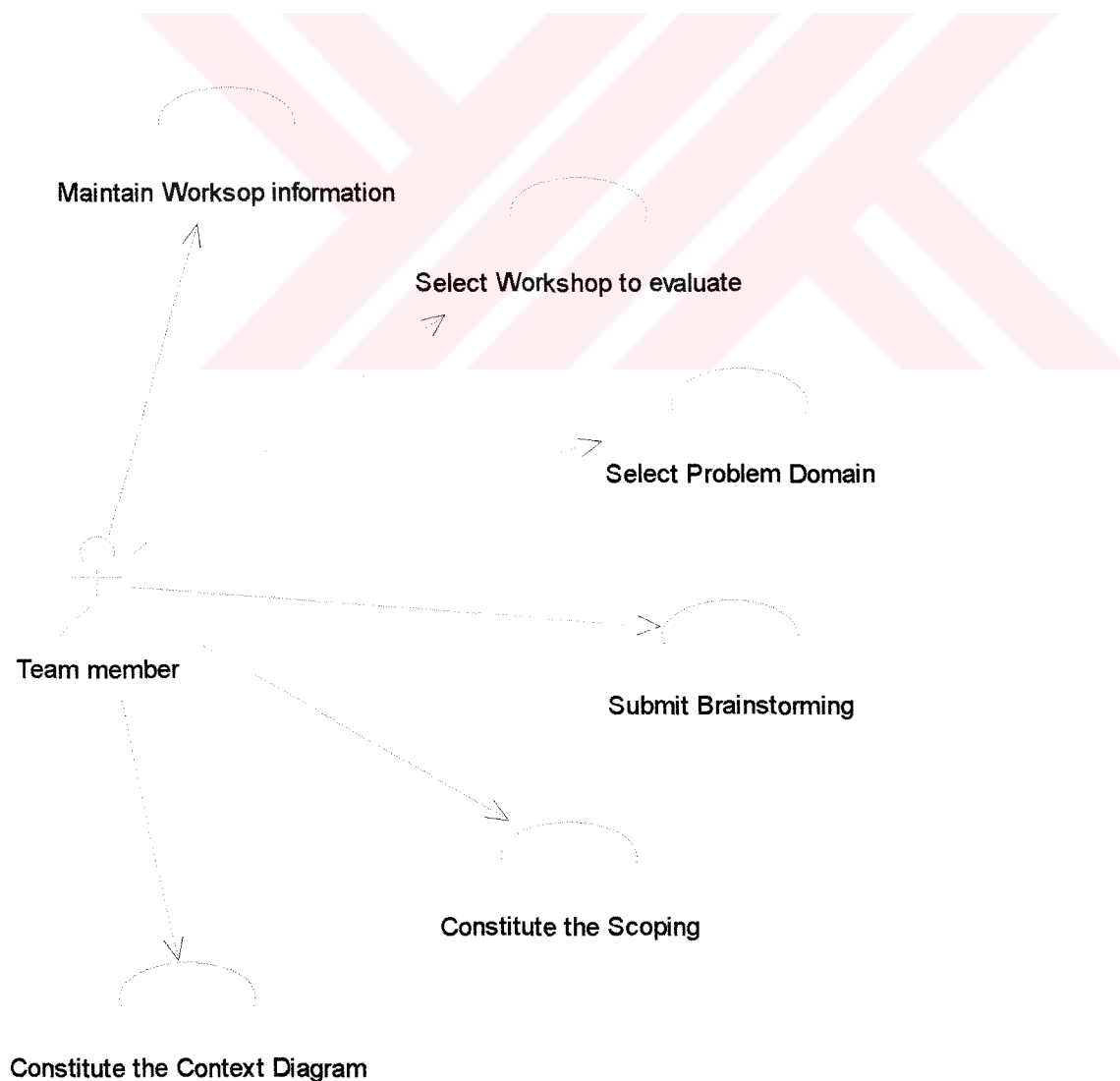
---

**INSTALLING THE “EARLY DESIGN TOOL (EDT)”**

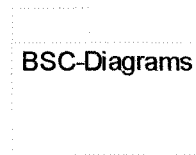
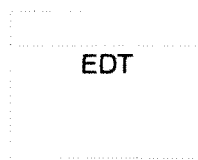
---

**A.1 EDT Use Case Diagrams**

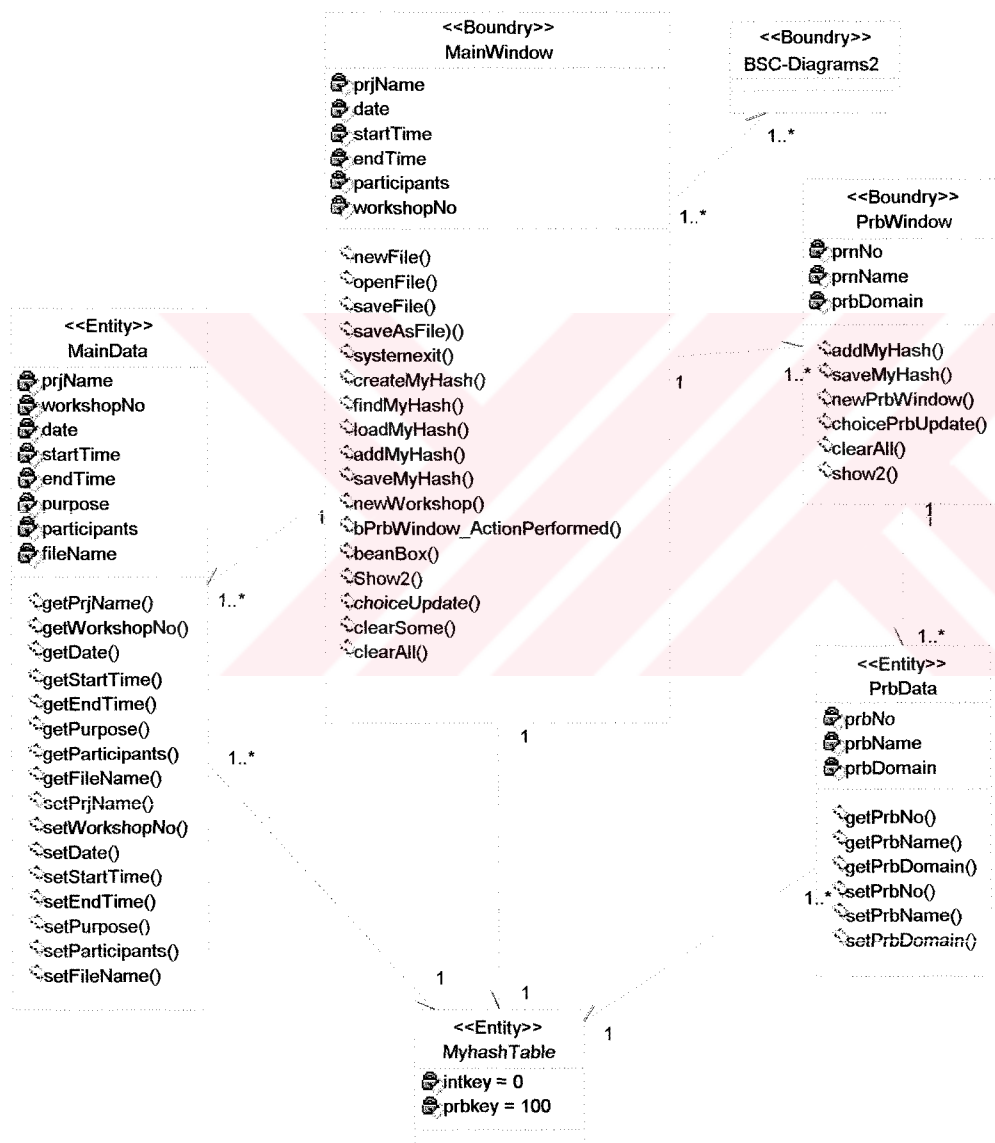
The Tool EDT is designed by “Rose Java 98™ Edition” using the graphical representation of the “Unified Modelling Language” [Booch, 1997].



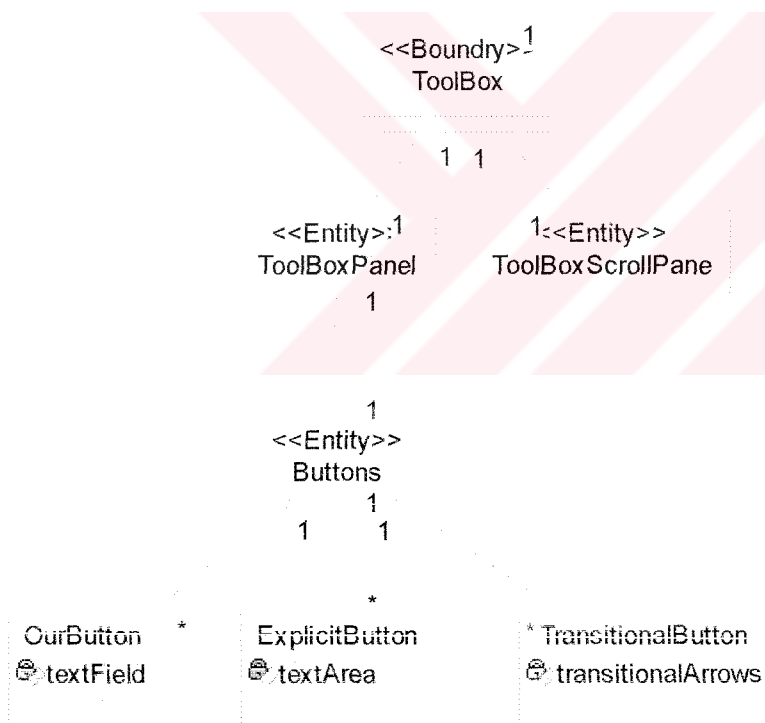
## A.2 EDT Packages



### A.3 EDT Class Diagrams



#### A.4 “BSC-Diagrams” Class Diagrams



---

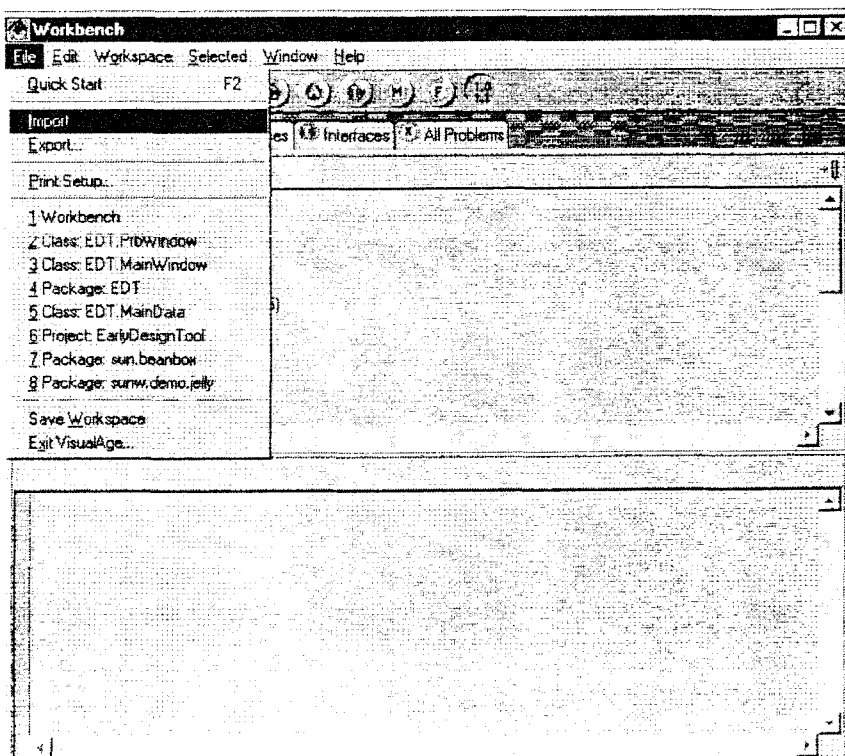
## APPENDIX B

# INSTALLING THE “EARLY DESIGN TOOL (EDT)”

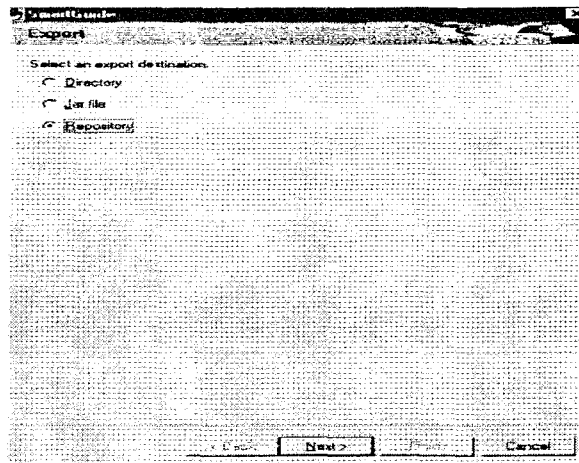
---

### B.1 Installing The “Early Design Tool (EDT)”

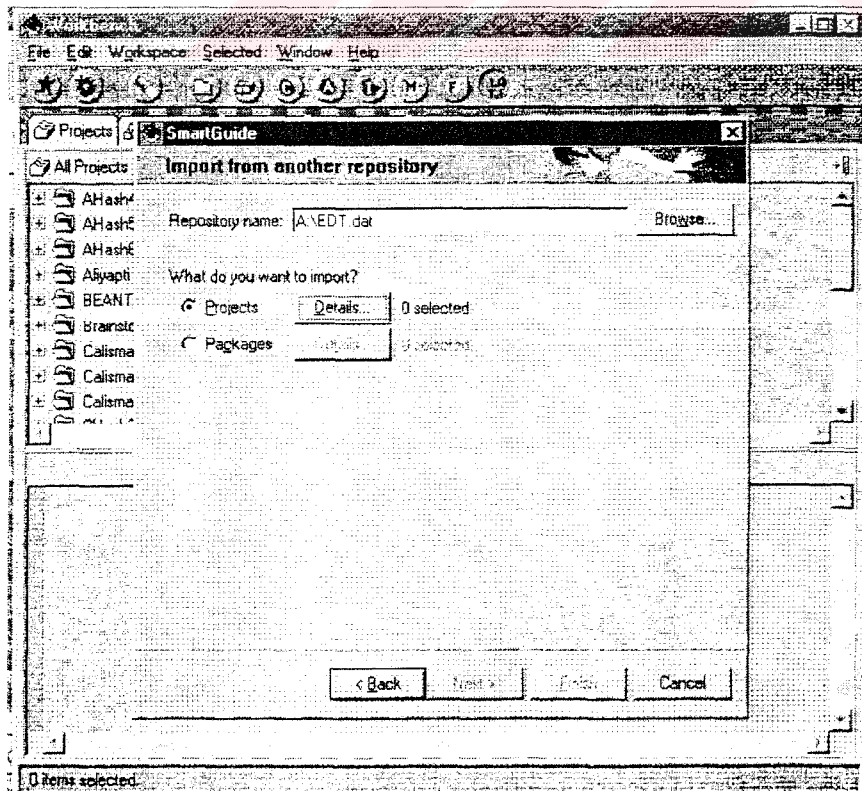
1. Visual Age for Java ver2.0 must primarily be installed.
2. Use the command “Import” from the “File” menu under the Visual Age for Java main window to start the installation of the program “EarlyDesignTool”.



- From the dialog window choose "Repository" then press "Next" to go to the next step.

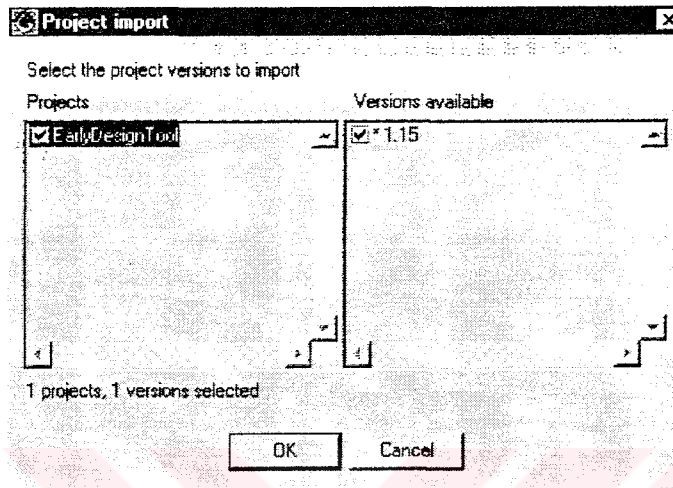


- Write the install file path and name next to "Repository Name:". Choose projects as to what you want to import.

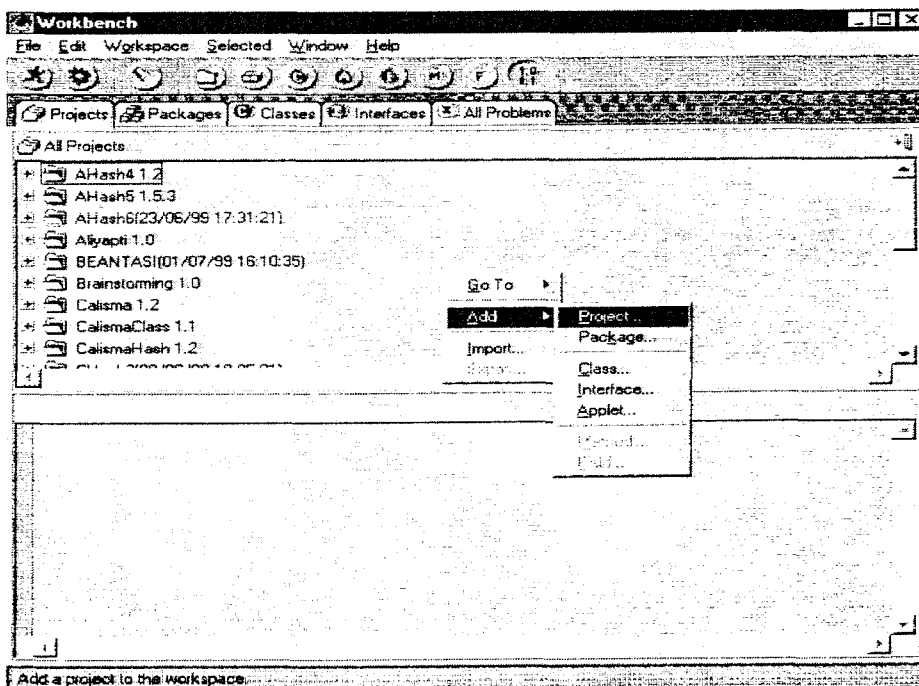




- Click on to “Details” (figure above) to open the “Project import” window (figure below) so that you can select the project and its version. You have now imported the program to the Visual Age For Java repository.

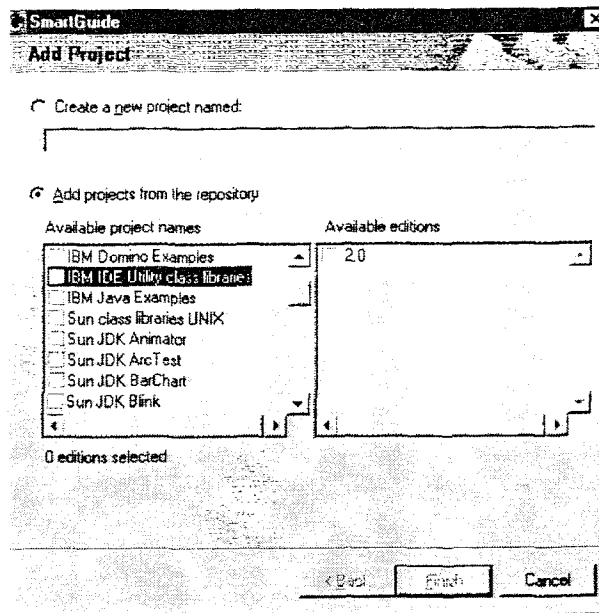


- To be able to see the project under the VisualAge for Java main window you have to install it from the repository. For this click on the right mouse button somewhere empty on the window. From the menu that appears click “add” and then click “project”.

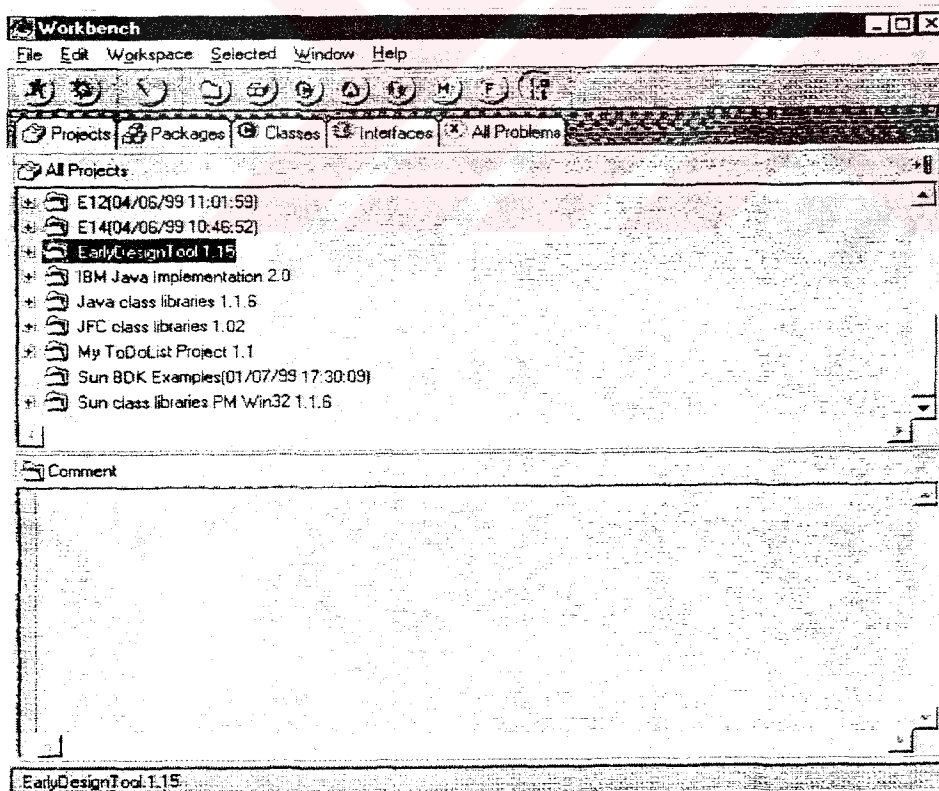


7. From the Add Project Dialog Window select “Add projects from repository”.

From the project list that appears select the program name and version (which for our case is the “Early Design Tool”).

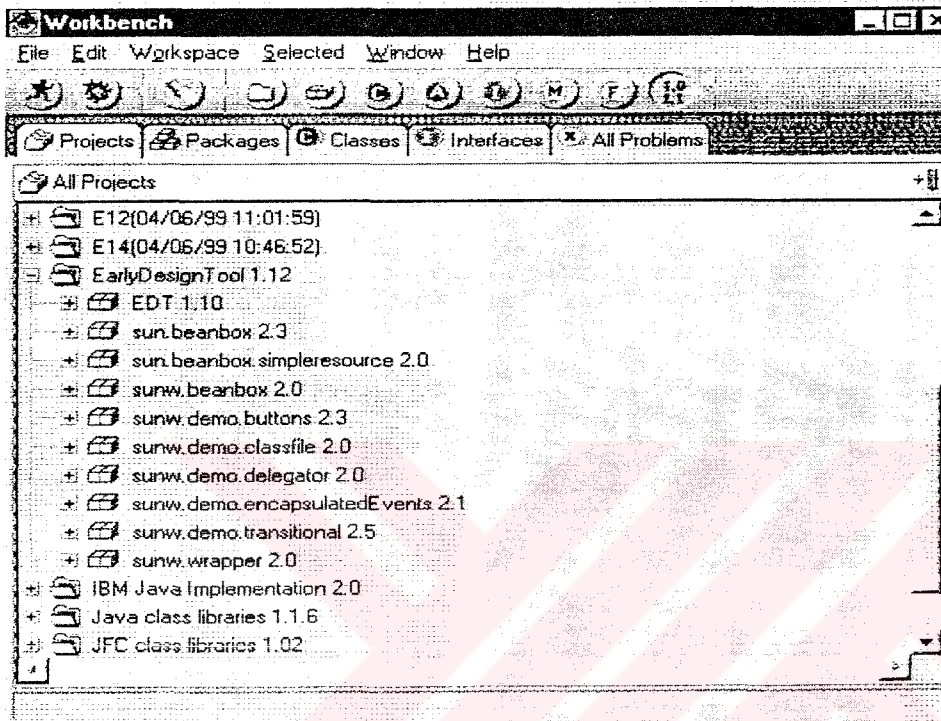


8. The project will be added in alphabetical order to the VisualAge for Java window.

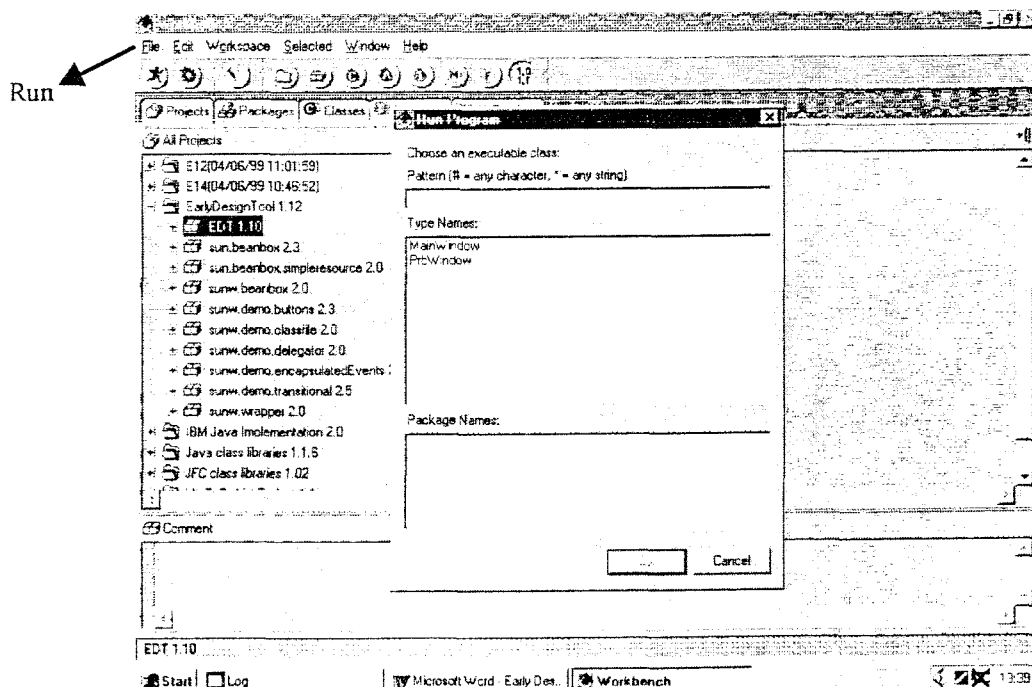


## B.2 Starting the Program

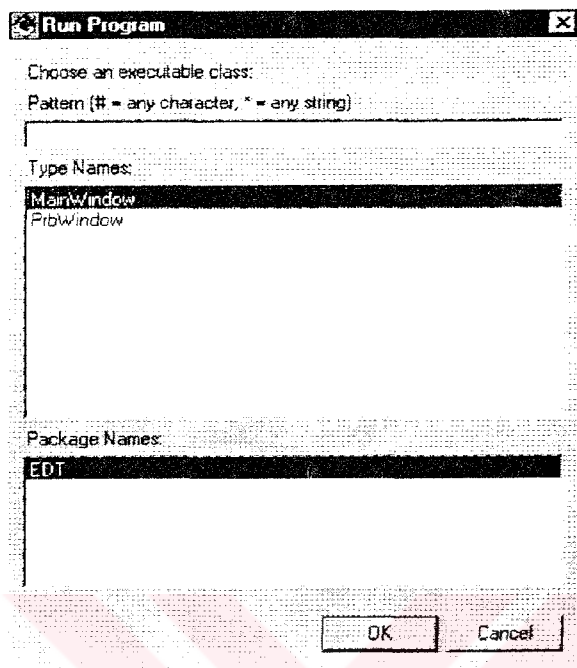
1. From the main window of “Visual Age for Java”, the project of “Early Design Tool” is selected. You see in the below figure the project and related packets.



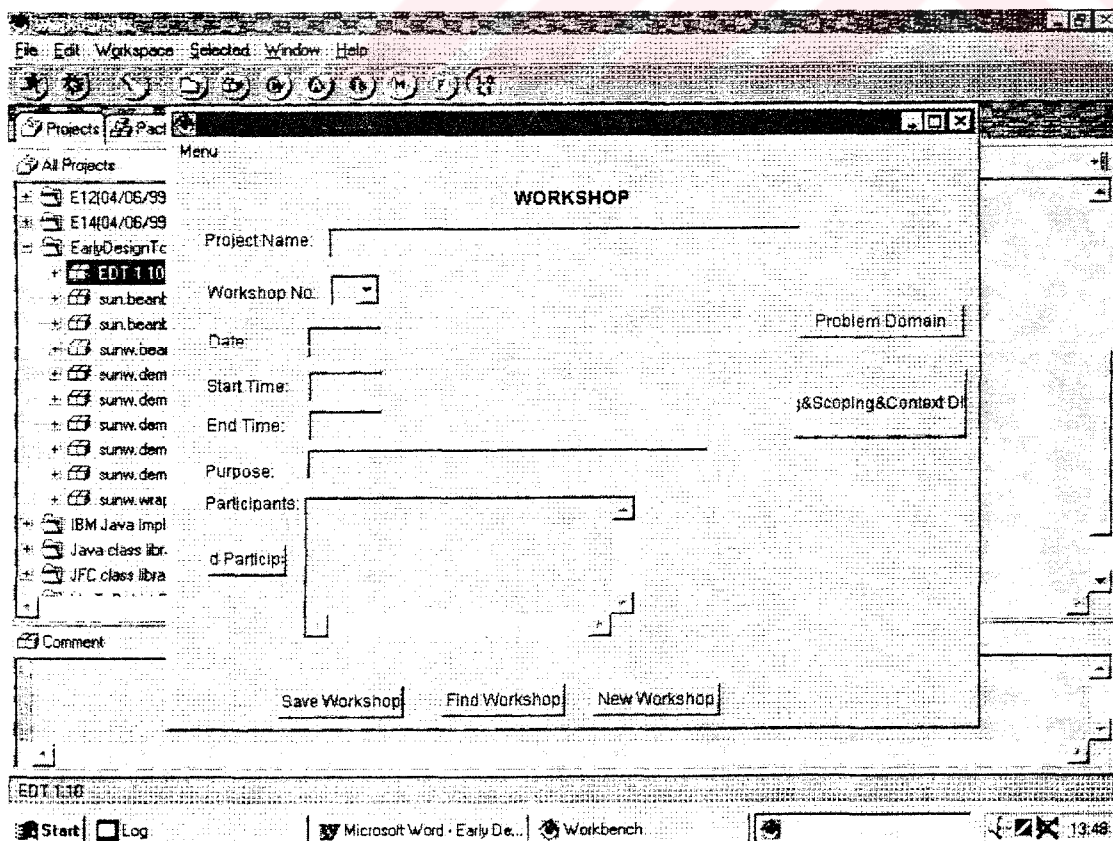
2. The first package EDT is selected from the project menu of the “Early Design Tool”. Then the “Run” command is started from the tool bar. This opens a dialog box.



3. One other alternative is selecting "MainWindow" and clicking on "OK". Which starts the program.



4. The main window of the "Early Design Tool" appears.



---

## APPENDIX C

### EDT PROGRAM CODE

---

This codes created in VisualAge For Java ver2.0.

#### C1. MainWindow

```
import MainData;
import PrbData;
import PrbWindow;
import sun.*;
import sunw.*;

public class MainWindow extends java.awt.Frame implements java.awt.event.ActionListener,
java.awt.event.ItemListener, java.awt.event.WindowListener {
    private java.awt.Panel ivjContentsPane = null;
    private java.awt.Label ivjLDate = null;
    private java.awt.Label ivjLEndTime = null;
    private java.awt.Label ivjLPrjName = null;
    private java.awt.Label ivjLPurpose = null;
    private java.awt.Label ivjLStartTime = null;
    private java.awt.Label ivjLTitle = null;
    private java.awt.Label ivjLWorkshopNo = null;
    private java.awt.Label ivjLParticipants = null;
    private java.awt.Choice ivjCWorkshopNo = null;
    private java.awt.TextArea ivjTAParticipants = null;
    private java.awt.TextField ivjTFDate = null;
    private java.awt.TextField ivjTFEndTime = null;
    private java.awt.TextField ivjTFPrjName = null;
    private java.awt.TextField ivjTFPurpose = null;
    private java.awt.TextField ivjTFStartTime = null;
    private java.awt.MenuBar ivjMainWindowMenuBar = null;
    private java.awt.Menu ivjMenu = null;
    private java.awt.MenuItem ivjMIExit = null;
    private java.awt.MenuItem ivjMINew = null;
    private java.awt.MenuItem ivjMIOpen = null;
    private java.awt.MenuItem ivjMISave = null;
    private java.awt.MenuItem ivjMISaveAs = null;
    private java.awt.Button ivjBBean = null;
    private java.awt.Button ivjBPrbDomain = null;
    private String fileName=null;
    MyHashtable aMyHashtable = null;
    private MainData aMainData=null;
    private java.awt.Button ivjBAddWorkshop = null;
    private java.awt.Button ivjBFindWorkshop = null;
    private java.awt.Button ivjBNewWorkshop = null;
    private int intkey;
    private int prbkey;
}
```

```

public void newFile() {
    java.awt.FileDialog dialog = new java.awt.FileDialog(this, "Save...",
    java.awt.FileDialog.SAVE);

    dialog.show();

    fileName = dialog.getDirectory()+dialog.getFile();
    String file=dialog.getFile();
    dialog.dispose();

    if (file != null) {

        saveMyHash(fileName);
        getMISave().setEnabled(true);
        getMISaveAs().setEnabled(true);
        getContentsPane().setEnabled(true);

        getCWorkshopNo().removeAll();
        clearAll();
        intkey=0;
        prbkey=100;

    }

}

```

```

public void openFile() {
    java.awt.FileDialog dialog = new java.awt.FileDialog(this, "Save...",
    java.awt.FileDialog.SAVE);

    dialog.show();

    fileName = dialog.getDirectory()+dialog.getFile();
    String file=dialog.getFile();
    dialog.dispose();

    if (file != null) {
        getCWorkshopNo().removeAll();
        loadMyHash(fileName);

        getMISave().setEnabled(true);
        getMISaveAs().setEnabled(true);
        getContentsPane().setEnabled(true);

        clearAll();

    }

}

```



```

public void saveFile() {

    saveMyHash(fileName);

}

public void saveAsFile() {
    java.awt.FileDialog dialog = new java.awt.FileDialog(this, "Save...",
    java.awt.FileDialog.SAVE);

    dialog.show();

    String fileName = dialog.getDirectory()+dialog.getFile();
    String file = dialog.getFile();
    dialog.dispose();

    if (file != null) {
        saveMyHash(fileName);
    }

    return;
}

```

```

public void systemexit() {
    System.exit(0);
    return;
}

```

```

public void createMyHash() {
    aMyHashtable = new MyHashtable();
}

```

```

public void findMyHash() {

    String userkey=getCWorkshopNo().getSelectedItem();
    aMainData=(MainData)aMyHashtable.get(userkey);

    getTFPrjName().setText(aMainData.getPrjName());
    getTFDate().setText(aMainData.getDate());
    getTFStartTime().setText(aMainData.getStartTime());
    getTFEndTime().setText(aMainData.getEndTime());
    getTFPurpose().setText(aMainData.getPurpose());

    getTAParticipants().setText(aMainData.getParticipants());

}

```

```

public void loadMyHash(String fileName) {
    try {
        java.io.ObjectInputStream in = new java.io.ObjectInputStream(new
        java.io.FileInputStream(fileName));
        aMyHashtable=(MyHashtable)in.readObject();
        this.intkey=aMyHashtable.intkey;
        choiceUpdate(this.intkey);
        this.prbkey=aMyHashtable.prbkey;

    }catch(Exception e){
        e.printStackTrace();
    }
}

```

```

public void addMyHash(String fileName) {

    MainData aMainData=new MainData();

    aMainData.setPrjName(getTFPrjName().getText());
    aMainData.setWorkshopNo(getCWorkshopNo().getSelectedItem());
    aMainData.setDate(getTFDate().getText());
    aMainData.setStartTime(getTFStartTime().getText());
    aMainData.setEndTime(getTFEndTime().getText());
    aMainData.setEndTime(getTFEndTime().getText());
    aMainData.setPurpose(getTFPurpose().getText());
    aMainData.setParticipants(getTAParticipants().getText());

    String userkey=getCWorkshopNo().getSelectedItem();
    aMyHashtable.put(userkey,aMainData);

    saveMyHash(this.fileName);
}

```

```

public void saveMyHash(String fileName) {
    aMyHashtable.intkey=this.intkey;
    try {
        java.io.ObjectOutputStream out = new java.io.ObjectOutputStream(new
        java.io.FileOutputStream(fileName));
        out.writeObject(aMyHashtable);
        out.close();
    }catch(Exception e){
        e.printStackTrace();
    }
}

```



```

public void newWorkshop() {
    intkey++;
    getCWorkshopNo().addItem(new String().valueOf(intkey));
    clearSome();
}

```

```

public void bPrbWindow_ActionPerformed(java.awt.event.ActionEvent actionEvent) {

    PrbWindow PrbWindow1 = new PrbWindow();
    PrbWindow1.show2(aMyHashtable,fileName);

    return;
}

```

```

public void beanBox() {
    sun.beanbox.BeanBoxFrame BeanBoxFrame1 =new sun.beanbox.BeanBoxFrame();
    String[] argv={" "};
    BeanBoxFrame1.main(argv);
}

```

```

public void show2(Object p_hash, String fileName) {
    aMyHashtable = (MyHashtable) p_hash;
    this.fileName=fileName;
    show();
}

```

```

public void choiceUpdate(int intkey) {
    for (int i=1;i<=intkey;i++){
        getCWorkshopNo().addItem(new String().valueOf(i));
    }
}

```

```
public void clearSome() {  
    getTFDate().setText(" ");  
    getTFStartTime().setText(" ");  
    getTFEndTime().setText(" ");  
    getTFPurpose().setText(" ");  
    getTAParticipants().setText(" ");  
}
```

```
public void clearAll() {  
    getTFPrjName().setText(" ");  
    getTFDate().setText(" ");  
    getTFStartTime().setText(" ");  
    getTFEndTime().setText(" ");  
    getTFPurpose().setText(" ");  
  
    getTAParticipants().setText(" ");  
}
```

## C2. MainData

```
import java.util.Hashtable;
class MainData implements java.io.Serializable {
    private String prjName;
    private String workshopNo;
    private String date;
    private String startTime;
    private String endTime;
    private String purpose;
    private String participants;
    private String fileName;
}

public String getPrjName() {
    return prjName;
}

public String getWorkshopNo() {
    return workshopNo;
}

public String getDate() {
    return date;
}

public String getStartTime() {
    return startTime;
}

public String getEndTime() {
    return endTime;
}

public String getPurpose() {
    return purpose;
}

public String getParticipants() {
    return participants;
}

public String getFileName() {
    return fileName;
}

public void setPrjName(String newValue) {
    this.prjName = newValue;
}

public void setWorkshopNo(String newValue) {
    this.workshopNo = newValue;
}
```

```
public void setDate(String newValue) {  
    this.date = newValue;  
}  
  
public void setStartTime(String newValue) {  
    this.startTime = newValue;  
}  
  
public void setPurpose(String newValue) {  
    this.purpose = newValue;  
}  
  
public void setParticipants(String newValue) {  
    this.participants = newValue;  
}  
  
public void setFileName(String newValue) {  
    this.fileName = newValue;  
}
```



### C3. PrbWindow

```

import PrbData;
public class PrbWindow extends java.awt.Frame implements java.awt.event.ActionListener,
java.awt.event.ContainerListener, java.awt.event.WindowListener {
    private java.awt.Panel ivjContentsPane = null;
    private java.awt.Label ivjLPrbDomain = null;
    private java.awt.Label ivjLPrbName = null;
    private java.awt.Label ivjLPrbNo = null;
    private java.awt.TextArea ivjTAPrbDomain = null;
    private java.awt.TextField ivjTFPrbName = null;
    MyHashtable aMyHashtable = null;
    private PrbData aPrbData=null;
    private java.awt.Button ivjBAdd = null;
    private java.awt.Button ivjBFind = null;
    private String fileName=null;
    private int prbkey;
    private java.awt.Choice ivjCPrbNo = null;
    private java.awt.Button ivjButton1 = null;
}

public void addMyHash(String fileName) {

    PrbData aPrbData=new PrbData();
    aPrbData.setPrbNo(getCPrbNo().getSelectedItem());
    aPrbData.setPrbName(getTFPrbName().getText());
    aPrbData.setPrbDomain(getTAPrbDomain().getText());


    String userkey=getCPrbNo().getSelectedItem();
    aMyHashtable.put(userkey,aPrbData);
    saveMyHash(this.fileName);
}

public void saveMyHash(String fileName) {
    aMyHashtable.prbkey=this.prbkey;
    try {
        java.io.ObjectOutputStream out = new java.io.ObjectOutputStream(new
java.io.FileOutputStream(fileName));
        out.writeObject(aMyHashtable);
        out.close();
    }catch(Exception e){
        e.printStackTrace();
    }
}

public void newPrbWindow() {
    prbkey++;
    getCPrbNo().addItem(new String().valueOf(prbkey));
    clearAll();
}

```

```
public void choicePrbUpdate(int prbkey) {  
    for (int i=100;i<=prbkey;i++){  
        getCPrbNo().addItem(new String().valueOf(i));  
    }  
}  
  
public void clearAll() {  
  
    getTFPrbName().setText(" ");  
    getTAPrbDomain().setText(" ");  
  
}  
  
public void show2(Object p_hash, String fileName) {  
  
    aMyHashtable = (MyHashtable) p_hash;  
    this.fileName=fileName;  
    this.prbkey=aMyHashtable.prbkey;  
  
    choicePrbUpdate(this.prbkey);  
    show();  
  
}
```



#### C4. PrbData

```
class PrbData implements java.io.Serializable {
```

```
    private String prbNo;  
    private String prbName;  
    private String prbDomain;  
}
```

```
public String getPrbNo() {  
    return prbNo;  
}
```

```
public String getPrbName() {  
    return prbName;  
}
```

```
public String getPrbDomain() {  
    return prbDomain;  
}
```

```
public void setPrbNo(String newValue) {  
    this.prbNo = newValue;  
}
```

```
public void setPrbName(String newValue) {  
    this.prbName = newValue;  
}
```

```
public void setPrbDomain(String newValue) {  
    this.prbDomain = newValue;  
}
```



## C5. BSC- Diagrams

```

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.beans.*;

public class BeanBoxFrame extends Frame implements LayoutManager, Runnable, ActionListener {

    private static String tmpDir = "tmp";
    private static boolean doShowTimes = false;
    private static Thread focusThread;
    private static Component nextFocus;
    private static BeanBoxFrame instance;
    private static String clipDir = "tmp";
    private static String clipFile = "beanBoxClip.ser";
    private static String clipResource = "beanBoxClip";
    private static String versionID = "BDK1.0 - March 1998";
    private static String clipLabel;
    private static String clipName;
    private static boolean clipFromPrototypeInfo;
    private static boolean quickStart = false;
    private static boolean defineOnDemand = true;

    private static BeanBox topBox;
    private static Wrapper topWrapper;

    private static Wrapper currentFocus;

    private static PropertySheet propertySheet;
    private static ToolBox toolBox;

    private static boolean hideInvisibleBeans;
}

```

---

```

import java.beans.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.Vector;

class ToolBoxPanel extends Panel implements Runnable, MouseListener {

    private class Helper implements DoOnBean {
        public void action(JarInfo ji, BeanInfo bi, String beanName) {
            String label;
            Image img = null;

            BeanDescriptor bd = bi.getBeanDescriptor();

```

```

debug("Helper.action; beanName: "+beanName);
debug(" beanName: "+beanName);
debug(" getName(): "+bd.getName());
debug(" getBeanClass(): "+bd.getBeanClass());

Class beanClass = bd.getBeanClass();
if (beanName.equals(beanClass.getName())) {
    label = bi.getBeanDescriptor().getDisplayName();
    img = bi.getIcon(BeanInfo.ICON_COLOR_16x16);
} else {
    label = beanName;
    int ix = beanName.lastIndexOf('.');
    if (ix >= 0) {
        label = beanName.substring(ix+1);
    }
    img = null;
}
addWithUniqueName(beanLabels, label);
beanNames.addElement(beanName);
beanIcons.addElement(img);
beanJars.addElement(ji);
}

public void error(String msg) {
    ToolBoxPanel.this.error(msg);
}

public void error(String msg, Exception ex) {
    ToolBoxPanel.this.error(msg, ex);
}
}

private Helper helper = new Helper();

Vector beanLabels = new Vector();
Vector beanNames = new Vector();
Vector beanIcons = new Vector();
Vector beanJars = new Vector();
private int topPad = 0;
private int sidePad = 0;
private final static int rowHeight = 20;
private Thread insertThread = null;
private static boolean debug = false;
private Object pendingBean;
private String pendingBeanLabel;
private String pendingBeanName;
private boolean pendingFromPrototypeInfo;
private Frame frame;

private static Cursor crosshairCursor = Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR);
private static Cursor defaultCursor = Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR);
}

```

---

```

import java.io.*;
import java.util.*;
import java.beans.BeanInfo;
import java.beans.BeanDescriptor;
import java.beans.Introspector;
import java.beans.PropertyEditor;
import java.beans.PropertyEditorManager;
import java.beans.PropertyDescriptor;
import java.awt.*;
import java.lang.reflect.Method;

```

```

// Abstracts a bean property
class BeanProperty {

```

```

    private boolean        propertyHasChanged; //true if property has changed
    private PropertyDescriptor pd;
    private Method         readMethod;    // getter
    private Method         writeMethod;   // setter
    private Class          propertyType;
    private PropertyEditor propertyEditor; // editor for property type
    private Object          propertyValue;
    private Object         bean; // bean that owns this property

```

```

}

```

---

```

import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.Serializable;
import java.util.Vector;

```

```

/**
 * A simple Java Beans button. OurButton is a "from-scratch"
 * lightweight AWT component. It's a good example of how to
 * implement bound properties and support for event listeners.
 *
 * Parts of the source are derived from sun.awt.tiny.TinyButtonPeer.
 */

```

```

public class OurButton extends TextField implements Serializable,
    MouseListener, MouseMotionListener {

```

```

    private boolean debug;
    private PropertyChangeSupport changes = new PropertyChangeSupport(this);
    private Vector pushListeners = new Vector();
    private String label;
    private boolean down;
    private boolean sized;

```

```

    static final int TEXT_XPAD = 12;
    static final int TEXT_YPAD = 8;

```

```

}

```

---

---

```
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.Serializable;
import java.util.Vector;
```

```
public class OurButton extends TextField implements Serializable,
    MouseListener, MouseMotionListener {

    private boolean debug;
    private PropertyChangeSupport changes = new PropertyChangeSupport(this);
    private Vector pushListeners = new Vector();
    private String label;
    private boolean down;
    private boolean sized;

    static final int TEXT_XPAD = 12;
    static final int TEXT_YPAD = 8;
}
```

---

```
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.Serializable;
import java.util.Vector;
```

```
public class OurButtonTextArea extends TextArea implements Serializable,
    MouseListener, MouseMotionListener {

    private boolean debug;
    private PropertyChangeSupport changes = new PropertyChangeSupport(this);
    private Vector pushListeners = new Vector();
    private String label;
    private boolean down;
    private boolean sized;

    static final int TEXT_XPAD = 12;
    static final int TEXT_YPAD = 8;
}
```

---

---

```

import java.awt.*;

public class TransitionalBean extends Canvas implements sunw.io.Serializable {

    private Color ourColor = Color.orange;
}

public boolean handleEvent(Event evt) {
    if (evt.id == Event.MOUSE_UP) {
        if (ourColor == Color.orange) {
            setColor(Color.green);
        } else {
            if (ourColor == Color.green) {
                setColor(Color.red);
            } else {
                if (ourColor == Color.red) {
                    setColor(Color.blue);
                } else {
                    if (ourColor == Color.blue) {
                        setColor(Color.orange);
                    }
                }
            }
        }
    }
}

return false;
}

```

---

```

public void paint(Graphics g) {
    g.setColor(ourColor);

    if (ourColor == Color.orange) {
        //Sola Ok
        g.drawLine(20,10,50,50);
        //Alt uc
        g.drawLine(50,40,50,50);
        g.drawLine(50,50,40,50);
        // Ust Uc
        g.drawLine(20,10,20,20);
        g.drawLine(20,10,30,10);

        } else {
    if (ourColor == Color.green) {
        // Saga ok
        g.drawLine(20,60,60,20);
        //Alt Uc
        g.drawLine(50,20,60,20);
        g.drawLine(60,20,60,30);
        //Ust Uc
        g.drawLine(20,50,20,60);
        g.drawLine(20,60,30,60);

        } else {
    if (ourColor == Color.red) {

        //Iki yana ok

```

```

        g.drawLine(20,10,100,10);

        g.drawLine(20,10,30,20);
        g.drawLine(20,10,30,1);

        g.drawLine(100,10,90,20);
        g.drawLine(100,10,90,1);

    }else{
if (ourColor == Color.blue){

    // Yukari-Asagi Ok
    g.drawLine(20,10,20,100);
    //      Ust Uc
    g.drawLine(20,10,10,20);
    g.drawLine(20,10,30,20);
    // Alt Uc
    g.drawLine(20,100,10,90);
    g.drawLine(20,100,30,90);

        }

    }

}
}

```

---

```

// Main method for our internal MenuItem handling thread.

public void run() {
for (;;) {
    // Wait for an event.
    ActionEvent evt;
    synchronized (this) {
        while (events.size() == 0) {
            try {
                wait();
            } catch (InterruptedException ix) {
            }
        }
        evt = (ActionEvent) events.elementAt(0);
        events.removeElementAt(0);
    }
    // now process the event.
    try {
        doMenuItem(evt);
    } catch (Throwable ex) {
        System.err.println("BeanBox caught exception "+ex+" while processing:
"+evt.getActionCommand()); System.err.println("  msg: "+ex.getMessage());
        if (ex instanceof ExceptionInInitializerError) {
            ExceptionInInitializerError ex2 =
                (ExceptionInInitializerError) ex;
            Throwable e = ex2.getException();
            e.printStackTrace();
        }
    }
}
}
}

```

---

---

```

* This implements the "save" menu item. This stores away the
* current state of the BeanBox to a named file.
*
* Note: The format is builder-dependent.
*/
public void save() {
// Write a JAR file that contains all the hookups and a
// single serialized stream.
// Then, on load, read the components and mock-up the AppletStub.

FileDialog fd = new FileDialog(getFrame(), "Save BeanBox File", FileDialog.SAVE);
// the setDirectory() is not needed, except for a bug under Solaris...
fd.setDirectory(System.getProperty("user.dir"));
fd.setFile(defaultStoreFile);
fd.show();
String fname = fd.getFile();
if (fname == null) {
    return;
}
String dname = fd.getDirectory();
File file = new File(dname, fname);

try {
    // create the single ObjectOutputStream
    File serFile = new File(serFileName(null));

    // we could use a JarEntrySource here to avoid writing the ser file
    FileOutputStream f = new FileOutputStream(serFile);
    ObjectOutputStream oos = new ObjectOutputStream(f);
    writeContents(oos);
    oos.close();

    String[] hookups;
    hookups = HookupManager.getHookupFiles();
    String[] files = new String[hookups.length+1];
    System.arraycopy(hookups, 0, files, 1, hookups.length);
    files[0] = serFileName(null);

    JarAccess.create(new FileOutputStream(file),
                    files);
} catch (Exception ex) {
    error("Save failed", ex);
}
}

```

---