COMP 290: Collaborative Systems [1]
Section 1: Introduction
Prasun Dewan

1 Introduction

In this course, we will study *collaborative systems*, also called *groupware*. We will study two forms of collaborative systems: (i) collaborative applications and (ii) collaborative infrastructures - systems that support the implementation of collaborative applications. This area raises both technical and social issues, in this course we will focus on the former.

1.1 Definition of a Collaborative Application

We need a definition of *collaborative/groupware applications*, so that we know what kind of applications to study and design infrastructures for. One straightforward definition of a collaborative application is: A computer application that allows its users to collaborate with each other.

So what exactly does “collaborate” mean? The Oxford dictionary defines it as: ”Work jointly esp. at literary or artistic production production; cooperate traitorously with the enemy.” Like most dictionary definitions, this definition is recursive (uses the synonym, “work jointly”) and subject to interpretation. For instance, it is not clear, according to this definition, if a chess game is a collaboration? Both players are working, they are doing it together, and a transcript of the game may (seem to them) to be an artistic production. But most people will say the players are competing rather than collaborating. It is perhaps because of examples such as this that Winograd defines groupware as a “state of mind”.

Several other definitions of collaborative applications have been proposed by researchers in this area [**?**]. Like the definition above, they are subject to interpretation, but give important insights on the nature of collaborative applications and how they differ from other traditional multiuser applications such as database applications. Malone defines them as “information technology used to help people work together more effectively,” implying, perhaps, that this new area is simply trying to take collaboration more seriously than traditional computer science fields. Lynch, Snyder, and Vogel hint at what is missing in traditional fields: “Groupware is distinguished from normal software by the basic assumption it makes: groupware makes the user aware that he is part of the group, while most other software seeks to hide and protect users from each other.” Wells clarifies that groupware is not simply software, it is ”software and hardware for shared interactive environments.” Ellis further qualifies this definition, requiring also the common task criteria: ”Groupware are computer-based systems that supports groups of people engaged in a common task (or goal) and that provide an interface to a shared environment.” Peter and Johansen-Lenz, credited in [**?**] with coining of the term ”groupware,” do not require that the tasks be associated with clear goals, They define groupware as ”computer-mediated culture, intentional group processes plus supporting software,” thus focussing on collaborative applications’ ability to mediate the interaction among its users.

The definitions above give us a good intuitive feeling about the properties collaborative applications have but use subjective terms such as “effectively”, “hide”, ”aware,” ”task,” and ”culture”. As a result, they are not sufficient to objectively distinguish collaborative applications from non-collaborative applications. Here is a more precise, system-based definition that is consistent with the definitions above: A collaborative application is a software application that (a) interacts with multiple users, and (b) links these users, that is, allows some input of some user to influence some output created for some other user (Figure 1). Input is

[1] Copyright by P. Dewan. All rights reserved.

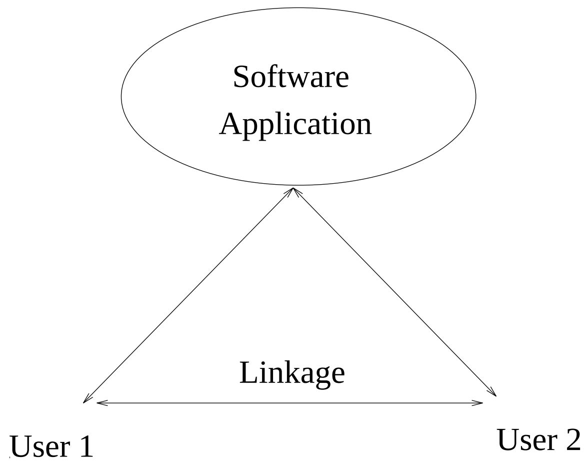

Figure 1: Definition of a Collaborative Application.

any event (e.g. keyboard, pointer, gesture, speech) generated by a user that is interpreted by the application and output is any event (e.g. screen update, audio output) generated by the application that is perceptible to a user.

This is a technical definition of a collaborative application and not a social one since the users linked together by the application may not in fact feel they are ”collaborating”. For instance, according to this definition, an application that allows users to send “flames” to each other is a collaborative application, though the users of the application may not feel they are collaborating. This definition essentially equates communication with collaboration. One can argue that communication is a necessary but not sufficient condition for collaboration and thus this definition includes several non-collaborative applications such as the “flame sender”. However, since collaboration is in the “eye of the beholder” or a “state of mind”, it is not clear how this definition should be further qualified to eliminate applications such as the one above, or in fact, if it is useful to eliminate these applications. In a technical course such as this one, it may not be as important to try to distinguish between collaborative and non-collaborative communication, since in both cases the technical challenges can be expected to be similar. More important, we will focus on actual applications that have been built and used, which presumably support useful, ”collaborative” forms of communication.

Some traditional applications that qualify as collaborative applications according to this definition:

- A mail application–linkage occurs when a user successfully sends a message to one or more collaborators.
- A talk program– linkage may occur at each keystroke or when the RETURN key is hit.
- A database application– linkage occurs when a user retrieves data that were deposited by another user or when a user is locked out because another user has acquired the lock needed by the former.
- A whiteboard application allowing users to create a shared drawing – linkage occurs when a drawing gesture made by a user is communicated to the other users.
- A video-conferencing application– linkage occurs when a user’s voice or video are communicated to other users.

Some applications that do not qualify:

- A traditional debugger allowing a single-user to debug a program at a time.
- A timesharing system–although the system interacts with multiple users it does not link them. Of course, it is possible to build file and interprocess communication primitives above time-sharing systems that support linkage.

In our definition we are including only overt I/O channels. As shown by Lampson, several covert I/O channels also exist, which could be used by users of non-collaborative application to, in fact, collaborate with each other. For instance, input to a traditional debugger may cause the load average displayed to another user to be changed. Is the combination of the applications that support debugging and load-average monitoring a collaborative application? Since the channels that link the two users are "covert" in that they were not explicitly set up by the two applications, we will not call the combination a collaborative application. However, it is important to keep in mind, specially for security reasons, that collaboration may occur among users of a timesharing system that are not explicitly linked together by collaborative applications.

Note that we call an application collaborative if it can link some input of some user to some output of some other user. We are not demanding that every input of a user immediately cause some corresponding output for all other users interacting with the application, though many collaborative applications, in fact, do so.

1.2 Motivation for Collaborative Applications

Collaborative applications enable Computer Supported Cooperative Work (CSCW). Why provide computer support for collaboration when people have been working together for ages without the computer?

- *Being There:* Computer tools can simulate face-to-face meetings among distributed users, giving them the illusion of ”being there” [**?**]. For instance a video-conferencing and whiteboard application can together simulate a face-to-face meeting conducted using a physical whitebaord. Today, this is considered perhaps the biggest motivation for CSCW.
- *Beyond Being There*: If all collaborative applications did was simulate ”being there,” then they would always support meetings that are inferior to the real face-to-face meetings, and thus be considered necessary evils by users who cannot be physically colocated. In fact, these applications can allow us to go "beyond being there," offering benefits we cannot get in meetings supported without the computer. We can have available to us all the resources in our local environment, be in several meetings at the same time, form subgroups without disturbing others, comment and vote with anonymity, be automatically forced to follow meeting protocols, rely on the computer to keep a log of the meeting and save state information, and do our private work when an item being discussed is not of interest to us. Hollan and Scott [**?**] argue that collaborative applications will not be truly successful unless people use them to collaborate with each other even when have the choice of face-to-face meetings. For that to happen, people must prefer the benefits "beyond being there" over the drawbacks of not actually being in a real face-to-face meeting.
- *Sharing of Computer State*: Collaborative application are also necessary to allow users to view and manipulate information that must be processed using the computer. For instance, a collaborative debugger is necessary to allow multiple users to jointly debug a program. Without collaborative

applications, users wishing to jointly manipulate information using the computer would be forced to huddle around a workstation, sharing a single physical set of input and output devices. This approach, though currently used for some pairwise collaborations, is not suitable for collaborations involving larger number of users.

It can be argued that applications that support distributed collaboration and allow us to go beyond being there are special cases of applications that allow sharing of computer state. The main difference is that in the former case computer support is provided for a collaborative task (such as a software inspection meeting) done traditionally without the computer and in the latter case we add collaboration to a computer-supported task (such as debugging) currently done individually.

While these may be well-argued reasons for CSCW, they have to be taken with a pinch of salt, since this area is still at the research stage and collaborative applications have not yet been deployed and tested extensively. Thus, we cannot say, based on actual usage, how important these reasons actually are. We can mainly argue, based mainly on hypothetical scenarios and lab experience, about the importance and nature of collaborative applications. In this respect, this course will be in the spirit of other advanced graduate courses, discussing several untested concepts that have not yet made it to the commercial marketplace.

Nontheless, there are several trends that indicate that the potential impact of CSCW would, in fact, be great:

- Some asynchronous groupware tools such as electronic mail, the Web, and Lotus Notes have gained widespread use.
- Some synchronous tools such as shared whiteboards, chat programs, videoconferencing systems and shared window systems have been commercialized by Intel, IBM, and perhaps more importantly, recently Microsoft.
- Lab studies of several non-commercialized collaborative applications have been positive.
- Surveys claim that much of the time (30-70 percent) of an office worker is spent in meetings [**?**] and organizations, specially software companies, are increasingly getting distributed.
- Not only have new, popular conferences emerged in this area, but traditional conferences in almost every systems area, including operating systems, database systems, software engineering, and user-interface systems, have special tracks on collaborative systems, making it currently one of the ”hot” areas.
- One can argue that every tool in the future would be collaborative since most complex tasks would be computerized and require collaboration. Thus, every application would have some state that more than one user may want to see and manipulate. Can you think of any application that you would not want to share in any situation?

1.3 Views of Collaboration Systems

We can study the area of collaboration systems from multiple, overlapping views:

- *Problems*: What are the problems that collaborative applications can solve and what kind of features are needed to solve these problems? This view ensures we are building systems that solve real problems and allows us to develop specialized solution for a problem.

- *Issues* What are the independent technical issues must be resolved by these systems and in what ways? From a systems point of view, this is perhaps the ”purest” approach in that it eliminates repetition of technical issues and allows us go in-depth into each of these issues. It corresponds, for instance, to the teaching of operating systems by identifying the technical issues such as process management, memory management, and process coordination; and teaching each issue in-turn.

- *Systems*: Which collaboration systems (applications and infrastructures) have been built and what are their properties? Most systems are collections of carefully integrated features, and this view allows us to understand specific collaboration constructs in the context of complete systems. It corresponds, for instance, to the teaching of programming language constructs using the comparative programming language approach, that is, teaching and comparing a variety of complete programming languages such as Ada, ML, Smalltalk, and Prolog.

- *Disciplines*: Which existing CS areas/disciplines do they extend and in what ways? This view allows us to understand new collaboration constructs in relation to existing concepts, thereby ensuring that we build on the knowledge and insight developed by traditional fields.

We can use these views to now define the nature and scope of this course.

1.3.1 Driving Problems

So what specifically are the problems solved by this research? Let us look at some of the problems have driven the research in collaborative systems. We will first look at some specific areas and them some general tasks that could benefit from collaborative systems.

Areas

- *Writing*: Documents of all kinds - papers, proposals, brochures, etc - are often coauthored, and manually managing a coauthoring process, specially when the coauthors are distributed, is generally difficult. Collaborative applications can allow distributed co-authors to easily observe and comment on each others’ activities and help ensure the consistency of the document.

- *Business Management:* As mentioned before, businesses require teamwork to be successful and are often distributed. Collaborative applications can enable managers in distributed businesses to make better group decisions [**?**], follow business processes [**?**], and monitor the status of ongoing projects.

- *Software Engineering*: Every phase of software engineering - requirements analysis, design, coding, testing, and maintenance, requires collaboration, and software development is rapidly getting distributed. Collaborative applications can help distributed software engineers share the results of software development tools such as debugging and testing tools [**?**] and to follow software processes [**?**].

- *General Engineering*: The benefits of collaborative software engineering extend to the general area of computer-aided engineering design/manufacturing, thereby allowing what is known as ”just-in-time” engineering. Of particular interest here are distributed collaborative VR interfaces for simulating real-world objects being engineered such as ships/automobiles [**?**].

- *Collaboratories*: The National Science Foundation has developed the vision of distributed collaboratories, allowing geographically-dispersed scientists working together on national/international projects to exchange results in a timely fashion and monitor and discuss data gathered from remote instruments [**?**, **?**, **?**].

- *Education*: There is increasing interest in distance education, especially in sparsely populated areas. Collaborative applications can enable distance education by allowing teachers to lecture to students in remote sites, lab assistants to consult with remote students. and distributed students to collaborate with each others.
- *Medicine*: Collaborative applications can enable telemedicine, and possibly even telesurgery [**?**], improve the communication among medical personnel and ensure that proper medical processes are followed.
- *Air Traffic Control*: One of the trickiest coordination problems is air-traffic control. Collaborative applications can allow air-traffic controllers and pilots to view upto date status information regarding the positions of aircrafts.
- *Command and Control*: Similar problems arise in military command and control operations, and collaborative applications can support planning and monitoring of these operations.
- *Games/Social Interaction*: As mentioned earlier, CSCW is not only about supporting work - it includes other forms of interaction such as playing games and, perhaps more importantly, interacting in informal meeting places [**?**, **?**, **?**].

Tasks

As we shall see later, the nature of a collaborative application often depends on the task it is supporting. Therefore, it is important to identify some of the important collaborative tasks that could benefit from computer support.

- *Design*: One of the most collaboration-intensive phases in the building of a product is its design, be it the design of an assignment, a software module, or an aircraft. Collaborative applications can help in this task by allowing users to brainstorm, make decisions, and create outlines of final products.
- *Implementation*: In comparison to design, the implementation of a product, in general, requires less communication among the developers of the product, as each developer, typically, works individually on a piece of the product. However, in this phase, it becomes more important to provide computer support to prevent/cure inconsistent concurrent interaction.
- *Inspection/Review*: Research in software engineering has shown that collaborative inspections of design, code, and other software documents is an effective mechanism for finding faults. Collaborative applications can, therefore, help distributed users browse and annotate documents, categorize faults, and follow appropriate inspection processes. They can be used not only for inspecting software documents but other kinds of important documents such as legal documents.
- *Consulting*: In the course of solving a problem, the problem solver often requires the help of a remote expert. For instance, a medical examiner trying to identify the cause of an injury might need the help of a remote pathologist who is the word expert on that particular kind of injury [**?**]. Similarly, a student might need the help of a remote lab assistant, a general practitioner might need the help of a remote specialist, and a person debugging a software module might need the help of a remote author of the module. Collaborative applications can help facilitate such remote consultation, allowing the problem solver to demonstrate the problem to the expert and receive advice.

This is a brief and superficial discussion of the CSCW problem space, which is still a matter of research. Later, we will discuss in more detail parts of this space when we look at various collaboration taxonomies, applications, and experience. At this point, what is important to observe is that there are a large variety of areas/tasks can benefit from the ability of collaborative applications to both allow distributed collaborators to feel they are colocated and to automatically provide facilities such as consistency control, categorization of faults, and decision making.

1.3.2 Systems

Several important systems, both applications and infrastructures, have been developed to support collaboration. The applications include:

- RTCAL (Real-Time Calendaring System) [**?**]: Developed as part of a Ph.D. thesis at MIT, it allows multiple users to schedule meetings in real-time. This research was instrumental in identifying several of the systems issues in collaborative systems.
- Cognoter [**?**, **?**, **?**]: Developed as part of the Xerox suite of collaborative applications, Cognoter allows users to collaboratively design an outline. A unique feature of this system is an automatically enforced meeting process. This research also helped identify issues in coupling among users, experimenting both with WYSIWIS (What You See Is What I See) and non WYSIWIS user interfaces.
- Grove [**?**, **?**]: Grove is also a group outline editor, developed at MCC. It recognizes the structure of the outline and provides fine-grained access control and non WYSIWIS user interface.
- GroupDraw [**?**]: GroupDraw is a group drawing tool, developed at the University of Calgary, which provides a new form of concurrency control called optimistic locking.
- MUD (MultiUser Dungeons) [**?**]: Developed by researchers at Xerox and elsewhere, it provides a text-based virtual environment in which people can meet and interact with each other based on which rooms they have entered.
- DIVE (Distributed Interactive Virtual Environment)/MASSIVE: Developed about the same time at Amsterdam and University of Nottingham, these systems, like MUDs, provide virtual meeting environments except that these environments are based on 3-D graphics.
- Coordinator [**?**]: Developed at Stanford, it is based on a theory of collaboration called the ”speech-act” theory and can be considered one of the first workflow systems.
- ActionWorkflow [**?**]: Developed by the inventors of Coordinator, this system is an extension of the Coordinator that provides more extensive workflow support.
- Quilt [**?**]: Developed at Bellcore, it provides rich support for collaborative writing, including logging of concrete and abstract user actions, typed annotations, and role-based access control.
- PREP (work in PREParation editor) [**?**, **?**]: Also supporting collaborative writing, PREP has been developed at CMU and provides novel facilities for commenting on the document, pinpointing the differences between different versions of the document, and coupling among users.

- CSI (Collaboration Software Inspection) [**?**]: This system supports both synchronous and asynchronous inspection of documents, providing facilities for making transitions between these phases of group work.

The infrastructures include:

- Web/Java: They offer several general-purpose facilities for constructing collaborative applications.
- XTV [**?**]: An example of a centralized shared window system, developed at ODU and UNC, XTV allows the windows created by an existing, collaboration-unaware, X application to be shared among multiple isers. It creates a single copy of the X application for all users sharing the application.
- MMConf [**?**]: An example of replicated shared window system, developed at BBN, this system supports sharing of collaboration-aware applications and creates a replica of the shared application for each user sharing the application.
- GroupKit [**?**]: Unlike the previous two infrastructures, which are extensions of window systems, GroupKit is an extension of a user-interface toolkit. Like MMConf, it supports replicated, collaboration-aware, applications, but provides higher-level support for implementing the application. Two different versions of it have been implemented, extending the InterViews and TCL/TK toolkits, respectively.
- Colab Programming Environment [**?**]: Also developed as part of the Xerox Colab system, this environment extends an object-oriented programming language with constructs for sharing arbitrary objects among users.
- Rendezvous [**?**]: Developed at Bellcore, it provides a declarative, object-oriented, constraint-based system for defining WYSIWIS and non-WYSIWIS user interfaces to shared objects.
- DistView [**?**]: Developed at the University of Michigan, it provides fine-grained replication of shared objects.
- Coterie [**?**]: Another distributed shared-object system, it has been developed at Columbia University and used to develop collaborative virtual environments.
- Suite [**?**]: Suite has been developed at Purdue and UNC and extends the C/Unix/X environment with support for a rich set of collaboration functions, including coupling, multiuser undo, merging, access control and concurrency control. It provides flexible support for each function, supporting high-level parameters defining a rich design space for the function.
- Oz [**?**]: This is a web-based system for defining process control.
- Trellis [**?**]: Developed at University of Maryland, Trellis associates shared objects with processes that control access to the document, thereby intergrating data and control.
- Sui'tre [**?**]: An example of interoperating systems, Sui'tre (pronounced Sweeter) is an integration of Suite and Trellis, combining properties of both systems and thereby providing both an interpretive and compiled environment for defining collaboration functions.
- Message Bus [**?**]: Developed at Brown University as part of the Field software development environment, it is a system supporting interoperation among general software system such as debuggers and editors.

- DistEdit [**?**]: Developed at the University of Michigan, it can be considered a special case of the message bus that connects existing single-user editors of different users to each other, thereby offering collaborative editing.

1.3.3 Discipline

:

Collaboration systems are related to and extend a variety of systems:

- *Operating Systems:* The goals of an operating are to (a) provide the basic systems software that comes with every machine and is used by a wide variety of applications (b) to manage resources shared among different processes. Collaboration systems are components of operating systems according to (a) if we believe collaboration is a fundamental requirement of an interactive application and (b) if we assume that different processes sharing a set of resources can execute on behalf of different users. While current operating systems allow the construction of a wide variety of collaborative applications, they have not been designed to support applications supporting close collaboration and as a result offer only low-level support for building these applications. Concepts in these systems can be reused/extended in collaboration systems. Important questions to be answered here are: What kind of collaboration support is required from a kernel? Does distributed shared memory support the construction of collaborative applications?

- *Database Management Systems:* Traditional database management systems provide automatic storage and access of data shared among processes executing on behalf of different users - facilities collaborative systems must also support. What they lack are facilities for defining complex data, sharing data among cooperating transactions, and triggering programmer-specified computations in response to access to data. Collaborations systems must extend database systems to support these facilities. Important questions to be answered here are: How should the serializable transaction model be extended to support cooperation? How can recovery be extended to support multi-user undo/redo?

- *Programming Languages:* The goal of a programming language is to provide general abstractions for supporting the construction of software applications. In collaborations systems, we will study how existing programming abstractions must be extended to support collaborative applications. Interesting questions here are: What kind of programming paradigm (e.g. procedural or declarative) is best suited for supporting collaboration? Is static or dynamic typing more important in a collaborative system?

- *User Interface Systems:* These systems provide abstractions for performing single-user I/O. We will study how these systems can be extended to support multiuser I/O. Important questions here are: At what level of a user interface system must multiuser I/O be supported: window level, toolkit level, UIMS level? What kind of architecture should be used to multiplex input and demultiplex output-centralized, replicated or something in between?

- *Software Engineering:* Many of the techniques and tools developed for supporting software engineering are relevant for collaborative systems including tools for supporting process control and interoperability.

Some of the concepts related to these systems are taught in existing courses on other kinds of systems. For instance multi-processing and concurrency control, two important components of collaborative systems, are taught in courses on operating systems and database systems, respectively. In this course, we will only review these concepts, and focus mainly on concepts related to collaboration systems that are not taught in traditional computer science courses.

1.3.4 Issues

Collaborative applications raise several design and implementation issues. The former address the user-interface of the application and correspond to the*collaboration functions* provided by the application to end user. The latter address how the application is programmed and its performance.

The design issues include:

- *Session Management*: How do distributed users create, destroy, join, and leave collaborative sessions?
- *Single-user interface*: What is the single-user interface presented to a user, that is, what are the application semantics if there is a single user in the session?
- *Coupling*: In a multiuser session, what feedback does a user receive in response to the input of another user?
- *Access Control*: How do we ensure that users do not execute unauthorized commands?
- *Concurrency Control*: How do we ensure that concurrent users do not enter inconsistent commands?
- *Merging*: How do we merge concurrent commands entered by different users?
- *Undo/Redo*: What are the semantics of undo/redo in a collaborative session?
- *User Awareness*: How are users made aware of "out of band" activities of their collaborators, that is, activities not deducible from the application feedback they receive from coupling?
- Fault Tolerance: How does the application react to faults?

The implementation issues include:

- *Objects*: What kind of objects are used to program a collaborative application?
- *Collaboration Awareness*: Which of these objects are collaboration aware and how are these objects integrated with existing, collaboration-unaware objects?
- *Layers*: How are these objects arranged in layers?
- *Concurrency*: How is the application decomposed into concurrent threads?
- *Distribution*: How are the application objects placed in different address spaces and hosts?
- *Replication/Migration:* Which of these objects are centralized and which are replicated? Which of the centralized objects can migrate?
- *Real-Time Support:* What kind of services are provided to ensure real-time interaction with tolerable jitter and latency?
- *Collaboration Awareness:* Which of the application objects are collaboration aware and how are these objects integrated with existing, collaboration-unaware objects?

- *Infrastructure Support*: Which of the application objects are implemented by the application programmer and which are provided by an infrastructure.
- *Interoperability*: How are objects of a collaborative application integrated with objects of other (collaborative and non-collaborative) applications?
- *Algorithms*: What are the algorithms used to implement the various collaboration functions of the object?

These views are overlapping since, for instance, good approaches to resolving collaboration issues can be found in existing collaboration systems. Thus, these views define a network of related concepts (Figure 2). In this course, we will traverse the network using first the systems view and then the issue view. Even though the other views will not be primary top-level views used to traverse this network, we will, nonetheless use them as secondary views. That is, for each construct we introduce as part of a system or as a resolution to an issue, we will consider the problems it solves and the relevant research in related disciplines. Since we will be looking at overlapping sets of concepts from two different views, there will be some repetition. The systems view will give us the broad context for many of the collaboration constructs we will discuss in-depth when we take the issue view later. We will not cover all of the problems, issues, disciplines, and systems because of since we do not have sufficient time and some of these issues, such as fault-tolerance in collaborative applications, have not been addressed in-depth.

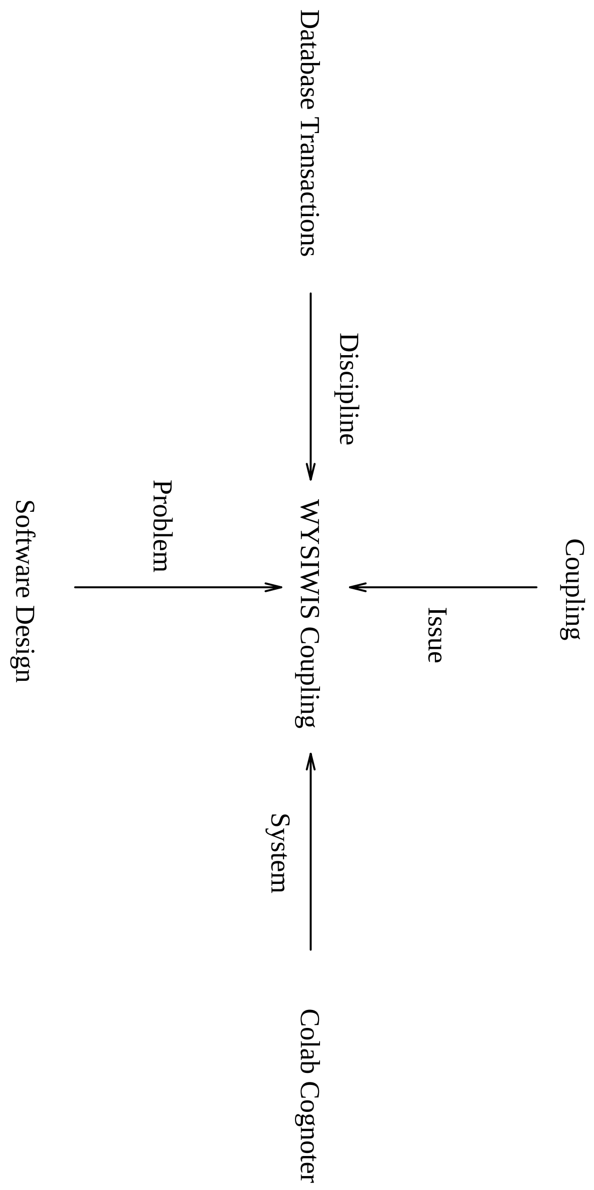

Figure 2: Different Views of a Concept.