Tools for Collaborative Software Design

James Wu

wuj@cs.queensu.ca

Technical Report 2003-462

Software Technology Lab, School of Computing

Queen's University

January, 2003

Abstract

Software design and development is a collaborative activity, potentially involving many people working on inter-dependent activities. This interdependence has motivated the development of tools that support collaboration in software design. There is a wide range of such tools, and this diversity is representative of the wide range of collaborative activities in which software designers engage. However, as no comprehensive design tool exists, designers must integrate heterogeneous tools to suit their need and accept the limitations of such a toolset. This means frequent migration between tools as the context of their work evolves. The purpose of this document is a survey the body of tools that comprise this toolset. This will serve to illuminate areas where further tool support is needed, and to motivate the design of new tools to address these needs.

Acknowledgements

I would like to gratefully acknowledge the contribution of my doctoral supervisor, Dr. T.C.N. Graham, without whose guidance and motivation this paper would not exist. I would also like to thank my wife, Anouchka, without whose love, support and patience this paper would never have been completed.

Table of Content

Table of Figures

1 Introduction

The design and development of large, complex software systems is a team activity, often requiring collaboration between a large number of people who share information and work on a variety of inter-dependent activities. A study by DeMarco and Lister found that developers working on large projects spend up to 70% of their time collaborating with others [25], while Jones found that team activities account for 85% of costs in large scale development projects [59]. This degree of interactivity between team members has necessitated the development of tools that can support collaboration within the software design process.

There are a wide variety of tools that are used to support collaboration in software design. They include routine communication tools such as a telephone or e-mail, electronic collaboration tools [14,84] that support synchronous communication and artifact sharing, as well as dedicated software design tools that provide traditional CASE (Computer Aided Software Engineering) functionality along with some degree of integrated support for collaboration [41,47]. Other such tools support informal media [66,97] to facilitate initial design and synchronous collaboration, while others focus on asynchronous artifact sharing [34,50] as a mechanism to support collaboration.

This diversity of available tools reflects the wide range of activities in which software designers engage. Software design involves more than simply the creation of formal design artifacts [17,91]. The content of design artifacts is developed through brainstorming and extensive discussion and analysis between designers before it is ever expressed in a formal design. Thus, a variety of tools exist to support these 'soft aspects' of software design [57]. This is motivated by the relative failure of most dedicated software design tools to integrate all aspects of design activities, in particular communication and informal interaction between team members. Although some tools exist that aim to facilitate and promote collaboration within the context of software design [23,30,66], most provide only the most basic facilities to support teamwork.

No comprehensive collaborative software design tool exists. Instead, designers cobble together a collection of heterogeneous tools to suit their needs, accepting the limitations inherent to such a toolset. This can mean frequent migrations between tools to suit current needs, which can impose additional overhead associated with switching tools, as well as recreating any design work that may not be importable into the new tool. For example, preliminary design work done on a whiteboard is not importable into typical analysis tools, and requires recreation before it can be used within such a tool. Challenges in building a more comprehensive collaborative design tool include integration of informal media with structured design tools, support for movement between different styles of collaboration, as well as flexible support for communication.

The purpose of this paper is to systematically survey the body of tools that support collaboration in software design. This will serve as a snapshot of the state of the art in the development of tools of this nature. Furthermore, it will illuminate areas where further research may be needed, and motivate the design and development of new tools to address any such issues. It is not the purpose of the document to describe the full functionality of each of these tools, but rather detail their support for collaboration in the design process.

1.1 The Nature of Collaborative Design

Collaborative design of software systems requires an environment that supports the interaction between developers and coordination of each member's knowledge, skills and expertise [3]. The nature of this kind of interaction is volatile, and changes frequently over the course of the design process. During the early stages, the focus rests on developing a shared understanding of what is to be built and how it is to be developed [111]. This involves the creation of informal, high-level requirements and designs, as well as frequent and varied communication and interaction between designers [62,63]. In these stages, designers tend to prefer the use of informal media such as paper or whiteboard because the structure imposed by most design tools can impose cognitive overhead and inhibit creativity [64,94,95,107].

However, much of the design activity cannot be readily supported with such informal tools [49]. As development progresses, designs and requirements become increasingly precise, and interaction amongst designers becomes more formal. In these stages, more formal, structured tools become useful in managing the growing complexity of the software. Such tools manage versioning, facilitate distributed access to artifacts, and promote cohesion between design and code. Additionally, the need for flexible communication and interaction between designers remains important.

What characterizes all stages of collaborative design is the variety of styles of interaction between designers as well as the artifacts with which they work [112]. Factors such as design progress, tool availability and distribution of collaborators change frequently, and designers accommodate these changes by adapting the ways in which they interact and the tools they use to do so. To understand how such interactions can change, consider the following scenario:

> *A group of designers are meeting informally to discuss preliminary design directions. Each participant makes individual notes on various issues of interest, and group notes are maintained on a whiteboard. A productive brainstorming session is interrupted because some visiting designers must leave and return home. They agree to participate in a future meeting via teleconference. They arrange a time for the next morning, and depart.*
>
> *At the next meeting, the group gathers around a whiteboard, as before. The visitors, at their remote office, participate by speakerphone. The group discusses the progress made since the last meeting. During the meeting, they exchange documents via email, and navigate the shared documents with verbal directions. After the meeting, Jack and Jill are emailed a transcription of the whiteboard contents for their own records.*

This scenario demonstrates how simple and common occurrences can influence the way in which designers work together. That the visiting researchers were forced to leave a design session affected the way in which the collaboration evolved, as well as the set of tools required by the collaborators to maintain their interaction.

1.2 Structure of this Document

This document classifies tools into six categories based on the novel *Workstyle* model for evaluating interactive systems [112]. This model characterizes a tool in terms of the collaborative work styles that it supports, as well as the kinds of artifacts that it is able to create. The definition of each category was motivated by the observation that clusters of tools conform to common evaluations using this model. Individual tools are categorized based on their common use, intent and the similarity of their Workstyle evaluation to these category definitions. By organizing the work in this fashion, this survey not only catalogues the tools available to support collaboration in software design, but also highlights where further tool development is required to better support the range of working styles in which designers engage. The *Workstyle* model is more completely described in section 1.2.1.

The content of this paper was influenced by field research involving the detailed observation of different of teams of designers at a large software company, each working in a wide variety of design activities. The categories of tools discussed here were originally revealed through application of the Workstyle model to the evaluation of their tools and collaborative activities. Furthermore, decisions to include or exclude individual tools or classes of tools were influenced by these observations. However, information and insight about individual tools was gained through first hand experience with them where possible, as well as extensive review of related published works.

The first category addresses *Software Modeling Tools*. These are tools that support expression and analysis of software designs in formal, typically graphical, languages. The second category contains tools that support software development through automation of various aspects a design process, collectively referred to as *Process Coordination Tools*. The third category, *Asynchronous Artifact Sharing Tools* or simply *Asynchronous Sharing Tools*, includes tools that provide support for collaborative design solely through asynchronous artifact sharing. In contrast, the fourth category is composed of *Synchronous Artifact Sharing Tools*, or *Synchronous Sharing Tools*, that support collaborative design through

synchronous sharing of artifacts. The next category addresses *Informal Media Tools,* and is composed of tools that support design work with informal media. Lastly, the final category considers additional *Communication Tools* used to support collaboration in design. It should be noted that certain tools may warrant categorization in multiple categories. In these cases, they are assigned to the most suitable category, with appropriate discussion of their extended functionality. The definitions of these categories with respect to the Workstyle model are discussion in section 1.2.2.

1.2.1 *Workstyle* Model for Tool Evaluation

The *Workstyle* model characterizes tool support for a working style as an eight dimensional space. Four of the dimensions address the style of collaboration and communication between designers. The remaining four address properties of the artifacts that are created during the collaboration. Support for a particular workstyle can then be represented as a point in this space. Similarly, support for a range of working styles can be represented as a region in the space. So, the functionality of collaborative design tools can be plotted in this space to specify the set of workstyles that they can support. It then becomes possible to compare preferred workstyles to those supported by available tools and to identify potential task/tool mismatches. These mismatches can be used to guide the design of new tools that are more appropriate to particular design activities. A graphical representation of the Workstyle model is presented in Figure 1. The axes are defined in the following sections.

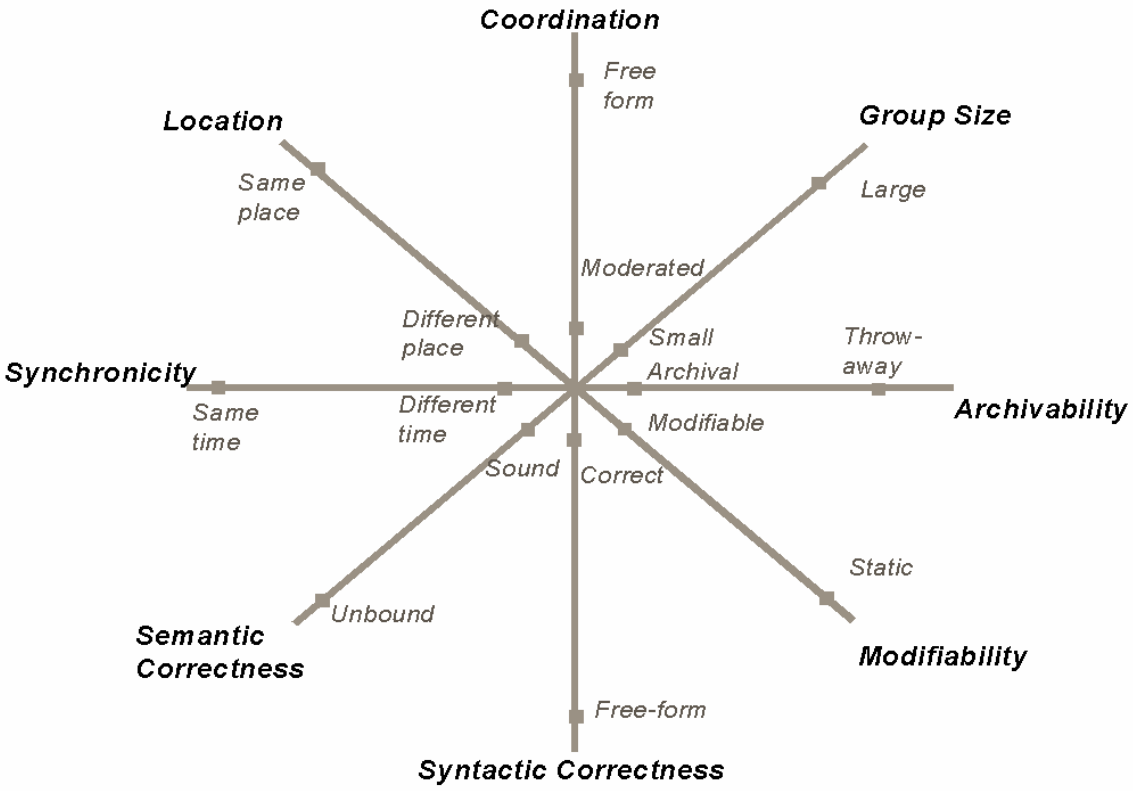

Figure 1: A graphical representation of the Workstyle Model

1.2.1.1 Dimensions Describing Collaboration Style

The first four dimensions describe the nature of the collaboration that can be supported by a tool. This is defined by the location and number of people involved in the collaboration, temporal aspects of their communication, as well as any coordination that is imposed on their interactions by social or technological constraints. The dimensions are defined as follows:

- *Location:* The location axis refers to the distribution of the people involved in the collaboration. As people become more geographically distributed, supporting certain collaborative workstyles becomes increasingly difficult [88]. For example, supporting synchronous communication is easiest in co-located, face-to-face workstyles, but may require complex concurrency controls and coordination policies when interacting at a distance. Along this dimension, people may be considered to be in the same place (*co-located*) or in different places (*distributed*). For example, collaborators may be from different locations in the same floor of a building, or from different geographical locations all together.
- *Synchronicity:* The synchronicity axis describes the temporal nature of the collaboration. People may work together at the same time (*synchronously*) or at different times (*asynchronously*). Synchronous collaboration facilitates more natural interactions between designers, as they are able to see the results of each other's actions in real time. However, synchronous interaction typically involves more complex tool support than asynchronous interaction. Face-to-face or telephone conversations are examples of same-time interaction, while email conversations or information sharing through a Lotus Notes database are examples of different-time interaction. Synchronicity is a continuous axis. For example, a rapid exchange of emails can approach a synchronous interaction.
- *Group Size:* The group size axis captures the number of people involved in the collaboration. Support for larger groups typically comes at the expense of intimacy in the interaction between collaborators. Conversely, limiting support to small groups affects applicability of the tool to larger projects involving larger teams. Generally, group size influences what styles of interaction are practical. For example, brainstorming may work in small groups, whereas larger groups may require support of tools or communication protocols to manage their interaction.
- *Coordination:* This axis describes any coordination that may influence an interaction, whether from the choice of tools or the adoption of some coordination model [67]. Free-style coordination supports informal interaction, though may break down as group size increases. Conversely, moderated coordination policies imposed by tools may inhibit progress in the early stages of design. For example, in a brainstorming session, free-form coordination is typical. Social protocols can determine the order of speaking or modifying shared artifacts. In meeting situations, more rigid coordination is typical, relying on a chair or even formal rules of order. Additionally, asynchronous tools typically rely on some form of moderated coordination, such as check-in/check-out protocols.

1.2.1.2 Dimensions Describing Artifact Style

The remaining four dimensions describe the nature of the artifacts produced. Artifacts are characterized by any required syntax or implied, as well as by the ease with which they may be stored, retrieved, and modified. These properties are defined as follows:

- *Syntactic Correctness***:** The artifact being produced may be required to follow a precise syntax. This requirement may inhibit progress in early stages of design by forcing initially abstract designs to conform to a predetermined syntax [64,111]. Conversely, a lack of support for enforcing or checking syntactic structure can increase ambiguity as designs grow in complexity. As an example of a syntactically correct artifact, consider a programmer who must follow the rules of the programming language to produce a source code artifact. Similarly, a designer may choose to follow the precise rules of a notation such as UML in order to create a design that will facilitate analysis.
- *Semantic Correctness:* An artifact is considered to be semantically *sound* if its meaning is unambiguous and free of contradiction. The production of semantically sound artifacts facilitates automated analysis and evolution. For example, UML designs can serve as a basis for automated code generation. However, the requirement to produce semantically sound artifacts may be impractical and unnecessary, and may inhibit progress in creative design. For example, design

documents often contain contradictions, inconsistencies and missing information. It has been argued that humans can often work effectively despite the presence of such contradiction [37]. Semantic correctness is a continuous axis.

- *Archivability:* Archivability represents the difficulty of saving an artifact so that it can be used at a later time. For example, word processing documents have high archivability, as they can be saved to disk and retrieved later. However, creation of documents that are easily archived involves a decision, a priori, about the value of the ensuing design. A whiteboard has poor archivability, as its content is lost once erased. Supporting archivability places restrictions on the creation of artifacts because specific tools support is required, and the decision to use such a tool must potentially be made before the value of the artifact is known. A doodle on a lunch napkin may evolve into a design, or may remain a meaningless doodle. However, any requirement to use a particular tool for the purpose of archivability may have prevented the doodle from ever being drawn. Archiving is considered more difficult if an archived version cannot be retrieved and used in the same way as the original. For example, a whiteboard drawing can be archived by photographing it; however, the archived version (the photograph) can no longer be manipulated on the whiteboard.
- *Modifiability:* This axis represents the ease with which an artifact can be modified. For example, small modifications to a whiteboard drawing are simply performed by erasing and redrawing. However, a modification such as reformatting a complex diagram is more difficult to accomplish in this manner. As with archivability, support for complex modification of an artifact requires specific tool support, which may impose sufficient overhead to prevent the artifact from ever being created. Similarly, though modifications to a diagram produced using a structured drawing program are more completely supported, they may require complex editing operations that can impede design progress more than less structured tools in certain stages of design.

1.2.1.3 Applying the Workstyle Model

To apply the model to the evaluation of a particular tool or set of tools, values for each property are plotted on the two-dimensional representation depicted in Figure 1. A particular workstyle is represented as a point in an eight dimensional space, while a range of workstyles is represented as a region in this space. Support for a single value in a particular property is indicated by a line intersecting the related axis, while a region over the axis represents support for a range of values in that property. So a plot that consists of a single line, with no expanded areas represents a tool or set of tools that supports a single, rigid workstyle. For example, see Figure 15. Conversely, a plot that covers areas of the graph represents a tool or set of tools that supports a range of workstyles. An example of such a plot can be seen in Figure 19. Once plotted, differences in the workstyles supported by various tools become visually apparent.

1.2.2 Tool Categorization

This document categorizes the various tools available to support collaboration in software design into six categories based on common Workstyle evaluations. Tools supporting similar workstyles are treated together, and the overall benefits and shortcomings of the entire class of tools are discussed.

Software Modeling Tools are characterized by their support for software modeling. They typically integrate one or more formal design languages, as well as variety of analyses for determining important design properties. This support for modeling and analysis also provides a basis for other functionality such as code generation and round trip engineering. The tools of this class provide support for collaboration through asynchronous repositories and revision control protocols. A generalized Workstyle evaluation of this category of tools is shown in Figure 3.

Process Coordination Tools are characterized by their integrated support for process modeling and enactment. They support the specification of collaborative design processes in order to coordinate team activities and communication. Furthermore, some of these tools support enactment of these processes.

This means they provide mechanisms for managing a design process as it executes. A Workstyle evaluation of this category of tools is shown in Figure 5.

Asynchronous Artifact Sharing Tools facilitate time-deferred collaboration by managing multi-user access to shared artifacts. Within this category, there are three identifiable tool subtypes; those that provide a 'shared-place', those that integrate team communication tools and multi-user repositories, and those supporting revision control and configuration management. Figures 8 and 9 graphically depict the Workstyle evaluations of these tools.

Synchronous Artifact Sharing Tools provide support for synchronous sharing of design artifacts and related applications. This means they facilitate concurrent viewing and manipulation of shared artifacts, and allow results of other's actions to be immediately viewable. There are three subtypes within this category; meta-tools that provide synchronous sharing for existing single user tools, collaboration-aware tools that integrate support for real-time sharing, and tools that support co-located collaboration. Workstyle evaluations for these tool subtypes are depicted in Figures 13, 14, and 15.

Informal Media Tools allow unstructured interaction through devices such as whiteboards, data tablets or paper. Once again, there are three subtypes of these tools; those that support a software design notation, augmented whiteboard applications and shared drawing tools. A graphical Workstyle analysis of all three of these subtypes can be found in Figure 19.

Communication Tools of a variety of types are also used to support collaboration in software design. These tools provide team communication and awareness facilities with no specific functionality for software design. The depiction of the Workstyle analysis for these tools is shown in Figure 21.

2 Category 1: Tools Supporting Software Modeling

Developing models for large-scale software systems is as essential as having an architectural blueprint in building construction. Good models are essential for communication among project teams and to assure important architectural properties in the resulting system. As the complexity of systems increases, so does the importance of applying rigorous modeling techniques. An example of this is the use of Unified Modeling Language (UML) [86]. UML is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software modeling tasks.

The tools in this category are characterized by their support for software modeling in a variety of graphical design languages, for example open-standard Unified Modeling Language [77]. These languages are formally defined, and provide a basis for advanced functionality such as static analyses and code generation. The most common of this category of tools is Rational's Rose. Other examples include Software through Pictures [6], TogetherSoft [104], as well research tools such as Rosetta [41], developed at Queen's University, and ArgoUML [7], developed as a free, open-source design tool. Although features vary across the tools in this category, none of these tools differ greatly from the approach taken by Rational to support collaborative software design.

The main benefits that these tools provide stem from functionality such as formal analyses, code generation and version management. Formal analyses allow properties of a design to be statically determined, while code generation reduces workload by generating code templates based on existing designs. The main drawback with these tools is that they are artifact centered, and focus their support on the production and management of artifacts rather than the activities and collaboration of designers. This cannot be considered sufficient support for group collaboration [57,62,63].

2.1 Collaboration in Software Modeling Tools

In this section, Rational Rose will be used as an exemplar of software modeling tools. We will describe Rose, and briefly describe how other software modeling tools differ from Rose in their support for collaboration in software design in terms of the Workstyle model. A graphical depiction of the Workstyle evaluation of this category of tools is shown in Figure 3.

Rose [81] is an object-oriented, visual modeling tool for software development. It supports a range of modeling notations, such as UML [77], Booch [16] and OMT [87], and provides code generation and both reverse- and round-trip engineering support for C++, Java and Visual Basic. Its support for teamwork is limited to the integration of distributed repository with configuration management and flexible locking granularity. An example of Rose in use is shown below in Figure 2. The entity-relationship diagram depicted is a UML class diagram. The boxes represent Java classes, and the arrows are relationships between them. A palette containing buttons to place any UML element within the diagram can be seen to the left of this diagram. The leftmost frame of the screen shows a hierarchical view of the entire current project, while the bottom pane contains detailed information about the classes represented in the design diagram.

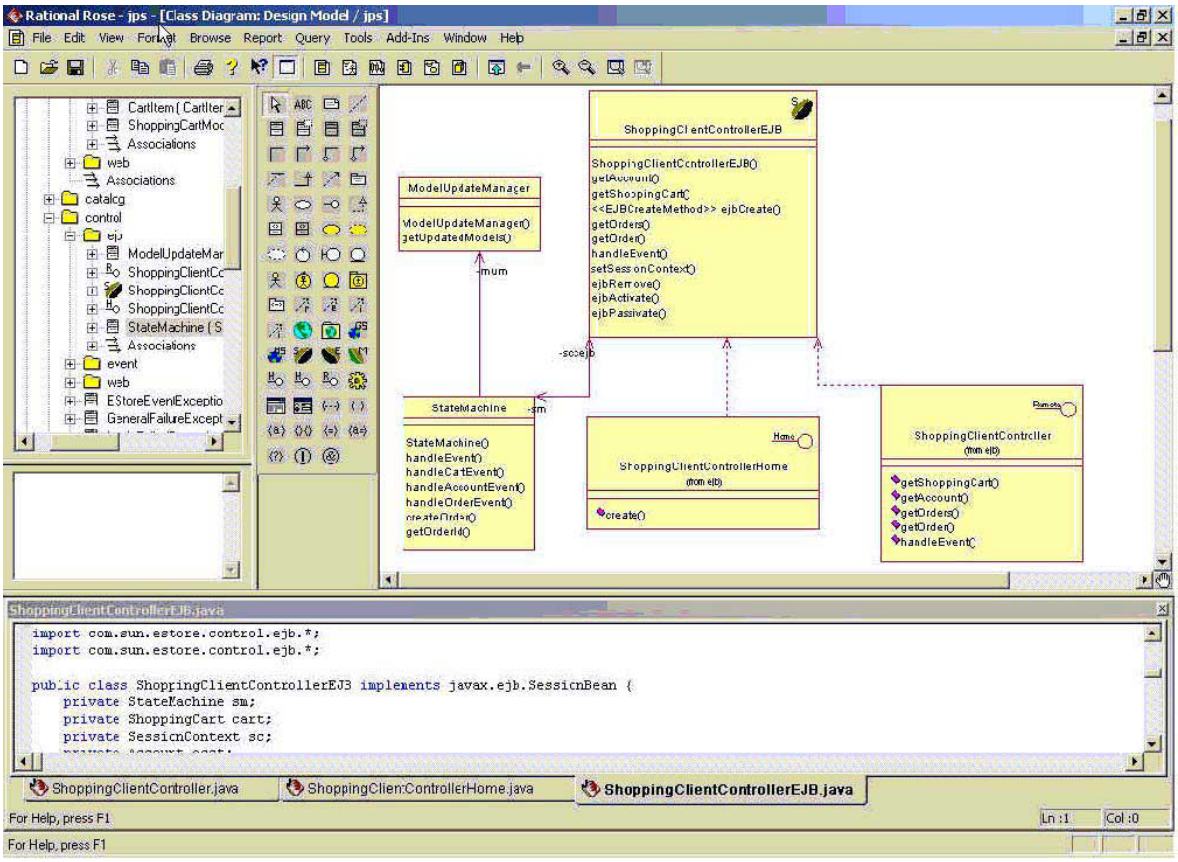

Figure 2: Developing a Java Class Diagram in Rational's Rose [81]

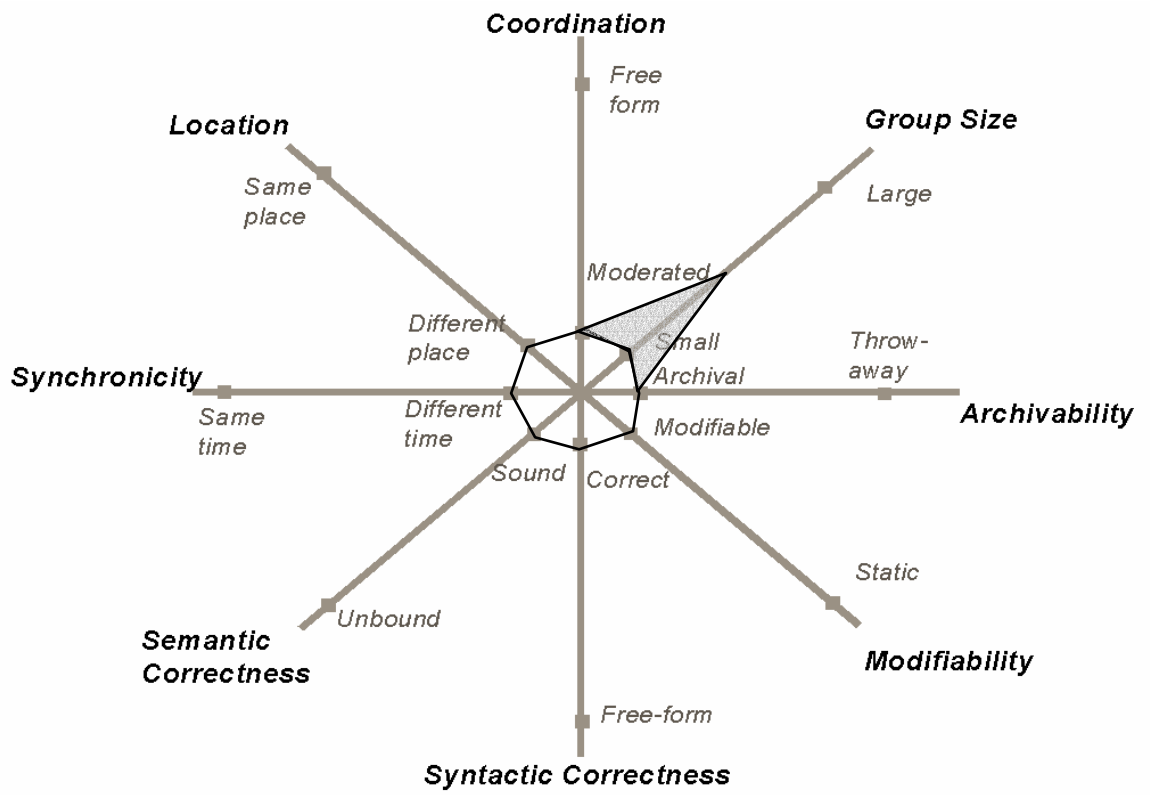

Figure 3: Workstyle evaluation of Software Modeling Tools

2.1.1 Collaborative Design Issues in Software Modeling Tools

Software modeling tools support software design through formal specification and analysis, as well as code generation and reverse engineering facilities. Software modeling provides a formal specification for a system, as well as a basis for advanced analysis techniques. Such a specification results in well-defined interfaces between components and facilitates independent parallel development, but requires precise syntax and defined semantics. This can impose undue overhead, particularly in early stages of the process [64,107] when designs are not sufficiently developed to be expressed with such rigor. This can make software modeling tools inappropriate for creative design tasks such as brainstorming. Code generation and reverse-engineering facilities also impose semantic interpretation on designs, as well as enforce their particular process on designers.

Integrated support for collaboration in these tools is based on the concept of a centralized multi-user repository. These repositories provide distributed access to documents, while managing any concurrency issues that may arise though synchronous, multi-user access. Typically, concurrency control is done through object-level locking of varying granularity. Version control facilities provided in some tools [104,102,82,72,34] also allow disparate models to be integrated to maintain consistency. However, these mechanisms impose a degree of coordination on collaborators interacting through them. Check in/out mechanisms limit the freedom that collaborators have to act in parallel, and versioning mechanism require consistency to be maintained. Generally, these tools are best suited to supporting distributed, asynchronous collaboration. We will now examine the workstyles supported by software modeling tools.

Archivability

Recall that archivability represents the difficulty of saving and restoring an artifact or document so that it may be used at a later time. Tools in this category excel in this area. At the most basic level, these software-modeling tools all allow the creation of digital, rather than physical, documents that are easily stored for on electronic media for fast and easy retrieval.

Rose provides direct integration with artifact repositories and revision control systems to facilitate storage. The tool provides built-in integration with Rational's own ClearCase and Microsoft's Visual SourceSafe, and may also be used with any command-line based versioning system. Like Rose, many tools in this category provide integrated repositories [80,69,35,32,102,104,6], while others rely on (and often provide support for) external repositories [81,7,76,52,41]. Some tools provide the flexibility of both. Repositories enhance archivability because they provide a central store for all documents while protecting them from unwanted or inappropriate deletion or modification. Many also support versioning of documents, and facilitate the archival of multiple concurrent versions of an artifact [104,102], or easily integrate with third party revision tools [81,6,32,7]. Furthermore, repositories often provide interfaces that allow artifacts contained within to be accessed by a variety of tools. This provides collaborators more freedom to use preferred tools for accessing and manipulating shared artifacts.

Rose documents may be exported to standard formats, such as XML and HTML, so that they may be stored in any kind of compliant repository or tool, or they may printed and archived in a physical filing system. Similarly, the archivability of artifacts created in many of these tools in this category is further enhanced by the individual tool's ability to export its artifacts in some standard format. These formats facilitate archiving and retrieval to and from different tools. Many tools support a variety of different export formats, including XML [81,7,69,76,102,104,6,41], HTML [81,32,6,41,104,102,35,69,80], and document and image formats such as RTF, ASCII, CDIF, JPG and BMP [41,7,76,69,52,32,104]. Structured formats such as XML, XMI and HTML derived formats can, and typically do, encode modeling information so that artifacts can be retrieved and manipulated in other tools in the same or similar ways as the tool in which they were created. Image formats only encode static syntactic information about the model, and cannot be manipulated further. Extensive support for data export further enhances the ability for collaborators to exchange design information without restricting their choice of tools.

Modifiability

Modifiability refers to the ease with which artifacts created within a tool may be modified, either within the tool or externally in other tools. All tools in this category provide fully structured editors for modifying the diagram types that they support. At the minimum, this means that they provide primitives for all of the entities and relationships defined by the diagram type supported by the editor, and relationships between diagram primitives are maintained.

Rose provides full structural editing for graphical designs, as well as language sensitive editors for code artifacts. Round trip engineering features allow for modifications in either the design or the code to be propagated so as to maintain consistency. Designs are input in a structured graphical editor that provides pre-defined primitives for the supported notations, UML, Booch and OMT. However, while structured editors are effective in producing precise and refined documents, they are not well suited to early stages of design when details about a design are undeveloped. The discrete and defined nature of the diagrams produced by such editors may imply structure and semantics where none yet exist in a design.

While modification of artifacts created in some tools implies independent editing of diagrams and code [41, 7], many tools provide features such as code generation and forward/reverse- and round-trip engineering. Code generation facilities generate various code artifacts in given languages based on the semantics of the design diagram. This facilitates the propagation of modifications made to the design down to the code and is supported by all of the rest of tools in this category [52,6,69,81,76,80,32,104,102,35]. However, some of these tools also provide reverse engineering [6,35,80,81,76], which allows changes made at the level of the code to be integrated into the design level diagrams. These features do not always produce clear and concise diagrams. Other tools provide an additional level of functionality and automatically synchronize design and code [104,32] such that changes made at either end are automatically and synchronously reflected in the corresponding document.

Syntactic correctness

Syntactic correctness refers to the requirement that an artifact conform to a precise syntax. This is effectively enforced by the structured nature of the editors provided within these tools to create the various diagrams – the primitives provided by the editor define the syntax of the diagram. All tools in this category impose this requirement, and differ only in the extensiveness of their support for the syntax of various graphical design languages. At a minimum, all of the tools in this category provide support for UML notation, although not all completely support the open standard specification [41,69,7,102]. Even the support for specific standards of UML vary between tools from versions 1.1 to 1.4; some tools do not specify the precise version of the standard that is supported. Other tools support additional diagramming techniques such as Booch [81,69,52,35,6], OMT [81,69,35,6], Jacobson [35,52], Object Interaction, Module Architecture Diagramming, and Buhr/Booch [52]. Most of these tools support context-free syntax checking only, though some [41] support a context-sensitive syntax. The requirement to conform to a precise syntax can interfere with early, creative design by restricting the expression of abstract design to the finite set of symbols provided by the editor.

Tools that support reverse engineering impose syntactic correctness on code artifacts as well. The syntax imposed is based on the language being reverse engineered, and may include Java [32,80,76,104,102,81,35,6], C/C++ [35,6,80,32,69,52,76,102,104,81], and others such as CORBA IDL [104,102,76,69,32,80,6], Visual Basic, Ada, Pascal, and SmallTalk.

Semantic correctness

This axis addresses the extent to which an artifact is required to be unambiguous and free of contradiction. For tools in the category, the semantics of artifacts are intended to be expressed with sufficient clarity and precision so as to facilitate analysis and code generation, both automatic and manual. This does not necessarily imply that artifacts are required to be expressed with such clarity. Rose can be used for object-oriented analysis, which can be arbitrarily imprecise. However, tools in this category are generally geared to facilitate the precise definition of meaning in artifacts. Many of the features supported by these tools reflect this. Synchronous round trip engineering [32,104] automatically synchronizes design and code artifacts, such that changes made at either end are appropriately reflected at the other. This feature can be considered to impose semantics on the design, as design level constructs are automatically interpreted to have a particular meaning at the code level. Other tools are less strict, and provide forward/reverse engineering features to give the developer some degree of control over the code-level interpretation of the design by allowing them to apply the semantics only when they feel it is appropriate [102,80,35,6,81,76]. MetaEdit [69]extends this flexibility by allowing the custom definition of semantics for any modeling formalism, while Rosetta goes even further and uses a *Conformance Checker* to highlight semantic inconsistencies without imposing any restrictions on the designer. Similarly, ArgoUML [7] implements *Design Critics*, which can be used to critique a design and suggest improvements. While this suggests a binding to a particular semantic interpretation, such suggestions are entirely optional and do not restrict the designer to the use of those semantics. Unfortunately, the production of design artifacts that are unambiguous and free of contradiction is imposes significant overhead, particularly with early design stages and brainstorming work styles. In these situations, semantics of a design are not well defined, and the inability to be ambiguous in certain areas of a design may hamper overall progress [37].

Synchronicity

Synchronicity addresses the temporal restrictions, if any, that a tool imposes on interactions between collaborators. No tool in this category supports truly synchronous interactions between collaborators, however concurrent development is facilitated through various version control mechanisms.

Collaboration with Rose is entirely asynchronous. The tool allows parallel development of a model by supporting decomposition of the model in units that can be placed under version control. Such units are referred to as *controlled units*, and stored as separate files in the operating system. When a controlled unit is loaded into a Rose model, it is considered write- protected or enabled depending on the current status of the corresponding file in the file system. Elements of a Rose model that can be placed under version control include the entire model, model properties, deployment diagrams, and logical, use-

case and component packages. The decomposition of the model into these units is determined individually. A separate tool, called *Model Integrator*, is provided to compare and merge models and their controlled units.

Most of the tools use object-level locking to permit synchronous access to a model while maintaining the consistency of the data [102,81,76,69,32,80]. Each object within the model is placed under separate version control, allowing multiple developers to access each object without restriction or risk of corruption of the data. The granularity of these objects is tool specific, and is in some cases user selectable [81]. Rosetta provides more granular locking at the diagram level, while others do not lock any aspect of the model [52,104]. Such tools support off-line versioning of the entire model, and provide integration tools to assist in the merging of disparate copies. Software through Pictures offers the option of either strategy, object-locking and versioning, though by default object-locking is used.

Technically, interaction with these tools can approach synchronous, as multiple designers can interact with the same model at the same time. However, none of these tools provide any additional functionality to support direct synchronous interaction between developers beyond merely facilitating concurrent editing of artifacts. No awareness of the actions of other developers is available to facilitate synchronous interaction. Furthermore, while modification to the state of the model can happen synchronously, individual views of the model do not update synchronously with such systems, thus any modifications made synchronously will appear asynchronous to collaborators.

Location

The location axis describes the kinds of geographical distributions of collaborators that a tool can effectively support. With the exception of ArgoUML, which does not integrate any team-support functionality, all tools in this category provide support for distributed collaboration only. Most of these tools use central repositories that enable file sharing through object locking, allowing concurrent access to designs through individual workstations [6,35,104,52,76,69,32,80,102,81]. Rosetta uses a different model to facilitate distributed collaboration – it is web-based and design artifacts are accessible through any Java-enabled browser. Technically, these access points may be located anywhere in the world, however practical considerations such as security may realistically limit the actual distribution. Additionally, these access points may be located as closely as the same room, even the same desk. However, none of these tools enable fully co-located collaboration wherein designers may collaborate using the same device.

Coordination

With the exception of ArgoUML, which provides no support for the coordination of collaborative activities, software modeling tools rely on moderated coordination through a centralized repository to prevent introduction of inconsistency into the model. For most of the tools, this coordination amounts to a check-in/check-out mechanism at the level of model objects [102,81,76,69,32,80] or diagrams [41]. Such a mechanism limits the degree of freedom developers have act in parallel, and therefore coordinates their collaboration. For example, conflicting check-ins must be resolved before they can be completed. For tools that support off-line versioning [52,104,6], interaction is coordinated through the process of merging the versions – the requirement that consistency be maintained enforces the need to coordinate, whether automatically or in person, the merging of disparate models. Furthermore, facilities such as round-trip engineering and code generation impose process coordination through automation. Code generation and reverse engineering produce artifacts, such as models or code, as a result of a process that is integrated into the tool and beyond a developer's control. Beyond these, no further support for coordinating collaborative activity is provided by these tools.

Group size

Generally, tools in this category can support a small to medium of collaborators, depending on the size of the models and the granularity of the concurrency control mechanism. Technically, Rose could support a large group of collaborators working asynchronously on different aspects of a large set of models. In this respect, the tool is limited only by the capabilities of the network on which it is installed.

However, synchronous work on an individual model is severely limited by the locking granularity of the revision control system. For example, at the level of a deployment diagram, Rose only supports a group size of one synchronous collaborator.

Owing to the object-locking nature of the repository integrated with many of these tools [102,81,76,69,32,80,41], both the size and granularity of the model at the object level, as well as the patterns of interaction with the model, directly influence the limits of supportable group size. A small model, consisting of fewer independent objects, can support proportionally fewer concurrent modifications before the limit is reached and a designer is prevented from accessing one of the objects. Furthermore, if the pattern of interaction with the model is such that many designers attempt to access the same object, then all but one will be prevented from making modifications.

For tools that support off-line versioning and merging [52,32,6], the limits on group size are related to the ability to merge multiple disparate versions of a model. Beyond this, the same arguments apply, as outline above. Proportionally fewer concurrent modifications can be made to a small model, consisting of fewer independent objects, before the limit is reached and the versions can no longer be easily or automatically merged. The same is true if the pattern of interaction with the different copies of the model is such that many designers attempt to access the same object.

2.2 Summary

Software modeling tools support software design through formal languages and methods and asynchronous document repositories. These tools are most appropriate to a formal collaborative work style characteristic of later stages of development after the initial, creative design work is complete. Modeling languages such as UML are useful for formally specifying and analyzing software designs, but are too cumbersome to support creative design activities where specific details are unknown, semantics are undefined and designs change frequently. Furthermore, the asynchronous interactions supported by these tools are better suited to these later stages when more of the relevant information has been precisely recorded within the shared design documents. This can reduce the need for extensive synchronous communication, as designs specified in these languages are semantically complete. This is in contrast to early stages of design, such as brainstorming, where synchronous communication is important to successfully eliciting the creative ideas required to develop more formal designs.

3 Category 2: Tools Supporting Process Coordination

Large-scale software development typically requires the participation of many distributed people, each addressing different aspects of the development. Each team may use its own selection of tools and internal processes while cooperating with the other teams to develop the product. Process-centered environments are used to develop process models that specify the coordination of people, computers, and software tools in support of such software development activities. In these tools, design and development processes are modeled and enacted. A process model is an actual definition of a project-specific process as specified by a process definition language [11]. An example of a process definition language is EVPL, Extended Visual Planning Language, which is designed and implemented by Grundy [44]. An example of EVPL can be seen in Figure 4. This diagram describes a simple software process model for updating a software system. Process stages *m1.1: design changes* and *m1.2: implement changes* have subprocess models. The expanded subprocess model for *m1.1: design changes* is shown in the window on the right. A process model can be instantiated by binding to real artifacts, users and tools, and become an enactable process model, or process program. These programs involve both physical and computational entities, and may be enacted physically and enforced socially, or implemented within an environment.

The tools in this category are characterized by their support for process modeling and enactment. This means that they assist in the specification of collaborative design and development processes, which in turn help to organize and coordinate the activities of development teams. They differ from other tools discussed in this document in that they do not produce artifacts related to the design of specific software products or support direct communication in a specific context. Rather, they support the design of the collaborative processes in which those artifacts are produced and direct communication may take place, and many provide facilities to enact those processes. Process-centered environments are a large field of study in their own right, so this analysis is not intended to be comprehensive, but serves as an overview of the their contribution in the area of collaborative software design. Examples of tools in this category include research systems such as Serendipity [44], Oz [11], Merlin [68], Marvel [9], SPADE [8], and ProcessWeaver [36], as well as commercial offerings such as eRooms [33], and SpeeDEV [90].

These tools offer many advantages for collaborative design. Formalized process definition supports analysis and refinement of development processes, and facilitates repetition. This means that a formally defined process is more easily and consistently repeated. Furthermore, it can enhance coordination between collaborators as well as management of available resources. These integrated environments also provide mechanisms for centralized project management and task traceabilility. The main disadvantage with these tools is the formality and rigidity of the processes they model and enforce. In many design contexts they can be heavyweight and obtrusive, and may conflict with the preferred processes of the designers and developers.

3.1 Collaboration in Tools Supporting Process Modeling and Enactment

As introduction to this category, process-centered environments will be considered in general. For reference, a screenshot of a specific environment, Serendipity [44], is depicted in Figure 4. Process-centered environments support collaboration in software design by disambiguating the collaborative process through formal modeling, and by enforcing the process through enactment. Support for collaboration within the processes developed in these tools is related to the nature of the process modeling language used. Each kind of language has strengths and weaknesses, and none is perfect. Generally, process-modeling languages can be characterized in terms of their support for explicit/implicit control-flow, local constraints and modular decomposition of a process [11]. Control flow refers to the ability to specify the set of activities or tasks, as well as any ordering or synchronization that may be applied to them. Local constraints are typically considered in the form of prerequisites to an activity, or implications of an activity. Support for modular decomposition provides flexibility in the specification of a process, allowing sub-processes to be integrated into larger workflows, as well as supporting re-use of process designs.

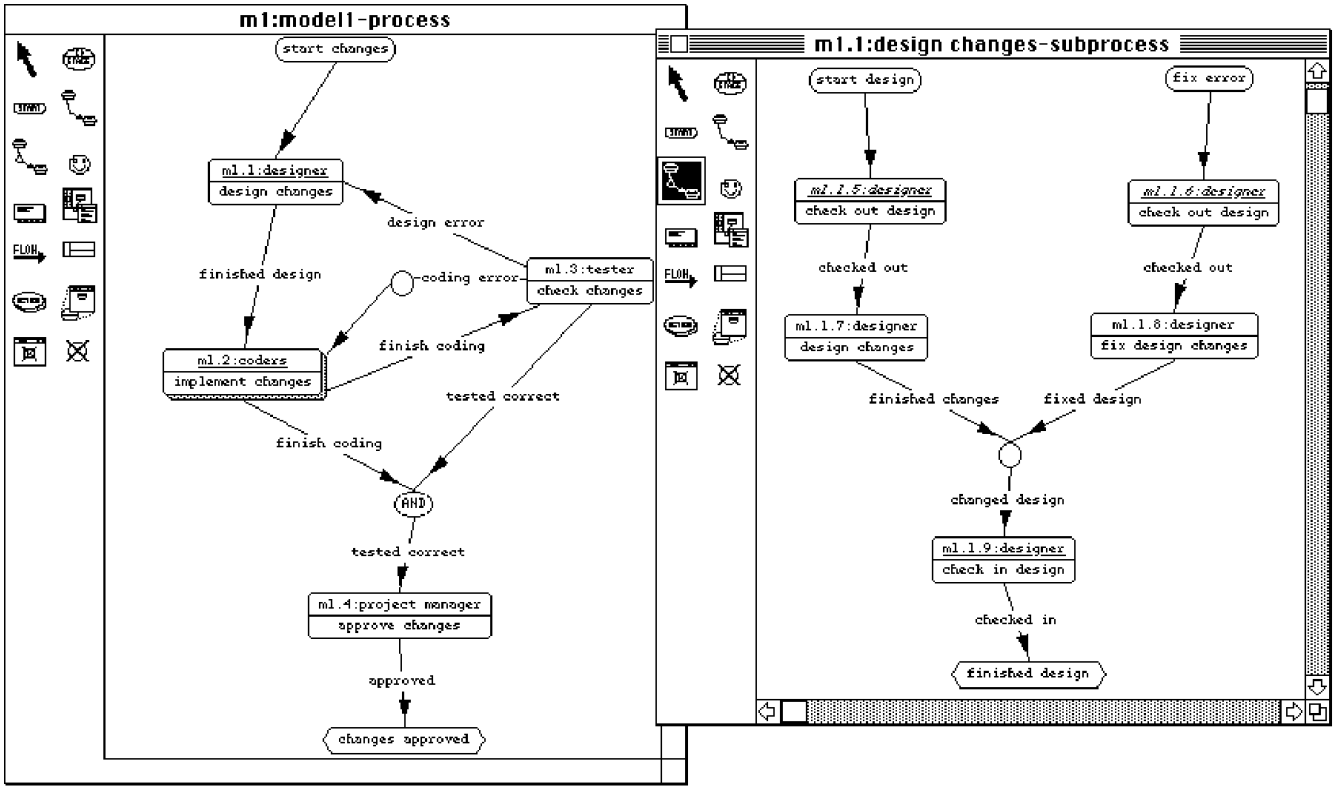

Figure 4: Developing an EVPL process model in Serendipity [44].

Rule-based languages [9,11,68,90,33] are best suited to defining local constraints and implicit control flow, but rules are inherently lower level and lack sufficient support for modularity and decomposition. Petri-net based languages [8,36] are good at explicit definition of the process' flow of control, but are also somewhat low-level and don't naturally support process decomposition. Grammar-based languages [54] support implicit control-flow and decomposition, while task-graph languages [44,94] support both explicit control flow and modular decomposition.

Individual differences between tools are discussed below in the context of the Workstyle model. A graphical depiction of this analysis is shown in Figure 5.

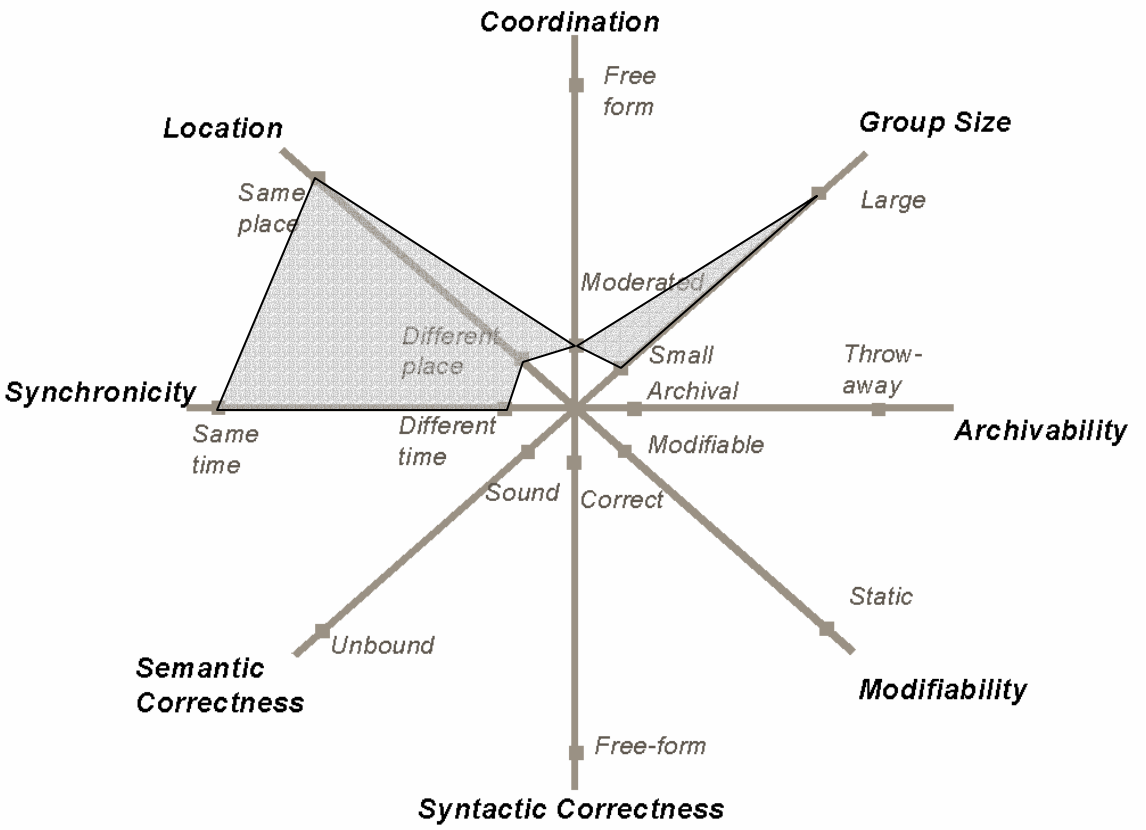

Figure 5: Workstyle Analysis for Tools Supporting Collaboration through Process Modeling and Enactment

3.1.1 Collaborative Design Issues in Tools Supporting Process Modeling and Enactment

Formalized process definition facilitates analysis and refinement of development processes, as well as consistent repetition in future projects. It can enhance communication and coordination between collaborators and resources by specifying the mechanisms and timing of interactions between distributed teams, team members and shared tools and artifacts. However, the formality of many the languages integrated with these tools results in rigid processes that may conflict with the preferences of users. Such processes can be heavyweight and obtrusive, and the result can be inhibited productivity and creativity in early design stages. Also, formal specification of such processes can be inappropriate for development of new types of software where the process is poorly understood.

A process-centered environment cannot enact all aspects of a process because humans are required to carry out significant portions. As such, different environments approach process enactment in different ways. This can influence the styles of collaboration that are supported. For example, some environments attempt to enforce a process, ensuring that constraints and obligations are met [68,11,44,90,33] before a process stage can be considered complete and the next stage enabled. This can impose unnecessary overhead in creative design situations by requiring a level of formality that is not appropriate to the design stage [64,111]. Similarly, some environments automate some aspects of the process and carry them out on behalf of the user [9,68,11,44,90]. This can restrict flexibility in how those tasks can be performed, and may discourage or prevent collaboration in these stages of the process. To address these shortcomings, some tools simply attempt to guide the user through the process, without forcing any kind of action [36,8], or just monitor the progress of a process [44,90]. Additionally, support for process evolution [8,44,9,11,90] is important in any such environment deployed in a collaborative

setting. This provides the flexibility to change an executing process to better match a team's collaborative needs.

As mentioned previously, tools of this category differ from the rest of the tools discussed in this document in that they do not produce artifacts related to the design of specific software products or support direct communication within a specific context. Because of this, the aspects of the Workstyle model that address the nature of the artifact produced by the tool are not applicable. While it is possible to analyze the modeling environment and resulting process model in these terms, it is not beneficial to understanding how these tools support collaboration in the software design process. As such, the following discussion will consider only the dimensions of the Workstyle model that address the style of collaborative work supported by a process – Synchronicity, Location, Coordination and Group Size.

Synchronicity

In general, all process-centered environments can model processes that support synchronous interaction amongst users as well as resources. The degree of synchronicity between collaborators that is supported depends on the nature of the process modeling language implemented. All modeling tools support intra-process synchronization [9,68,90,33], or *centralized* process modeling, while some support inter-process synchronization, or *decentralized* modeling, as well [11,8,44]. Furthermore, the environments themselves can support asynchronous collaboration [33,44,11,90,68,9,8], or real-time synchronous work through shared artifact views [33,44,11].

Location

The development processes produced and enacted by these tools all support both co-located and distributed collaboration. However, those tools whose modeling languages support inter-process synchronization [11,8,44] better support distributed work through the ability to synchronize between executing distributed processes. This means existing processes at remote sites can be integrated with each other, then treated and analyzed as single unit.

Coordination

Process models are intended to specify coordination of activities. Thus, when enacted they may provide and enforce strict coordination between collaborators. The degree of coordination depends on the nature of the enactment mechanism. Environments that enforce processes [68,11,44,90,33] can place the tightest restrictions on collaboration by preventing certain actions until conditions are met. Similarly, those that automate aspects of a collaborative process [9,68,11,44,90] may restrict or prevent collaboration in those process stages.

Conversely, environments that merely guide users along a predefined process [36,8], and those that provide mechanisms for monitoring the current process state [44,90] provide facilities for assisting in the coordination of collaborative activities without imposing undue restrictions on them. Similarly, environments that support evolution of enacted processes [8,44,9,11,90] provide the flexibility to adapt an executing process to better suit the needs of collaborators.

Group size

The processes developed in these tools can support all ranges of group size, from small to large. Group size depends on the process, the language in which it is specified, and how it is specified by the designer. However, process definition languages that support decomposition [44,54,94] and/or inter-process synchronization may specify processes involving larger groups more easily and robustly.

3.2 Summary

Process coordination tools support collaborative software design by facilitating the definition and supporting the enactment of collaborative software processes. They support a wide variety of collaborative work styles, depending on the design of the enacted process. Such a process may be designed to facilitate

synchronous or asynchronous work, distributed or co-located interactions, as well as all sizes of groups. Furthermore, the processes designed and enacted by these tools are independent of artifact type; i.e. they do not necessarily impose restrictions on the nature of the artifacts produced during enactment. The fundamental characteristic of work styles supported by these tools is coordination. A process is designed to coordinate interaction between people as well as availability of resources, and enactment is intended to impose the specified coordination. As such, people are restricted from a free-form style of interaction that may be most suitable to initial, creative design and/or the development of novel software systems.

4 Category 3 : Tools Supporting Asynchronous Artifact Sharing

Although supporting synchronous collaboration is important for the many reasons discussed in section 5, developers tend to spend the majority of their time in asynchronous modes of collaboration [40]. Asynchronous collaboration increases the potential concurrency amongst team members during the development process by allowing them to independently proceed with individual tasks. Asynchronous collaboration is most commonly supported in tools by providing remote access to shared artifacts. Concurrency controls are provided to prevent inconsistencies from arising, as are communication mechanisms such as email, chat and centralized scheduling.

The tools in this category are characterized by their support for asynchronous artifact sharing. This means they facilitate different-time collaboration by providing and managing concurrent, distributed access to shared design artifacts. The most common of these tools is Lotus Notes [50], a document database for gathering and distributing team artifacts, as well as team communication and scheduling. Other asynchronous artifact sharing tools include commercial offerings such as Lotus QuickPlace [51] and eRooms [33], as well as research tools such as Orbit [60,61], PoliTeam [78], Bayou [26] and FLECSE [30]. Additionally, stand-alone version control and configuration management tools such as Unix RCS [103], Rational ClearCase [82] and Microsoft Visual SourceSafe [72] are included in this category as they help manage the evolution of shared design artifacts.

There are three main kinds of tools that support software design through asynchronous artifact sharing:

- *Shared Places:* Some tools support collaboration through a 'shared place' metaphor [33,51,61,78]. They facilitate sharing and interaction by providing a common location for people to interact with shared artifacts.
- *Document Repositories:* Other tools support asynchronous collaboration by integrating team communication and management facilities with multi-user document repositories [50,30].
- *Revision/Configuration Management:* Version control and configuration management tools [72,82,103] provide support for the asynchronous evolution of artifacts by managing the creation, organization and use of concurrent, possibly divergent, versions of artifacts.

It should also be noted that many of the software modeling tools discussed in section 2 support collaboration through similar asynchronous mechanisms such as version control, configuration management, multi-user repositories, or integration with an appropriate external tool.

Asynchronous sharing tools offer many benefits in a collaborative work setting. Supporting asynchronous work maximizes parallelism by providing the ability for collaborators to work as independently as possible on shared tasks and artifacts. Document repositories provide an essential storage facility for project documents while facilitating distributed access. Additionally, version and configuration control facilities help to manage the divergent artifacts that are an inevitable result of such asynchronous collaboration, while maintaining both consistency and project history. Finally, those tools that implement 'shared-place' environments [33,51,61,78] foster cooperation and awareness by providing a centralized location for shared artifacts and communication [58]. The main disadvantages to asynchronous sharing tools is that they can be overly focused on supporting collaboration through artifact sharing, while failing to provide sufficient support for informal interaction and awareness between collaborators. Also, inflexible concurrency control mechanisms can impose unnecessarily restrictive coordination policies.

4.1 Collaboration in Tools Supporting Asynchronous Artifact Sharing

To introduce this category we will describe archetypes of each of the three subclasses of asynchronous sharing tools introduced above. More detailed discussion of the differences between these

tools can be found in the following section. Graphical depictions of the Workstyle analyses of these tools are shown in Figures 8, and 9.

- *Shared Places:* Orbit [60,61] is an example of a tool implementing a 'shared place'. A screen shot of this tool is depicted in Figure 6. The window on the left provides information about the shared locales that are available, as well as awareness information about current collaborators. The window on the right shows a collection of shared artifacts persisting within a locale. Orbit is a research prototype intended to provide a generic collaboration framework that supports the construction and design of domain-specific collaboration systems. It is based on lightweight desktop conferencing, access to shared repositories, ubiquitous awareness between collaborators, and flexible views of shared data. It evolved from the Worlds project [15,39], and was developed around the *Locales* [38] theory of collaborative activity. Locales theory builds on a spatial metaphor to provide context for actions and interactions in a collaborative work environment. A locale provides the setting in which cooperating team members can undertake their collective action or in which distributed teams or team members can meet together for some purpose. Examples of tools that provide similar 'place-based' collaboration mechanisms include eRooms [33], QuickPlace [51] and PoliTeams [78].

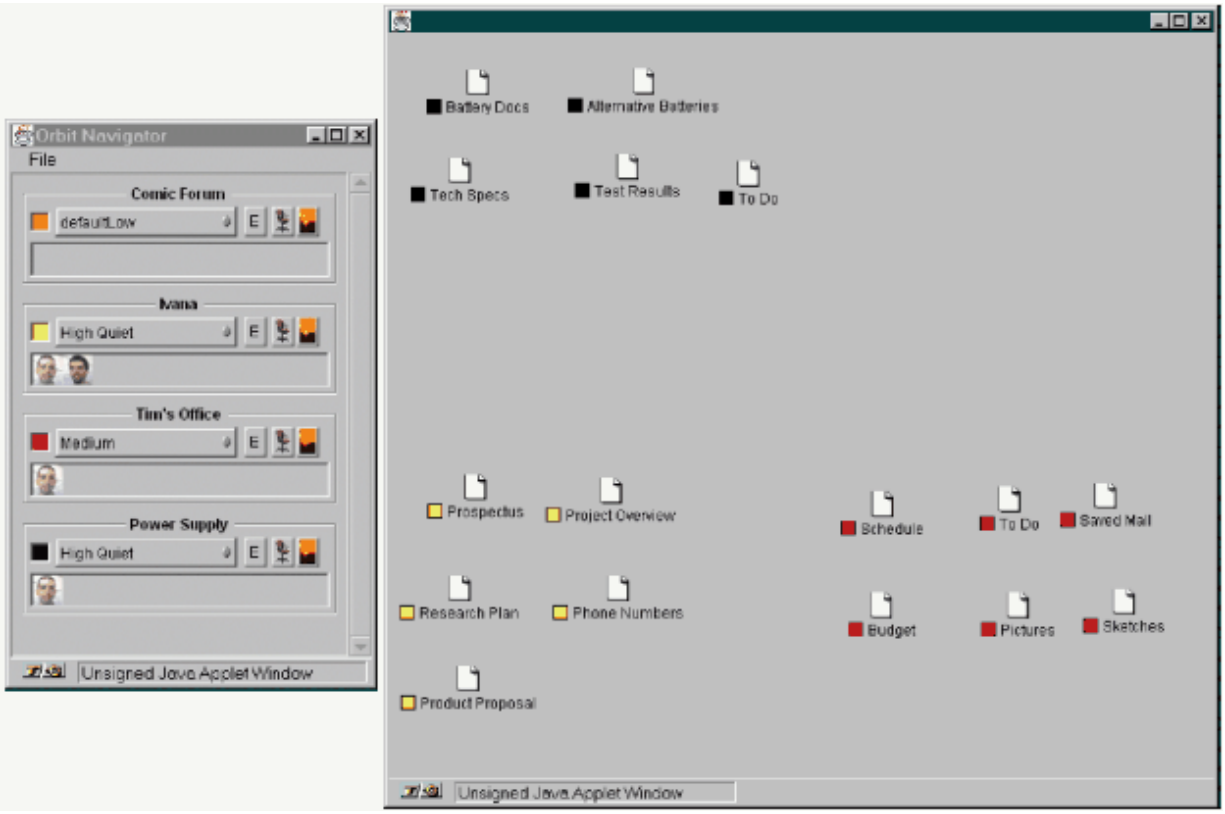

Figure 6: A Shared Place in the Orbit collaboration tool [60]

- *Document Repositories:* Notes [50] is an example of a repository-based tool for asynchronous collaboration. A screen shot from Notes is depicted in Figure 7, and shows an example Notes workspace. This workspace provides access to shared databases as well as a variety of asynchronous communication tools. Notes is a distributed client/server database application that facilitates coordination, communication, workflow management and information sharing. Its database foundations allow complex, unstructured

information of any type to be organized and made available to groups of networked users through a standardized graphical interface. Internal development languages facilitate extension and customization of the tool to better integrate with existing systems and processes. Notes also supports communication through email, chat, centralized scheduling and addressing as well as web publishing and browsing. FLECSE [30] and Bayou [26] are others example of a similar tools that supports asynchronous collaboration through multi-user repositories.

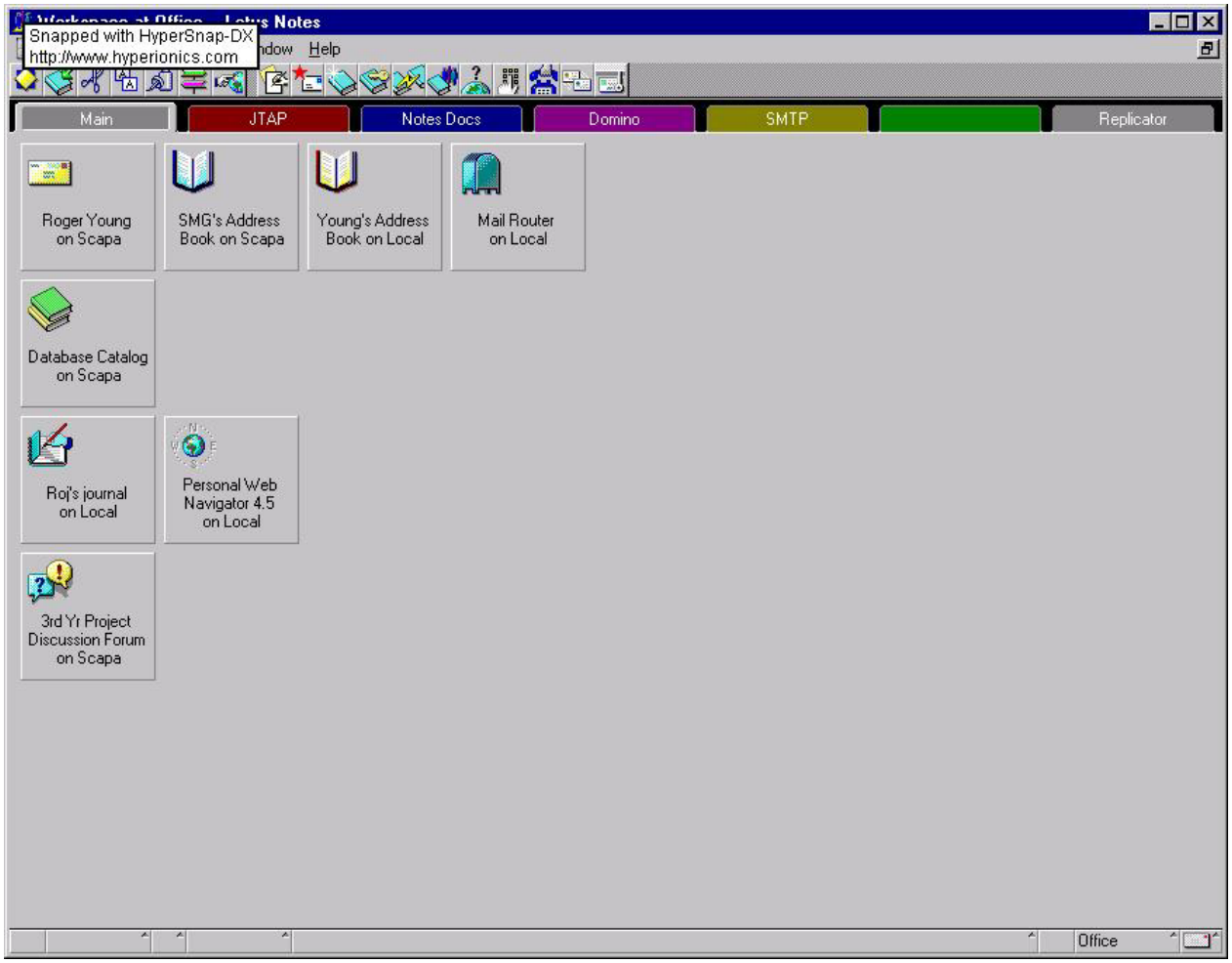

Figure 7: Asynchronous collaboration in a Lotus Notes workspace [50].

- *Revision/Configuration Management Tools:* Rational ClearCase [82] is a leading commercial tool for software version control and configuration management. It combines version control, workspace and build management, as well as defect and change tracking in an environment that accommodates a variety of development processes. ClearCase enables continuous parallel development by providing controlled web-based access to shared information, and integrates with third party development environments (IDEs) and other development and design tools. It is intended to simplify and manage the change process for complex development artifacts. Other similar tools include Microsoft SourceSafe, Unix RCS tools and the Voodoo [34] document management system.

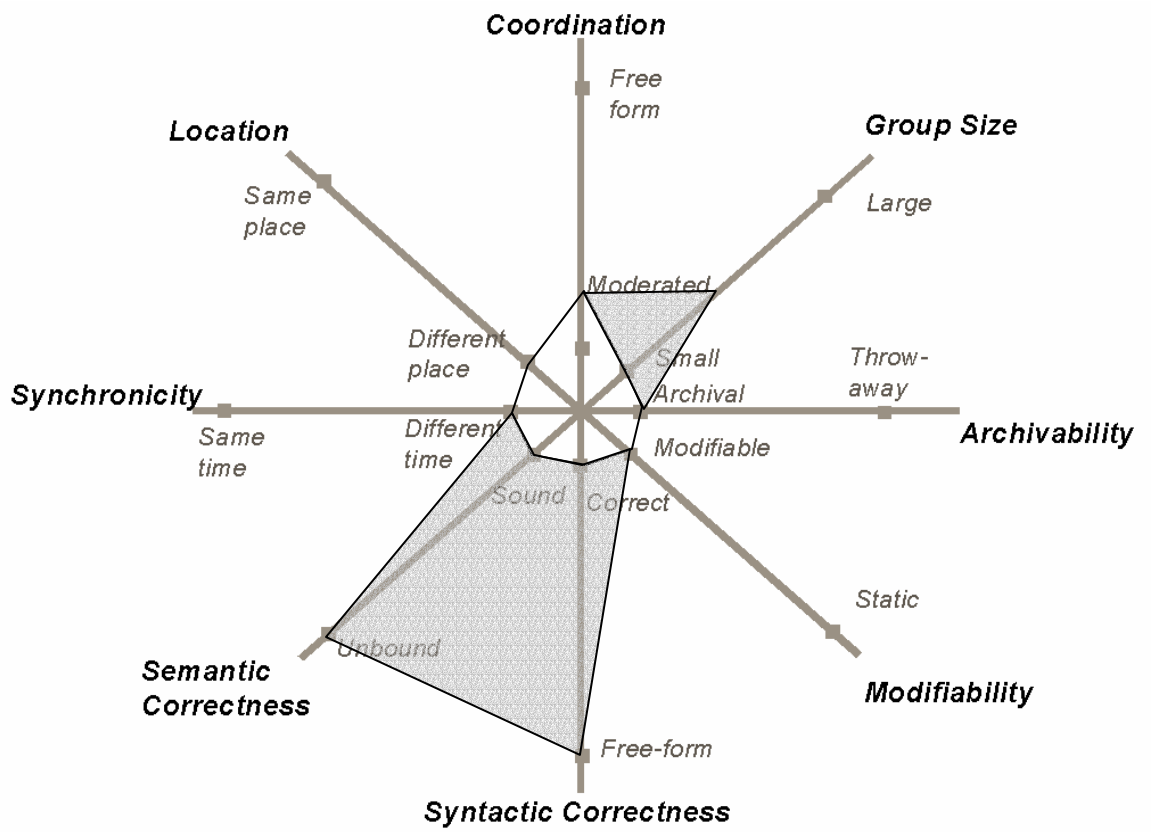

Figure 8 : Workstyle analysis for tools supporting asynchronous collaboration through shared document repositories and configuration/version management tools

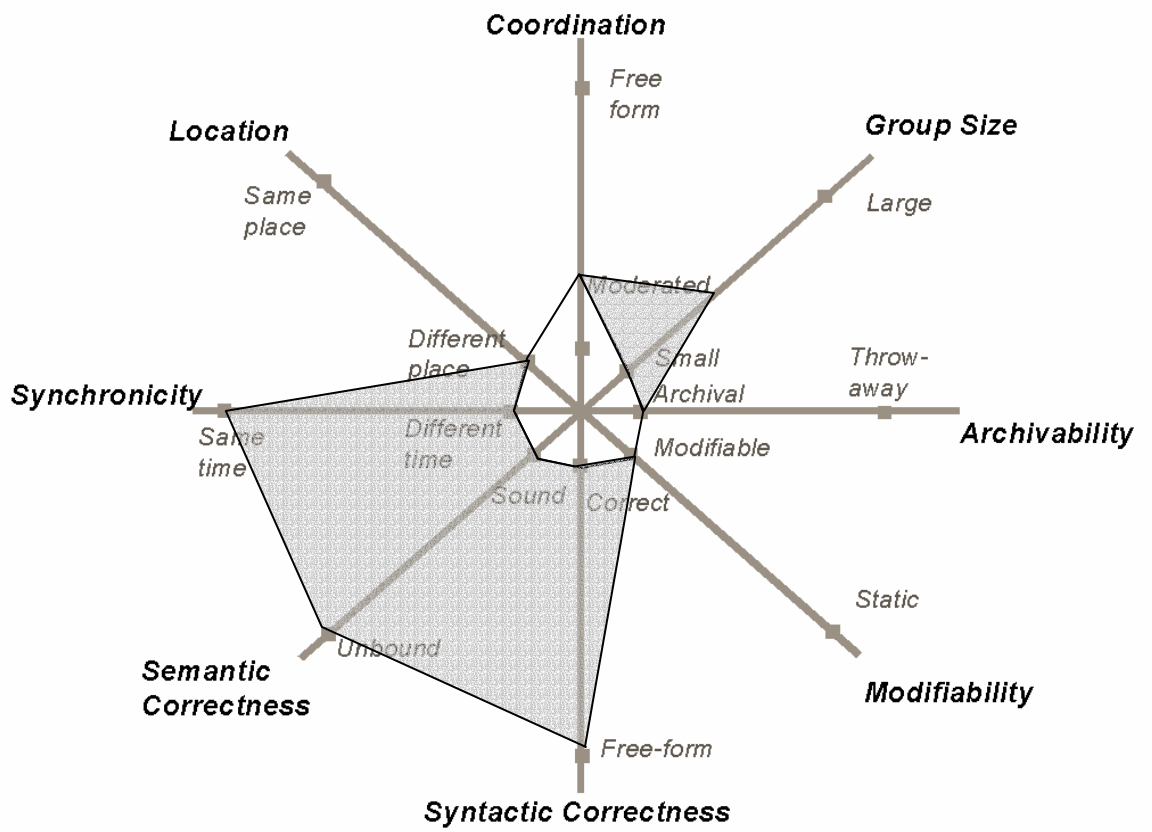

Figure 9: Workstyle Analysis for Tools Supporting Asynchronous Collaboration through a Shared Place Metaphor

4.1.1 Collaborative Design Issues in Tools Supporting Asynchronous Artifact Sharing

Working in an asynchronous manner allows team members to work concurrently and independently on shared tasks and documents, without compromising the consistency of the design artifacts. In practice, this is implemented in a variety of ways. For example, web-based access [51,61,33] enables a team member to access shared data from anywhere in the world with commonly available tools. Similarly, enabling remote access to document repositories [50,30,51,61,33,26] provides constant and concurrent access to design documents from distributed locations. Check-in/out and locking protocols [51,50,61,30,72,82,103] maintain data consistency by locking artifact elements at varying granularities to ensure that no aspect of a design artifact is modified by more than one person. However, such protocols can impose inflexible coordination between collaborators that may be unacceptable in certain stages of design and development. Similarly, tools with version and/or configuration management facilities [51,61,30,72,82,103] support creation and tracking of multiple versions of an artifact, and provide mechanisms to merge divergent copies into a single, consistent state. However, automatic merging is not always possible, and often requires human intervention. This can impose further restrictions on coordination. Finally, some tools [78,61,30,33] provide awareness facilities to alert others to actions and changes that are being made, allowing social protocol to prevent inconsistencies.

However, supporting asynchronous work solely through artifact management neglects the human aspects of collaboration [57,62,63]. Although tools that implement 'shared-place' environments [33,51,61,78] can foster cooperation and awareness through a centralized location for shared artifacts and communication, they are overly artifact-centered and do not support lighter weight and real-time interactions. Such interactions are very important to early, creative stages of collaborative development.

Similarly, asynchronous communication mechanisms such as email and chat are often not rich enough to convey the wealth of information exchanged by collaborators. These tools lack support for annotations, shared drawing surfaces such as whiteboards and other mechanisms for lightweight, informal communication. We now consider the workstyles supported by asynchronous artifact sharing tools.

Archivability

All of the tools in this category provide complete support for document storage and retrieval. The ability to archive design artifacts is central to supporting asynchronous collaboration via shared access to resources such as repositories [50,30,51,61,33,26] or shared places [33,51,61,78]. Repositories are document databases that allow remote retrieval and storage of a variety of document types. For example, Orbit integrates three different kinds of repository: BSCW for storing standard office document, Envy for Smalltalk code, and CVS for other code bases. Conversely, Notes databases are considered document-oriented object stores for semi-structured and unstructured data, and can store heterogeneous document types. Tools that use a 'shared-place' metaphor support archivability through object persistence. Like physical places, objects left in a place remain there until they are taken away. Thus, any artifact brought into such a tool is automatically archived until it is expressly removed.

Modifiability

Though artifact creation and modification is not central to the functionality of asynchronous sharing tools, modifiability of artifacts is enhanced by version and configuration management control facilities. Versioning allows multiple copies to be modified, with any changes easily tracked and undone if necessary. This facilitates making changes to documents because parallel versions can be created and maintained without the risk of destroying the original.

Synchronicity

As indicated by the category name, asynchronous sharing tools support asynchronous interaction between collaborators. The degree of synchronicity is dependent on concurrency control mechanism. In those tools that rely on repository-based check-in/out or locking protocols [51,50,61,30,72,82,103], synchronicity is limited by the granularity of the entities that may be checked in/out. Courser grained protocols may limit one person per document, while finer-grained ones may control smaller elements within a single artifact. This can facilitate more synchronous work. Similarly, tools with version and/or configuration management facilities [51,61,30,72,82,103] support a greater degree of synchronicity by supporting the creation and tracking of multiple versions of an artifact that can result from concurrent and independent work on a shared document. They also provide mechanisms to merge divergent copies back into a single, consistent state, which can facilitate transitions from asynchronous to synchronous work styles. Finally, some tools support fully synchronous interactions [33,78,30] and communication mechanisms [33,30], supporting shared views of artifacts and real-time communication tools.

Location

These tools support distributed collaboration only, though co-located collaboration is technically possible in highly asynchronous situations where team members share the same machine to access shared artifacts. Web based tools [51,61,33] can support widest distribution, as no special tools or installation procedures are required to access shared resources or interact with team members. Conversely, some tools [72] use proprietary communication protocols that limit their operation to smaller, local area networks.

Coordination

Tools in this category rely on moderated coordination through access controls [51,50,33] and repository-based check-in/out or locking mechanisms [51,50,61,30,72,82,103]. These mechanisms can limit the freedom that team members have to act independently, thereby coordinating their actions. Furthermore, tools that support versioning [51,61,30,72,82,103] can require and impose coordinated interaction during the merge process, as consistency requirements enforce the need to coordinate the

merging of divergent models. Finally, some tools provide limited support for process design and enactment [33,50,51], which can enforce a strict coordination between collaborators by preventing or requiring certain actions under particular conditions.

Group size

Generally, tools in this category support a reasonably large number of collaborators working asynchronously. However, concurrency control mechanisms such as check-in/out protocols and object-locking [51,50,61,30,72,82,104] limit group size based on the granularity of the protocol. For example, for large enough groups, or small document sets, it is possible that all artifacts become checked out and/or locked, preventing any additional team members from working. Additionally, versioning and configuration management facilities [51,61,30,72,82,104] place limits on group size based on their ability to merge multiple, disparate versions of an artifact. As the number of parallel versions increases, it becomes increasingly difficult to easily and/or automatically merge divergent versions.

4.2 Summary

Many of the tools in this category support asynchronous collaboration by integrating document repositories, team communication facilities and/or configuration/version management. These tools support fairly restrictive workstyles. They are flexible in the nature of the artifacts that they support, as there are a variety of such tools to accommodate any artifact type. However, interaction with those artifacts is highly moderated and restricted to small or medium distributed groups working asynchronously.

The remaining tools support asynchronous collaboration through the notion of a ‘shared place’. These tools support a similar set of working styles as those described in the previous paragraph. They are independent of artifact type, and interaction with those artifacts is moderated by concurrency control mechanisms to maintain consistency. However, ‘shared place’ tools may also support synchronous work styles, though only with smaller group sizes.

5 Category 4 : Tools Supporting Synchronous Artifact Sharing

The tools in this category are characterized by their support for synchronous artifact sharing. This means that they allow multiple designers to concurrently view and manipulate the same artifact, and see the results of other's actions in real time. This is in contrast to other artifact sharing mechanisms, where copies (or portions) of artifacts are checked out, modified and merged with an original or a parallel version. The most ubiquitous of these tools is Microsoft's NetMeeting [70], a screen-sharing application. Others include TeamRooms [84] developed at University of Calgary, and JComposer [47] developed by the universities of Waikato and Aukland.

Synchronous artifact sharing tools vary greatly in their overall functionality and intent. There are three kinds of tools in this category:

- *Meta-tools:* These are tools that provide synchronous sharing for collaboration-transparent design applications. For example, NetMeeting supports the sharing of generic Windows applications, as well as other activities such as collaborative drawing, file sharing and chat. This meta-tool could be used to support synchronous, collaborative software design in a collaboration-unaware system such as Rational Rose.
- *Collaboration-aware Tools:* These are tools that integrate support for synchronous artifact sharing. For example, JComposer [47] is a fully functional CASE tool that supports synchronous sharing of its own design artifacts.
- *Co-Located Tools*: These are tools that support synchronous sharing for co-located collaborators. For example, Insight Lab [65] facilitates co-located teamwork through the interaction of computer technologies with environmental objects such as papers and physical whiteboards.

Regardless of their differences, the main benefits that these tools offer is the ability for collaborators to work together on the same artifact(s), while providing awareness of each other's actions. Their main drawback is that they support only a small group of collaborators, and may impose restrictive coordination models in exchange for artifact consistency.

5.1 Collaboration in Synchronous Artifact Sharing Tools

As an introduction to this category, we describe Netmeeting, JComposer and InsightLab. These tools serve as exemplars of the three subtypes of tool described above. The differences between these tools will be discussed in context of the Workstyle model. Graphical depictions of this analysis are show in Figures 13, 14 and 15.

- *Meta-tools:* NetMeeting [70] is a general collaboration tool that supports generic application and file sharing, shared whiteboards, textual chat, as well as voice and video conferencing (see Figure 13 for Workstyle analysis). An example of NetMeeting in use is depicted in Figure 10. Two collaborators are synchronously searching the web. Two telepointers can be seen. The 'arrow' indicates the user who has control over the application, while users without control over the shared application are indicated by a circular pointer. It allows software designers to share existing design tools that are collaboration-unaware, i.e originally designed for single users. Netmeeting also supports communicating through voice, video, text and/or the whiteboard application. However, its support for teamwork is limited by a turn-taking coordination policy that limits concurrent interaction with shared artifact, as well as only supporting point-to-point collaboration for voice and video streams. Other application-sharing tools may not share these restrictions. Examples of similar tools include CECED [20], Disciple [108], JAMM [10] and Xtv [113].

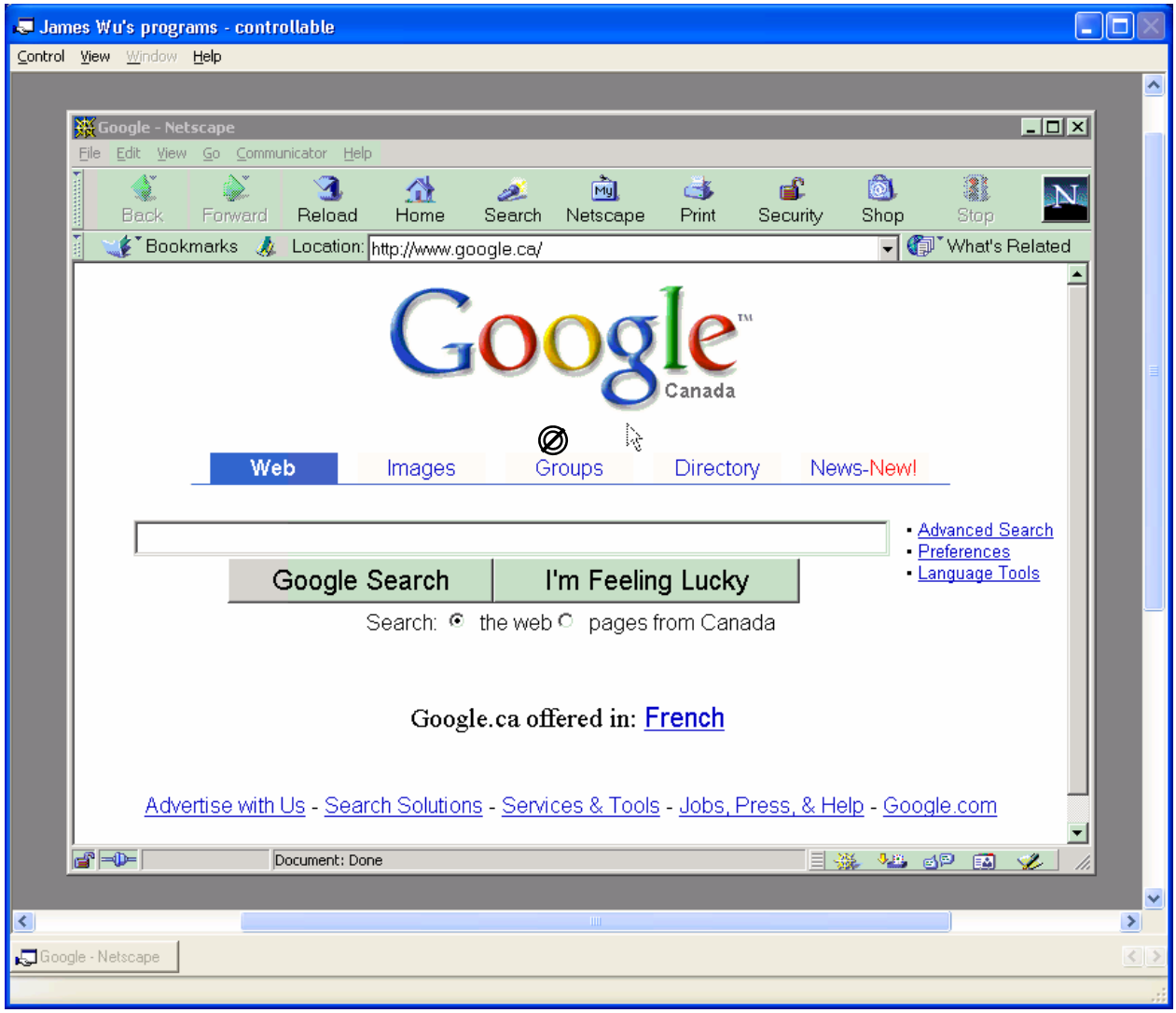

Figure 10 : Collaborative web browsing through Netmeeting. Each user has their own telepointer.

- *Collaboration-aware Tools:* JComposer [47] is an object oriented CASE tool for creating and manipulating graphical and textual views of object-oriented and component-based software systems (see Figure 14 for Workstyle analysis). An example screenshot from JComposer is shown in Figure 11. The large window contains a view of the shared artifact, while the surrounding windows provide awareness information about collaborators. Jcomposer supports specification, design, implementation, documentation and reverse engineering. The tool integrates a proprietary design language, *Jcomposer Architecture Description Language* [45], and a structured editor that allows manipulation of change propagation and response graphs based on this language. JComposer also supports both synchronous and asynchronous collaboration, as well as smooth transitions between these two styles of work. Other tools that support synchronous sharing of internal artifacts include TelePictive [74] and TeamRooms [84].

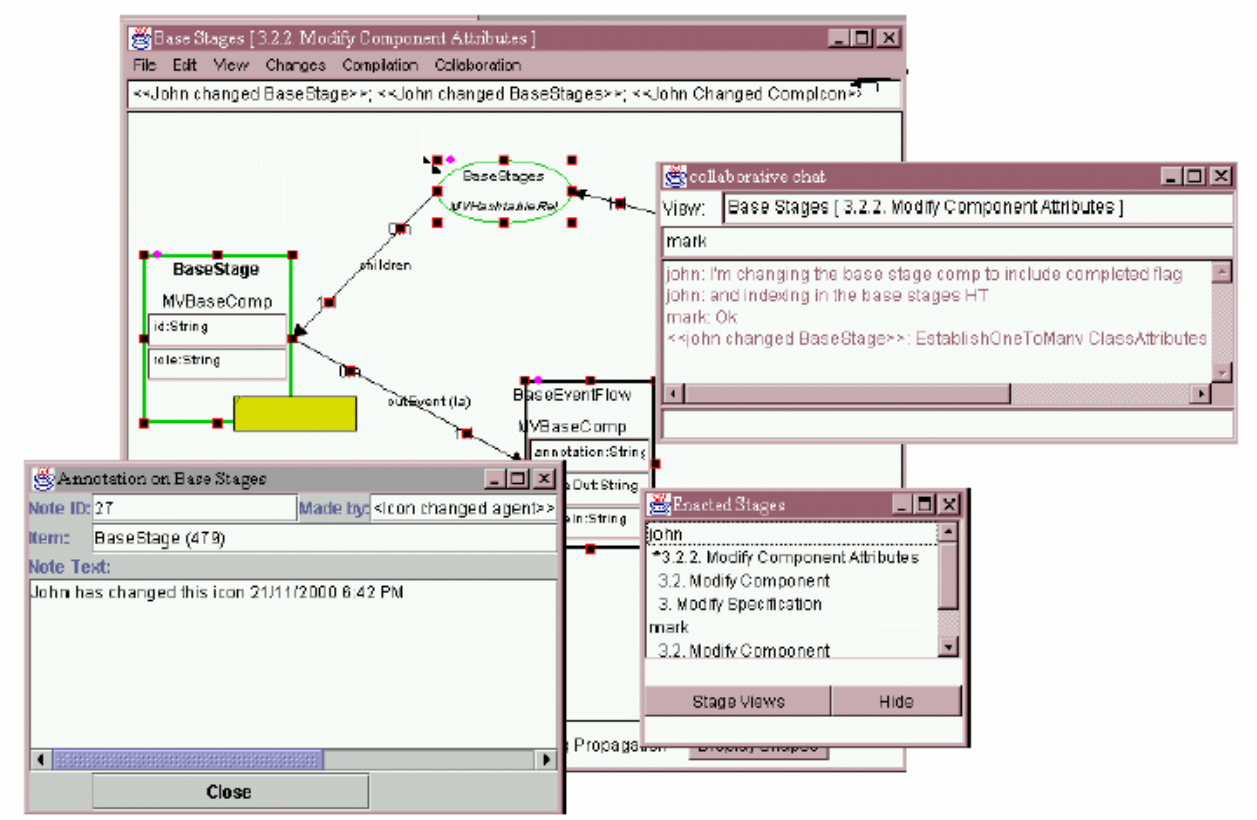

Figure 11: Collaborative component design in JComposer [47]

Insight Lab [65] is tool intended to support the development of design requirement documents by teams (see Figure 15 for Workstyle analysis). Elements of InsightLab are shown in Figure 12, and described in the caption. The development of requirements documents requires analysis of large quantities of qualitative data collected from various sources that reflects the work and the environment in which a system is to be used. This may involve in-depth review of audio and video recordings, photos, or interview notes. This tool facilitates co-located data analysis by teams through the interaction of computer technologies with environmental objects such as papers, whiteboards. Similar tools include i-Land [5,93].

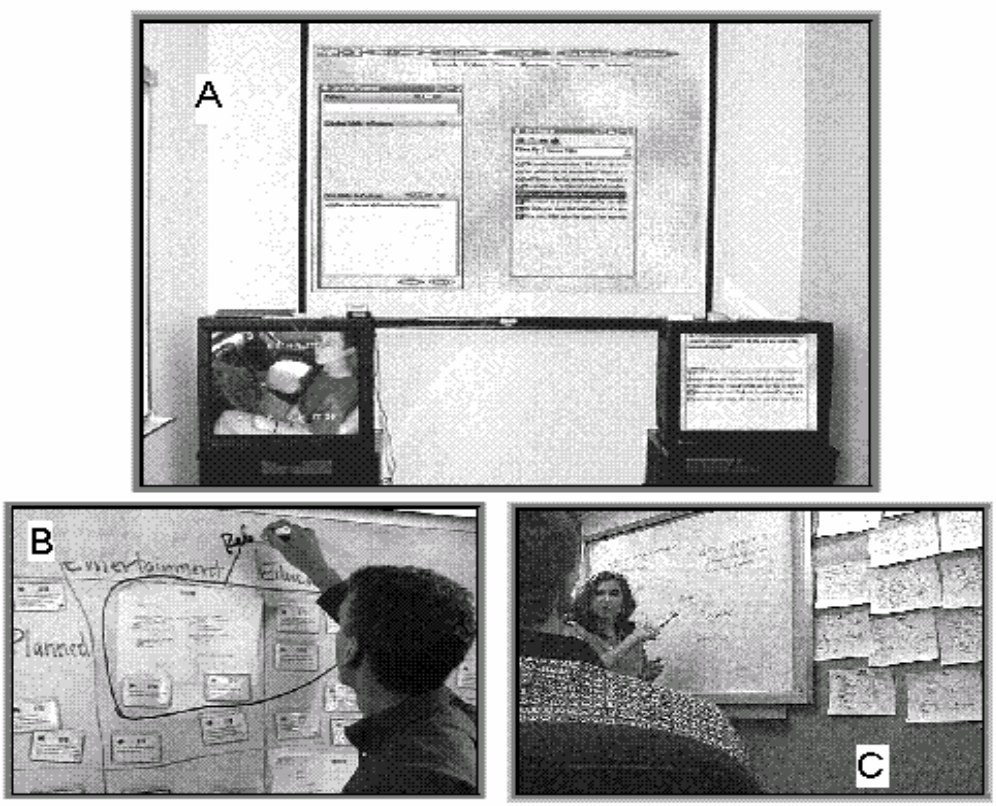

Figure 12 : A) Three screen display within InsightLab. B) Insight Lab walls as information displays. C) An electronic whiteboard surrounded by linked artifacts

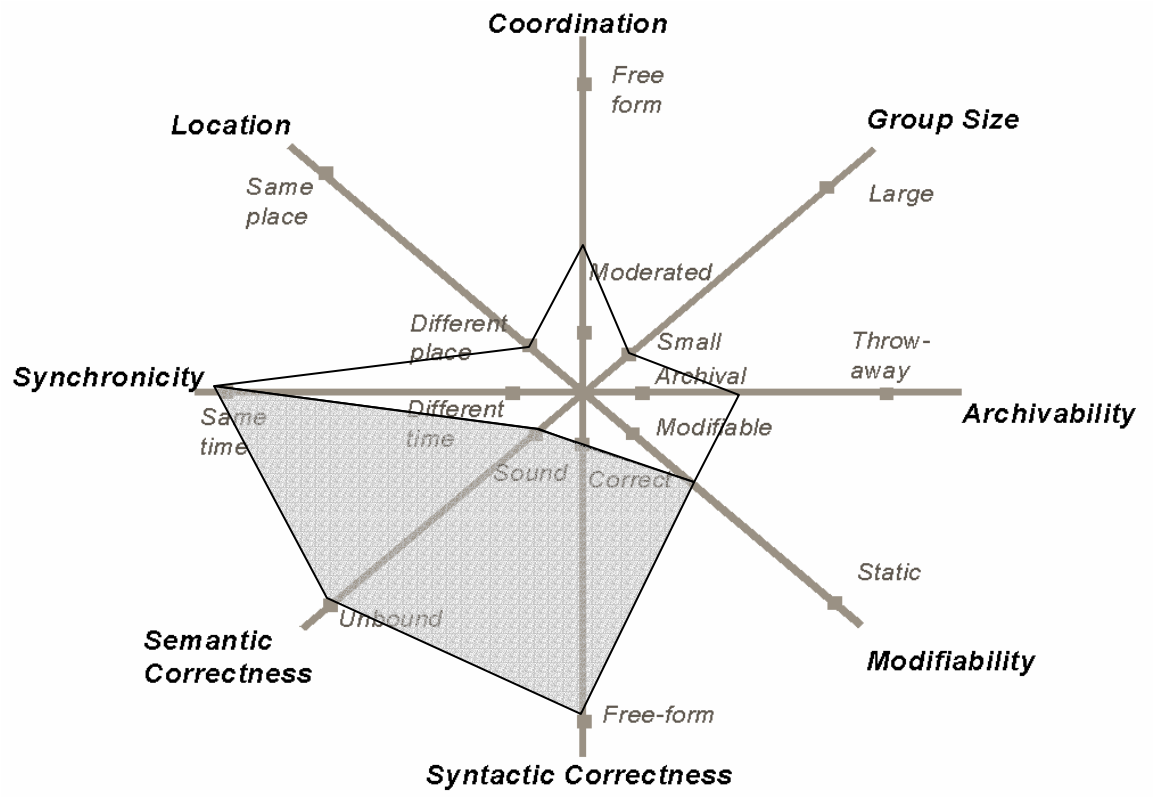

Figure 13: Workstyle Evaluation of Meta-tools

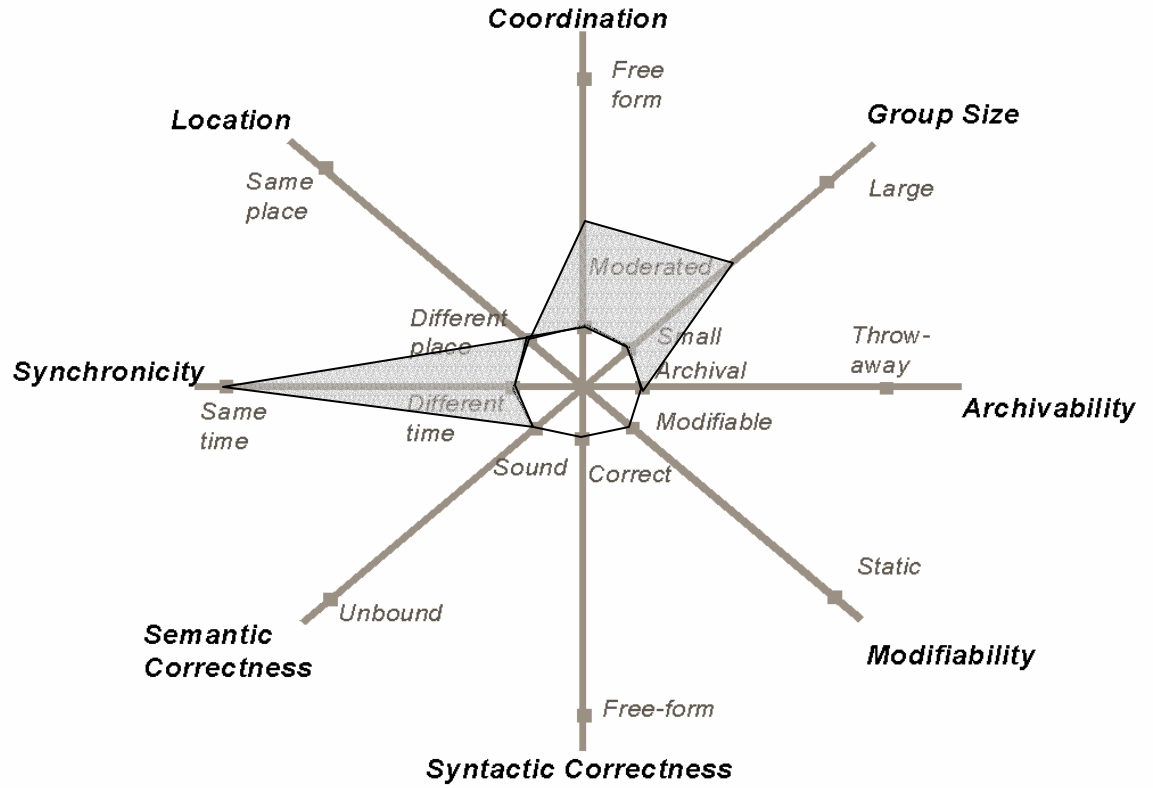

Figure 14: Workstyle Evaluation of Collaboration-aware Tools

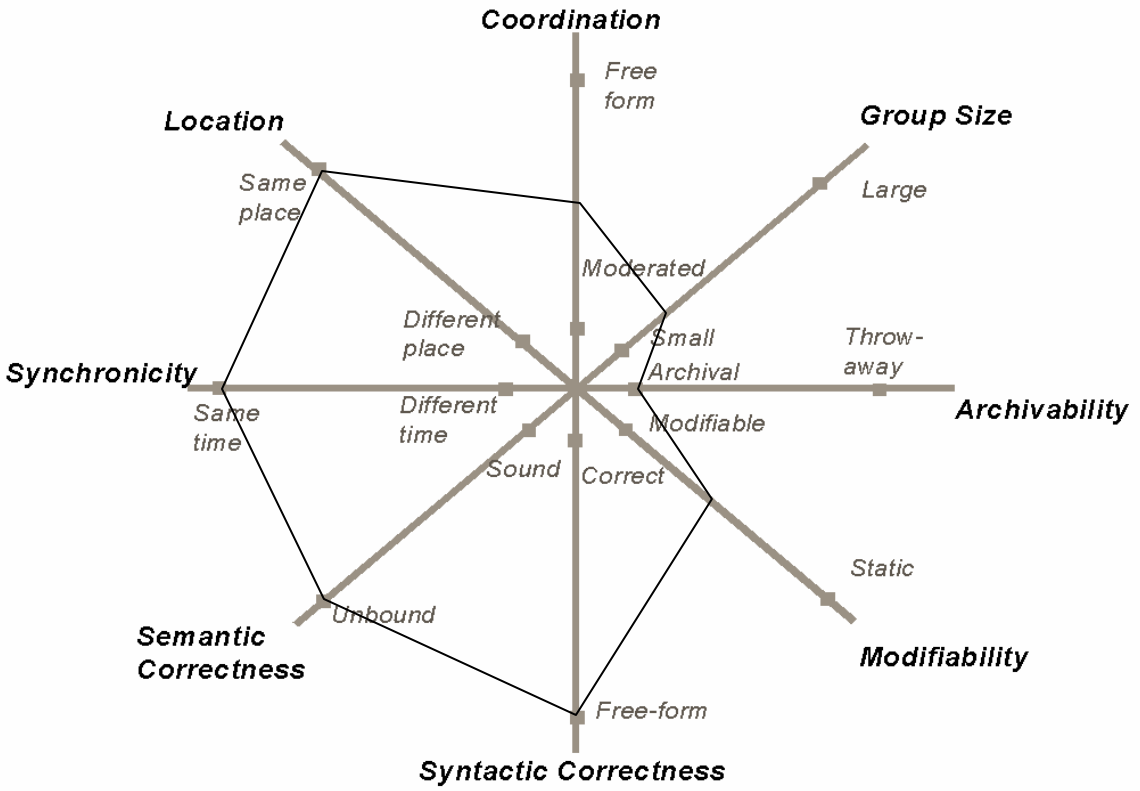

Figure 15 :Workstyle Evaluation of Co-located Tools

5.1.1 Collaborative Design Issues in Synchronous Artifact Sharing Tools

Collaborators often interact with each other through shared artifacts. Artifact sharing provides a common focal point for collaborative activities, as well as a mechanism to record results of collaborative work. For example, two team members responsible for creating a document draft might work together in front of a shared sheet of paper. Such sharing allows each person to interact with the document, as well as witness and modify each other's changes to the document in real-time. They would be able to discuss changes they make, or plan to make, as they are seated next to each other in a common location. Software tools that support synchronous artifact sharing attempt to support this style of interaction with electronic tools, in potentially distributed collaborative situations.

There are three basic approaches taken to support synchronous artifact sharing in collaborative software design. Meta-tools [70,20,10,108,113] facilitate synchronous sharing of software tools that were originally intended for interaction with single users. They rely on the underlying collaboration-unaware tool to manage artifact properties such as archivability, modifiability, syntax and semantics. Furthermore, they manage concurrency issues through coordination mechanisms, such as turn-taking [70,108,20] or locking [47] that prevent concurrent access to the artifact. The mechanisms present a considerable barrier to fluid interaction, as only one collaborator can control the artifact at a given time. Other mechanisms serialize input [10] to convert multiple inputs the single stream expected by the underlying application. This maintains consistency in the shared document. However, issues may still arise if the single-user semantics of the underlying tool conflict with the concurrency control mechanism in place. For example, if a serializing mechanism is used to share a bit-mapped drawing program the results may be unpredictable. The serialized stream of input that reaches the application would consist of interleaved data

from each user, resulting in a continuous scribble across the shared drawing space rather that a set of discrete lines.

Collaboration-aware tools that integrate support for synchronous sharing avoid the conflict that may arise between the single-user semantics of a collaboration-unaware tool and any meta-tool that may be used to synchronously share it amongst collaborators. These tools [47,74,84] support sharing of proprietary artifact types, as opposed to sharing generic, third party artifacts. They integrate the appropriate concurrency semantics and control mechanisms, such as full replication with event broadcasting [43,47,84] to prevent inconsistencies during collaborative editing. Furthermore, these collaboration-aware tools provide higher-level support for synchronous work, such as flexible coupling [47], which permits varing degrees of synchronicity between collaborators, as well as supporting fully de-coupled, asynchronous work [47, 84].

The final approach involves augmented electronic support for co-located interactions such as those described in the motivating example. These tools [65,5] facilitate co-located collaboration through interaction with both computer and environmental objects such as paper or whiteboards. These tools excel in supporting informal communication, as they bring collaborators together an enable rich face-to-face communication, and permit social protocol to mediate interaction rather than imposing control to maintain consistency. Their main drawback is the fact that they cannot support distributed work, as a physical room maintains all shared tools and artifacts. We will now consider the workstyles supported by synchronous artifact sharing tools.

Archivability

Each sub class of tool in this category addresses artifact storage differently. As with all meta-tools, NetMeeting relies on the underlying application to manage saving and restoring documents, as well as any compatibility with third party tools. Because of this, shared artifacts are only archived on the machine on which they were originally created, i.e. the machine of the person running the NetMeeting server. Other tools that support a smaller set of applications also rely on this model for archiving artifacts [20]. This is a barrier to effective collaboration, and has been addressed by tools [108] that provide full local replication of each shared artifact.

Similarly, support for archivability in collaboration-aware tools is dependent on the application. For example, TeamRooms supports collaboration-aware GroupKit [83] applications and relies on them to support archivability for their own artifacts. All objects within a teamroom are fully persistent, meaning that the current state of a collaboration session can also be archived. TeamRooms' persistence database stores arbitrary number of versions of artifacts in rooms, allowing users to browse the history of a document.

Insight Lab is an environment intended to support the storage and management of both electronic and physical artifacts. It maintains relationships between multimedia data, electronic whiteboard drawings and the various analysis elements. Barcodes are used to maintain these links between physical objects such as paper or whiteboards. This kind of functionality is unique in this category, and is useful in early collaborative design situations, where various requirement details may have been collected and stored on heterogeneous media.

Modifiability

Recall that modifiability refers to the ease with which artifacts created using a tool may be modified, either within the same tool or externally in other tools. As before, the modifiability of artifacts created or manipulated through synchronous artifact sharing tools is highly dependant on the artifact itself. Artifacts created by tools that provide a high degree of modifiability maintain the property when shared through meta-tools.

However, coordination mechanisms that are intended to manage concurrent updates can reduce the modifiability of an artifact by certain collaborators during a shared session. For example, a turn-taking coordination policy prohibits modifications to an artifact by anyone other than the holder of the floor [70]. This can reduce the modifiability of artifacts by groups because requests for control must be made before

each user can make a modification. To avoid this issue, some tools use more sophisticated protocols, such as Flexible JAMM [108], COMET [20], or partial locking [47] for managing concurrent input. Others place no restrictions at all, relying on social protocol to resolve conflict [65,47]. These approaches allow each user to modify the shared artifact with reduced concurrency-based interaction barriers.

Some tools also implement trace history and recording mechanisms to provide full change histories for artifacts created or manipulated collaboratively through the tool [20,84,65]. This functionality allows collaborative changes to an artifact to be logged such that modifications may be undone, applied to other artifacts in similar fashion, or recorded for documentation purposes.

Syntactic correctness

Syntactic correctness refers to the requirement that an artifact conform to a precise syntax. For most synchronous artifact sharing tools, any syntactic restrictions that may be imposed are done so by the underlying application. Using application-sharing tools [70,20,108] to share a Rational Rose diagram amongst designers imposes no additional syntactic requirements beyond Rose itself. Similarly in TeamRooms, syntactic restrictions are imposed only by the underlying collaborative applications within a room. A room itself has no notion of syntax; objects and applications of mixed type may be brought into, and freely arranged within a room. This freedom from syntactic requirements can allow designers to express their ideas in the manner, and with the tools, of their choosing.

Insight Lab analyses can contain data from a wide variety of sources. Data sources may include video, audio, photographs or other data. However, each analysis is composed of 4 distinct elements – Evidence, Patterns, Hypotheses and Themes. There is no particular syntax applied to entries within these elements, though their content is based on the particular data source and tagged with a set of keywords to facilitate categorization and analysis. As Insight Lab is intended to facilitate analysis of large quantities of data, this mild syntax helps to structure the data in such a way as to emphasize patterns without hindering the designers in any significant way.

JComposer is entirely different from the rest of the tools in terms of syntactic requirements. Like other tools that support software modeling (see section 2), JComposer enforces syntax by nature of its structured editor – the primitives provided by the editor restrict the possible syntax of the artifact. In this case, the editor provides primitives for the *JComposer Architecture Description Language* [47], which is based on Change Propagation and Response Graphs [45]. CPRGs propagate change description objects between component objects via relationship objects. These elements constitute the syntax supported by this tool. This requirement to use a structured syntax may impede informal or creative design work [111], making JComposer less appropriate to early stages of design.

Semantic correctness

Semantic correctness refers to the clarity and precision with which the meaning of the artifact must be expressed. Again, for most of the tools, any semantics that may be interpreted or assumed are done so by the underlying application.

Again, JComposer is different from other tools in this category because its artifacts are intended to be expressed with such clarity and precision so as to facilitate code generation. Based on the CPRG [45] model, change description events are generated whenever components or relationships are modified. These event representations are propagated to associated relationships or components for an associated reaction. The reactions are used to implement semantic constraints on the environment. Some component types have predefined actions that are performed on receipt of specific types of event. In general, however, application specific actions must be defined to specify the semantics design.

Synchronicity

Synchronicity refers to any the temporal restrictions that a tool may impose on interactions between collaborators. As the category itself suggests, all of the tools support fully synchronous interaction with artifacts. However, the tools vary greatly in how they manage synchronous interaction, as well as in the support they provide for synchronous collaboration beyond artifact sharing.

Even among tools that provide synchronous artifact sharing through a screen sharing mechanism [70,20] there are a variety of approaches to synchronization. Netmeeting uses a coarse-grained floor control policy that requires users to request and be granted control of the shared application before they may manipulate the artifact, while CECED applies a more sophisticated distributed floor control policy. This algorithm grants the floor to the local site only if no activity is detected from the remote conference site that is holding the floor, without the need for a formal request mechanism. However, both of these mechanisms actually prevent multiple users from interacting concurrently with an artifact, and may become cumbersome during group interactions.

Other tools use approaches such as JAMM [10] that serialize multiple input streams to present a single flow of control to the collaboration-transparent application [108]. Similarly, collaboration aware tools [84,47] can use even more sophisticated concurrency mechanisms that are based on views of replicated objects. Each participant maintains their own copy of a shared artifact, and changes made to any of these copies are propagated to all of the others. Individual views of the shared artifact are updated as changes are applied. These approaches avoid the disruptions that can result from concurrency policies that prevent concurrent interaction, and allow all users equal access to the artifact.

In JComposer, inconsistency management may be left to the individual designers. Depending on the level of collaboration, which will be discussed shortly, changes may be presented to the user before being applied, and changes that cannot be applied can be detected and marked invalid. Additionally, fine-grained locking policies may be applied to prevent inconsistencies from arising. This allows flexibility of choice as to when or if changes made by others are applied to individual views, and provides tolerance to inconsistencies between views.

Some tools support also support asynchronous collaboration, and provide mechanisms for easy transitions between synchronous and asynchronous interactions [84,47]. TeamRooms supports asynchronous work through the notion of persistence. Objects placed in a room persist even when they are not being shared by anyone. This allows users to interact with objects, and each other, either synchronously or asynchronously with no difference in the way they use the tool. Transitions between synchronous and asynchronous work amount to users leaving and re-entering a room. JComposer provides five levels of collaboration as motivated by Suite [27,28] – asynchronous, notify, present, action and synchronous. They differ in the way in which individual views respond to changes, from no response (asynchronous), through various mechanisms for presenting changes to the user for selective application, to use of a locking protocol to prevent inconsistent changes from being applied (synchronous). Users may move freely between levels, and are not all required to be at the same level, even when sharing the same artifact. This increases work style flexibility by allowing individual users to decouple from the group, allowing people to collaborate in different ways at different times. For example, a designer may want to interact synchronously with others when connected via a high-speed, high-reliability network, and asynchronously when connected through a low-speed, low-reliability network, e.g. when traveling and connecting through a dial-up internet service provider.

Beyond the ability to synchronously share artifacts, many of these tools provide additional support for collaboration, for example communication facilities or awareness mechanisms. Voice and video connections [70,20] and textual chat tools [70,84] provide natural communication mechanisms between participants. Tools such as shared whiteboards [70,84,65] facilitate lightweight interaction and brainstorming. Similarly, telepointers [70,84,20] provide awareness cues about others who are interacting with an artifact. TeamRooms goes futher still and provides awareness information about participants currently collaborating in a room in the form of still images, as well as radar overview of the room, which tracks both users attention and room objects.

Location

The location axis describes the kinds of geographical distribution of collaborators that a tool can effectively support. Generally, synchronous artifact sharing tools only support distributed collaboration [70,20,108,84,47]. They are intended to replace face-to-face interaction when such meetings are not feasible. The user interfaces for these tools are designed for a single user interacting with others using the

same application in remote locations, rather than a group of co-located users interacting with a single computer or application. The exception is Insight Lab, which can support co-located collaboration only. It is an physical environment that integrates multiple devices and objects into a single system that supports simultaneous interaction from co-located users. Team members interact while working on the same artifacts, in the same room, using the same devices.

Coordination

Though some applications [85] may add higher-level coordination policies, the coordination of collaborative activities within these tools is typically determined by the tool's concurrency control mechanism. For example, tools that employ turn-taking coordination mechanisms and require formal requests to be made and granted impose a significant degree of coordination on collaborators. More sophisticated policies can mitigate this coordination to varying degrees. The distributed floor control policy employed by CECED relieves users of submitting requests for control; however, it still limits interaction to one participant at a time. Serialization mechanisms such as JAMM allow truly concurrent interaction with an artifact while still maintaining consistency and respecting the single-user semantics of the underlying application. Collaboration-aware application such as TeamRooms and JComposer minimize coordination restrictions further still, but require the use of a limited number of predetermined applications [84] or proprietary design languages [47]. However, JComposer tolerates inconsistencies between individual views, which serves to further reduce the degree of coordination between designers. Co-located tools such as Insight Lab coordinate collaboration by requiring collaborators to be co-located, which may be both inconvenient and impractical.

However, as these tools [70,20,108] allow sharing of collaboration-transparent applications, the semantics of these single user applications may become an issue within a multi-user context. For example, DISCIPLE is based on a replicated architecture, wherein all participants have their own copy of any application used in collaboration. Consider a situation where DISCIPLE is used to share a software modeling application, such as Rose, amongst multiple designers. Designers would have their own copy of both the modeling application and the artifact being shared. At such a point as the shared artifact is to be checked into the team repository, multiple copies of the same document may be stored because the underlying application not aware of the semantics of collaboration. A user awareness of this kind of semantic mismatch would be required to avoid such issues when using synchronous meta-tools in this manner.

Group size

Generally, synchronous artifact sharing tools will only support small groups of synchronous collaborators. Voice and video communication are often point-to-point [70], rather than multicast, which can limit group size, though CECED does support multicast voice conferences. Additionally, concurrency control based on a floor control mechanism [70,20] will limit the number of collaborators in practice, as competition for control of the floor increases with group size. Similarly, concurrency mechanisms based on serialization [108] may begin to impact application responsiveness, particularly as groups grow larger. Co-located tools such as Insight Lab are limited by the physical size of the room in which they are installed.

JComposer is theoretically limited only by the granularity of the locking mechanism for synchronous interaction, though certain considerations will limit this in practice. Artifacts containing fewer elements will be able to support fewer concurrent collaborators. In more asynchronous collaboration modes, it may support larger groups. However, the various concurrency mechanisms for asynchronous collaboration may prove insufficient for large-scale collaboration. Presentation of all remote updates may overwhelm an individual user as group size grows, and manual merging of divergent versions will become increasingly difficult.

5.2 Summary

Meta-tools support collaborative software design by enabling synchronous sharing of collaboration design applications. As with tools specific to asynchronous artifact sharing, these tools support a fairly limited set of workstyles. While they are independent of the type of artifact being shared, they are limited to supporting small groups of distributed collaborators interacting synchronously. Additionally, these restrictions as well as any concurrency control mechanisms may impose significant restrictions on how designers may interact with each other and the shared artifacts.

Collaboration-aware tools integrate synchronous sharing facilities with software design applications. These tools have the similar restrictions in the nature of the workstyles they support as the software modeling tools described in category 1. They impose the same restrictions on artifact properties, as well as supporting only a small or medium group of distributed collaborators. However, unlike the tools of category 1, they may allow both synchronous and asynchronous interaction.

The remainder of tools in this category support synchronous sharing of artifacts amongst co-located collaborators through integration of computers with environmental objects. These tools support a fairly specific workstyle. Small groups of collaborators must be co-located, and interact synchronously. This imposes a certain degree of coordination, though beyond this interaction between collaborators and with artifacts is mediated socially.

6 Category 5: Tools Supporting Interaction through Informal Media

People often carry out design work using informal media such as paper or whiteboards [64]. Particularly in the early stages of design, informal media are appropriate as they allow design diagrams to be quickly and fluidly sketched [107]. Computational analogues of such informal media include electronic whiteboards, data tablets and stylus input for computers. Tools supporting interaction with informal media attempt to extend the free form, fluid interaction afforded by physical informal media to these computational counterparts.

The tools in this category are defined by their support for user interaction with informal media. This means that they support unstructured interaction through devices such as whiteboards, tablets or paper. This contrasts with more formal styles of work that emphasize structured interaction mechanisms such as structure editors, for example PowerPoint [71] or text editors such as Word [73]. Three sub categories of tools will be considered:

- *Informal CASE Tools:* These are software design tools that support interaction through informal media. For example, IdeogramicUML [53] is a commercial tool that supports UML modeling based on interpretation of hand gestures rather than a structured UML editor. IdeogramicUML evolved from the research tool Knight [24].
- *Enhanced Electronic Whiteboards:* These are electronic whiteboard applications that attempt to replicate and extend the functionality of physical whiteboards using electronic whiteboards such as a Smartboard [89] or Mimio [106]. Tools such as Tivoli provide computational augmentation to standard whiteboard functionality, and allow informal stroke data to be manipulated in the same fashion as objects in typical structured graphical editors. Other similar tools [75] allow for the application of predefined behaviors to such data. For example, textual lists can be treated appropriately, and support list-appropriate manipulations insertion, deletion, reordering, etc.
- *Shared Drawing Tools:* These tools support collaborative sketching or drawing tasks such as often found in early design work [110,100,97,55] without providing support for any specific notation. Shared drawing tools provide functionality to support collaborators synchronously sharing a drawing surface.

The main advantage to informal media tools is that they support a natural working style without imposing significant cognitive overhead on the user by means of heavyweight interaction mechanisms. They allow users to use the tool transparently, without having think about the tool itself. The drawback in many of these tools is the limited, or non-existent, support for movement towards more formal, structured work, and may limit development as a design progresses begins to require more formal treatment. Also, many of these tools have general-purpose intents, and are not specifically designed to support early software design tasks.

6.1 Collaboration in Tools Supporting Interaction Through Informal Media

As an introduction to this category of tools, we will describe IdeogramicUML, Flatland and Clearboard as archetypes for each of the three subclasses described above. The differences amongst tools will be discussed using the Workstyle model. A graphical depiction of this analysis is show in Figure 19.

- *Informal CASE Tools:* Ideogramic UML [53] is a commercial tool that evolved from the Knight research project [24]. An example of this tool in use is depicted in Figure 16, which shows how both formal and informal design notations can be used to express a UML design. IdeogramicUML is intended to support the “agile” use of UML [2], meaning effective and lightweight use of UML. It supports a wide variety of interaction

devices, including PCs, tablets, Tablet PCs and electronic whiteboards. This tool supports gesture based modeling in UML, as well as free hand diagramming with no gestural interpretation. IdeogramicUML supports a subset of the UML standard limited to Class, Use case, Sequence and State Chart diagrams. Furthermore, IdeogramicUML only supports co-located collaboration using electronic whiteboards, and requires additional tool support to be used by distributed teams. Other similar tools include UML Recognizer [66] and Tahuti [48].

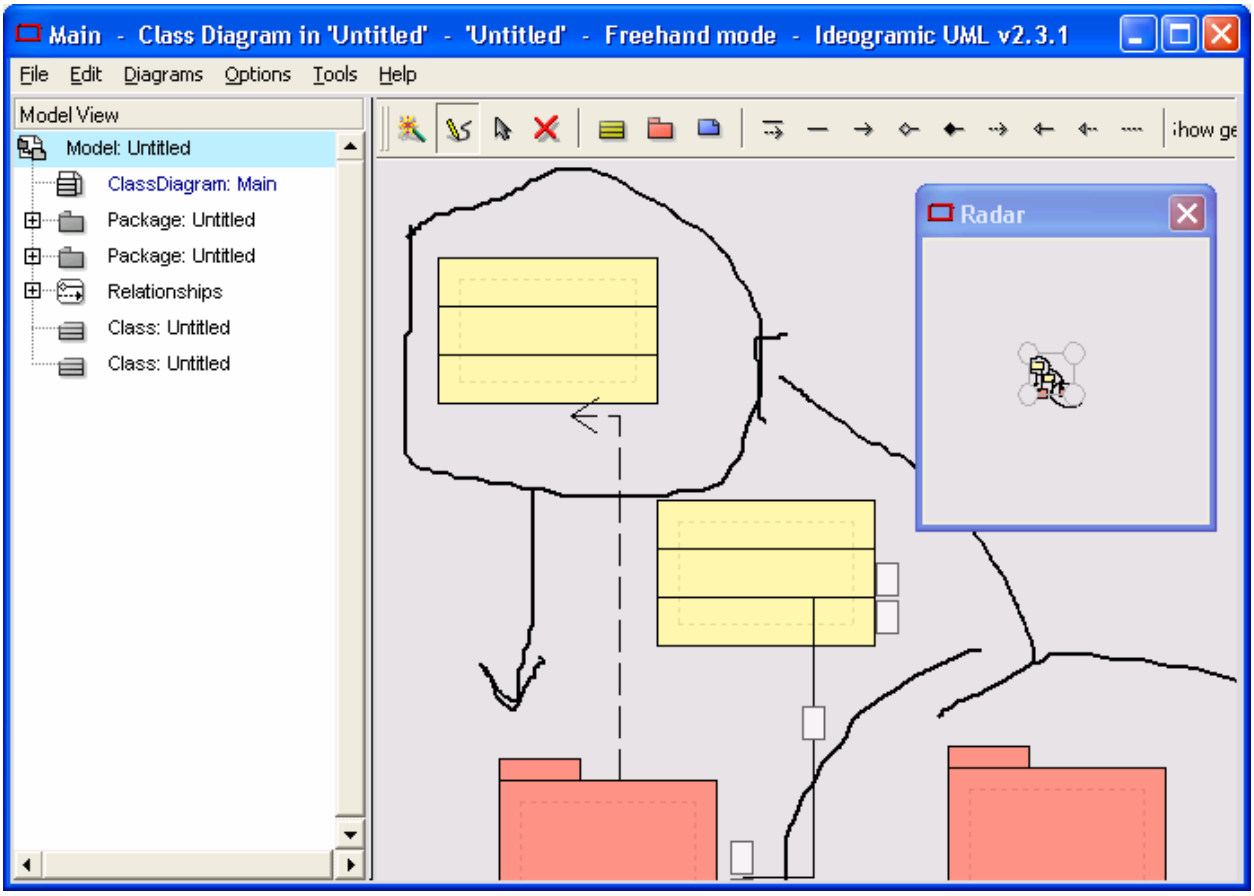

Figure 16: Preliminary designs mixing formal and informal notations in IdeogramicUML

- *Enhanced Electronic Whiteboards:* Flatland [75] is an augmented whiteboard application designed to support informal office work. Flatland provides various stylus-appropriate techniques for interaction and space management on an electronic whiteboard. Furthermore, it provides the ability to apply different behaviors to define application semantics. This is shown in Figure 17, where each segment on the board responds to differently to stylus input based on the applied semantics. Flatland can also manage the change history of the board space. However, it does not specifically support design tasks, but is intended to support for informal work in an office environment and as such can be appropriate in early software design tasks. Furthermore, Flatland does not support distributed collaboration, it only facilitates teamwork in a co-located setting. Other similar tools include Tivoli [79], Dolphin [92], MagicBoard [21], and IdeaBoard [1].

Figure 17: The Flatland user interface [75]

- *Shared Drawing Tools:* ClearBoard [55] is a shared drawing program that allows two remote users to simultaneously draw in a shared space while providing awareness information such as hand gestures and gaze. It is based on the metaphor of 'talking through, and drawing on, a big transparent glass board' [55], and implements a shared drawing technique drawn from VideoDraw [97]. Figure 18 demonstrates this technique more clearly. Clearboard also provides additional functionality such as simple stroke manipulations, recording of working results, as well as the ability to integrate generic files into the drawing space. Unfortunately, ClearBoard is implemented using experimental hardware that is not widely available, and is limited to support for only two users. Other similar tools include Commune [14], GroupSketch [42], Mediaboard [105], WeMet [110], as well as VideoWhiteboard [100] and Videodraw [97].

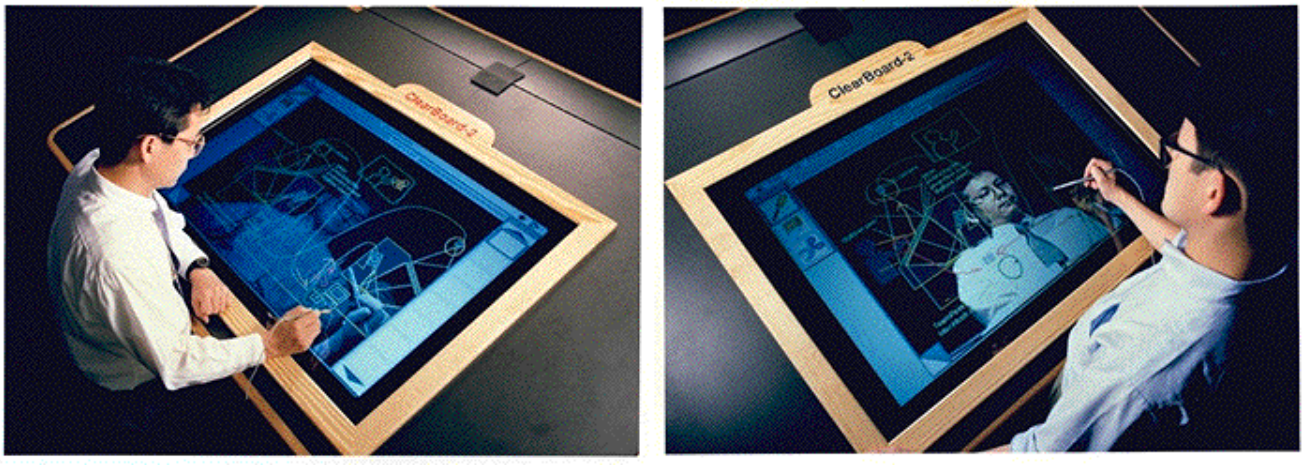

Figure 18: Clearboard-2 prototype in use [55]

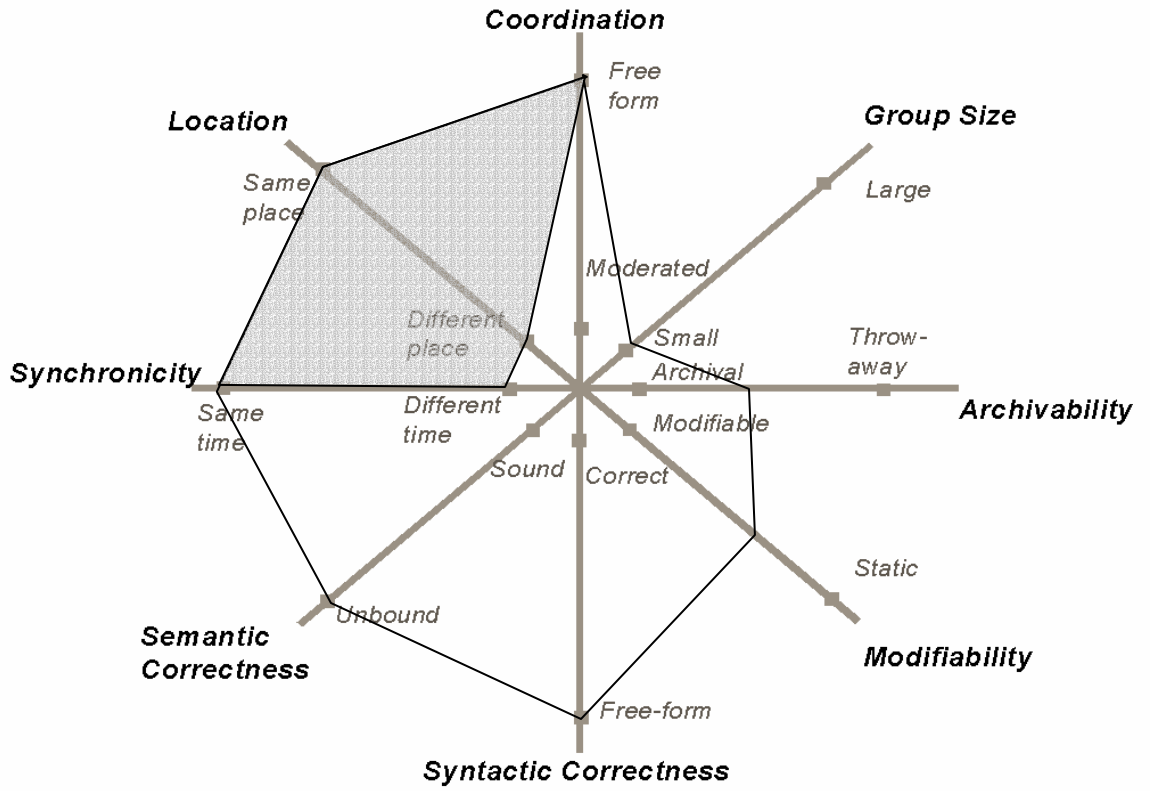

Figure 19: Workstyle Analysis of Tools Supporting Interaction through Informal Media

6.1.1 Collaborative Design Issues in Tools Supporting Interaction through Informal Media

Informal media such as whiteboard and paper have been shown to be effective tools for cooperative design [95,96,99,12,23,24]. They support simple, direct interaction without unnecessary overhead, and allow several designers to work simultaneously, facilitating co-located collaboration. Furthermore, they do not impose any special notation on the user, thus supporting the production of both formal and informal diagrams, as well as the use of flexible and extensible notations. Software tools that support interaction through such media attempt to combine these benefits with those of traditional software tools. How this is accomplished varies between tools, however it may involve augmented editing and stroke manipulation techniques [75,79,92,21,1], support for distributed collaboration [55,110,100,97,92,14,105,42] or some integration of application-specific semantics [53,24,66,48,75]. The nature of these application semantics varies, ranging from software design specific (e.g. UML notation) to more generic tasks (e.g. To Do lists).

However, these tools have limitations related to both the artifacts they can produce as well as the way in which they support interaction between collaborators. The lack of syntactic restriction and limited application semantics can limit expressiveness and increase ambiguity in complex, formal designs. Furthermore, lack of defined syntax and semantics often implies the use of image-based file formats (e.g. bmp, gif, etc) instead of structured formats based on accepted standards (e.g. XMI). This serves to limit archivability and interoperability with other design tools. Additionally, unstructured editing mechanisms such as found many of these tools lack much of the functionality of their structured counterparts. Complex editing functions that are common to structured editors are much less common in these tools, leading to poorer modifiability of artifacts produced.

Interaction between collaborators is also influenced by the use of these tools. While all of these tools support synchronous, co-located interaction, few provide support for distributed, synchronous collaboration, and those that do are very narrowly focused on that task. For example, the tools that provide application semantics specific to software design [24,66,48] provide no support for distributed work. Those that do support distributed collaboration [55,110,100,97,92,14,105,42] integrate no notion of application semantics, and do not provide much of the functionality typical of augmented whiteboard applications. Another issue that impacts collaborator interaction is the small group size that these tools can support. The lack of coordination models, as well as the nature of informal media itself, limits the effective size of the group that may collaborate through informal tools. Some [55,97,110] only support synchronous collaboration between two people.

A final issue concerns the hardware available for supporting interaction with electronic informal media. While it is the intention of these tools to support natural and fluid interaction in a manner similar to that supported by physical informal media, the ability of the hardware involved falls short for achieving this goal. Issues arise concerning latency, resolution and synchronous input. For example, unlike a physical whiteboard, user's of a SMARTBoard [89] find a delay between drawing a stroke and having it appear on the drawing surface. Similarly, a SMARTBoard can only accept input from a single user at a time, while a physical whiteboard can be used by as many people as can reach at a given time. Finally, any tool such as electronic whiteboards, styli and data tablets will be limited in resolution by the nature of the input and display mechanisms.

Archivability

Like most modeling tools, Knight/IdeogramicUML provides complete support for saving and restoring native artifacts, however it does not integrate a multiuser artifact repository or revision control system. Furthermore, it supports data exchange with XMI-compliant tools including Rational Rose [81] and ArgoUML [7], however free-hand information is not exchanged because it is not defined in the UML XMI specification [77]. Knight/IdeogramicUML also supports output of HTML and SVG documentation, as well as JPEG, PNG and SVG image types for diagrams. The other modeling tools in this category [66,48] are less extensive in their support for archiving artifacts, as they are research prototypes rather than commercial tools. However, both do support saving and restoring of native diagrams, as well as data exchange through XMI with Rational Rose. Data exchange from informal CASE tools to more typical CASE tools is important in order to support movement from an informal workstyle to a more formal style as designs develops.

Augmented whiteboard applications typically support archivability of their whiteboard space through the notion of persistence. This means that data placed on a board persists until erased. To facilitate data storage and board space management, Flatland integrates a document respository, and each segment of the board is automatically saved without explicit user action, and identified by its surrounding context on the board. Others implement more explicit mechanisms requiring user initiation and input [79,92,21,1].

Like whiteboard applications, shared drawing programs also support the notion of persistence, and some [55,105,110] provide file-based storage in variety of formats. Most of these formats are image-based, though some provide native file formats to facilitate reuse in future work [105]. Additionally, tools may support session history recording to allow the previous state of a session to be recalled [105,110]. However only those tools supporting native file formats can allow a previous state to be recalled and reused in the same manner in which it was originally created. Furthermore, shared drawing programs do not structure data in such as way as to facilitate exchange with typical CASE tools. This limits collaborators' ability to move into more formal workstyles as such workstyles become appropriate. Finally, some tools intended to support shared drawing provide no file based storage [97,100], though video images may be recorded and used for archival purposes.

Modifiability

There are three levels of modifiability amongst tools in this category. Tools that support input with physical rather than electronic ink [55,97,100] all support basic modifications through traditional

means. Physical ink may be erased wholly or in part, by any convenient means (finger, palm, cloth, etc.), and does not require a special input device such as a stylus. This style of interaction is well suited to informal or creative design, as interaction with the tool becomes transparent. However, physical ink does not support augmented manipulations such as move, copy or resize. Tools that support input with electronic ink may sacrifice some interaction transparency to provide such augmented manipulations [24,75,92,79]. Some support both physical and electronic ink [21] and provide the advantages of both styles of input. Conversely, some tools strike a balance between functionality and interface simplicity and support only basic manipulations of the electronic ink strokes, such as delete [110,14,42].

Syntactic correctness

Informal media tools do not typically restrict the stroke alphabet that may be used to create artifacts. This means that they do not enforce a particular syntax; all support completely informal interaction, and impose no restrictions. However, those tools that support some application semantics [53,24,66,48,75] do require appropriate syntax for this functionality. For example, tools that use gesture recognition, such as Knight, Tahuti and UML Recognizer, are based on particular properties of the input strokes. Failure to make strokes that have such properties can result in mis-recognition of the gesture. Similarly, Flatland requires certain properties to be present in the input stoke set in order to successfully apply particular behaviors. A lack of strict syntax is one of the fundamental properties of the tools in this category that facilitate lightweight group collaboration, particularly in the context of early design.

Semantic correctness

None of the tools in this category ascribe semantics to the artifacts produced. Even those providing functionality based on application-specific semantics [53,24,66,48,75] do not impose a semantic model on input, though a particular semantics are applied at the option of the user. For example, gestures in Knight may be interpreted as UML input, or simply stored as uninterpreted stroke data depending on the user's choice. Similarly, segments in Flatland may have particular behaviors applied. For example, segments containing list data can be automatically formatted and manipulated as structured list. However, any such behavior is applied at the user's discretion, and no behavior is automatically applied to any stroke. The flexibility afforded by informal media tools with respect to semantic correctness allows the expression of design where the semantics are unclear. This is common in early design stages, as many details are not yet defined.

Synchronicity

In general, informal media such as whiteboards and paper support concurrent interaction between multiple people, and software tools that support interaction through such media are no different. All of these tools support synchronous interaction, either at a distance [97,110,14,42], or co-located [24,48,66,21,1,79,75], or in combination [55,100,92]. The nature of these tools is also such that they support asynchronous interaction through the notion of persistence. Information left in the shared drawing space can persist over time, allowing collaborators to interact with the same artifact at different times.

However, there is variation in the degree of support for distributed, synchronous interaction amongst those tools that permit it. Dedicated shared drawing tools [97,110,14,42,55,100] all provide support for distributed collaboration with such mechanisms as telepointers [42], hand gestures [97,100,110], gaze awareness and eye contact [55].

Location

Because of the nature of informal media such as whiteboards and paper, all of these tools can support co-located collaboration in an asynchronous manner. In addition, these tools support may either distributed interaction [97,110,14,42], co-located interation [24,48,66,21,1,79,75], or both [55,100,92].

Coordination

The coordination of collaborative activities within these tools is controlled solely by social protocol. None of the tools place any restrictions on how or when participants may interact with each other or the artifact that they are creating, beyond any imposed by specific hardware devices. For example, whiteboard based applications [79,75,92,21,1,100,55] may use different kinds of electronic whiteboard hardware that may allow multiple, concurrent streams of input [79,92], or restrict input to a single stylus at a time [75,66,24]. This lack of enforced coordination is another fundamental property of informal media tools that facilitates lightweight group collaboration, particularly in the context of early design.

Group size

Generally, informal media tools only support a small number of collaborators. Whiteboard based tools [79,75,92,21,1,100,55] can support synchronous interaction amongst co-located users, but are limited to supporting a small group due to their physical size. The tools that add support for distributed collaboration [110,14,42, 92] rely on social protocol to mediate interaction, which begins to break down with larger groups. Similarly, communication mechanisms designed to support distributed interaction, such as telepointers, gestures, eye contact etc., do not scale well to larger groups. Finally, specific hardware designed for some shared drawing tools [97,100,55] may only support two synchronous collaborators. However, asynchronously, all these tools could potentially support larger groups. For example, team information is often recorded on whiteboards for long-term storage and easy access.

6.2 Summary

Tools supporting interaction through informal media support collaboration in software design by facilitating unstructured interaction in a way appropriate to the early, creative design stages. They support an informal workstyle that allows users to interact naturally and to use the tool transparently without imposing unnecessary overhead. Informal media tools support a small group of designers, and rely on social protocol to mediate group interaction. They typically produce informal artifacts of unbound semantics and free-form syntax. Furthermore, informal media tools are independent of synchronicity or location, i.e. they support synchronous and asynchronous, as well as distributed and co-located interactions.

7 Category 6 : Communication Tools used to Support Collaboration

Collaboration in distributed groups, particularly those separated in both location and time, is more difficult than in physically co-located groups [4]. All forms of distance communication require tool support, such as telephone or email. Additionally, physical proximity has been shown to foster and maintain working relationships [62] by facilitating lightweight, informal interaction. Providing a sense of awareness and proximity is equally important in distributed situations, and requires further tool support.

This category addresses communication tools that are used to support collaboration in software design. These tools support team communication and awareness, but have no functionality specific to software design. They support collaboration by enabling communication, both synchronous and asynchronous, and/or providing awareness information that can be used to support informal, or impromptu interactions. Additionally, they provide a sense of cohesion and proximity to distributed teams. Examples of these tools include media spaces [13] such as Montage [98,101], Piazza [56], Portholes [31], Telefreek [18] and CoMedi [19], as well as more common communication tools such as telephone, videophone (e.g. Netmeeting), email, real-time chat, discussion groups and instant messages.

Communication tools facilitate collaborative design by providing the basic mechanisms required to allow distributed teams to communicate, whether through sound, video or text. Additionally, some try to supply some of the contextual social cues from co-located collaboration that are lost in distributed settings, thereby providing a more natural environment in which to collaborate.

7.1 Collaboration in Tools Supporting Team Communication

This category addresses two different kinds of tool, media spaces and common communication tools. A graphical depiction of the Workstyle analysis of media spaces is shown in Figure 21. As an introduction to the class of tools, we will look at Montage as an archetype of media spaces. See Figure 20 for a depiction of an example mediaspace. As communication tools are sufficiently common so as not to require such an introduction, they will be addressed individually in the context of the Workstyle model analysis that follows this section.

- *Mediaspaces:* Montage [98,101] is a prototype mediaspace that uses video to help collaborators find opportune moments for interaction. Montage is intended to provide a sense of teleproximity. Montage tries to emulate the opportunities afforded to physically co-located team members who are able to walk down hallways and peek into offices to determine whether someone is available. The system uses a desktop videoconference tool to support 'momentary, reciprocal glances among networked workstations' [101] that allow users to peek into another's office and, if appropriate, start a videoconference. If the moment is not appropriate, Montage provides access to scheduling information, and alternate communication tools such as email or an instant message. Other similar tools include Piazza [56], Portholes [31], Telefreek [18] and CoMedi [19].

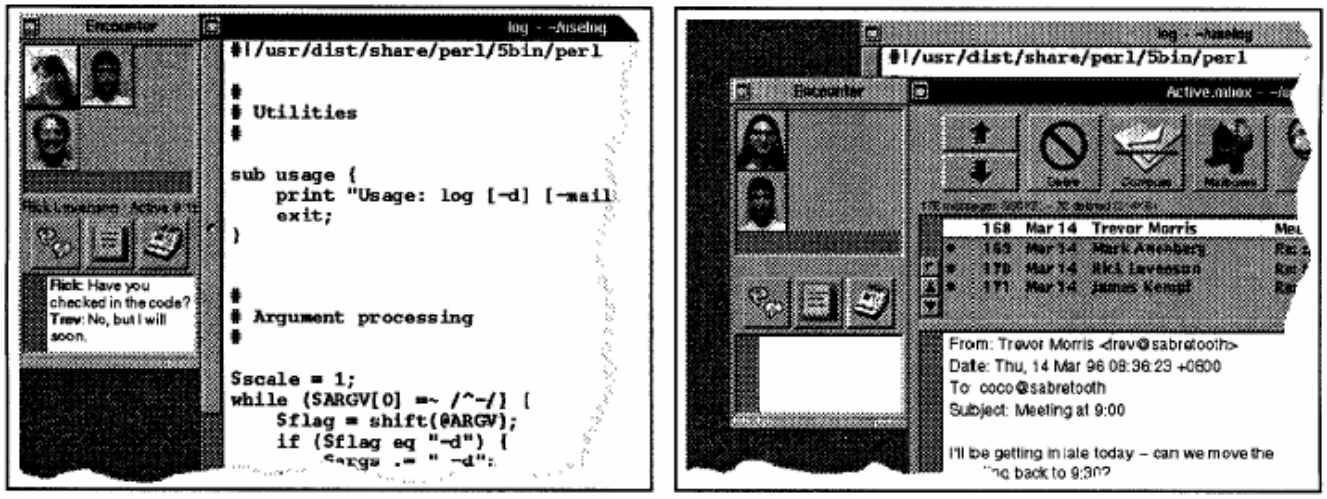

Figure 20: Piazza as an example of a mediaspace. On the left, a user opens a document to find two others also viewing it. On the right, a user moves to a mail message to find another already viewing it [56].

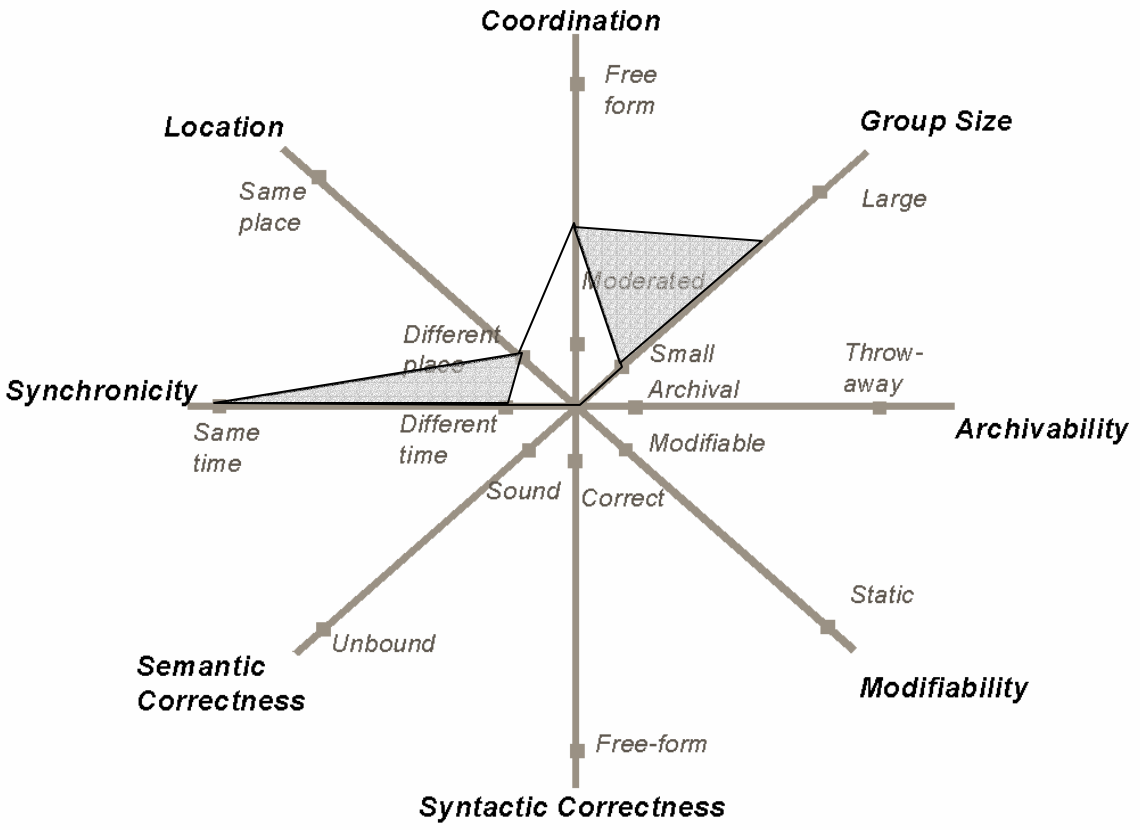

Figure 21: A Workstyle analysis of media spaces

7.1.1 Collaborative Design Issues in Tools Supporting Team Communication

Communication is very important in collaborative software design. Discussion among software engineers is correlated with progress in design [17], as well as cited as the most important of eighteen ways in which teams coordinate themselves [63]. Furthermore, opportunistic and spontaneous

conversations constitute over half of workplace interactions [62], while [109] found that 92% of observed interactions were not pre-arranged. This means that most interactions were unanticipated by one or more of the parties involved, as well as opportunistically instigated. When groups are co-located, people often 'run into each other', which facilitates such interactions. At a distance, it becomes more complicated. Thus, not only is it important to enable distance communication through tools such as telephones, videophones and textual tools such as email, chat, or discussion groups, but also to support some of the social awareness cues that are familiar to co-located settings. For example, some media space tools may support team awareness by providing a low resolution video view port into a remote office [98,101], or set of offices [19, 31]. This allows remote collaborators to 'peek in' at particular team members, and if they deem them to be available try to negotiate an interaction. Other tools take a different approach and provide more dynamic awareness based on virtual proximity, i.e. awareness of people considered 'nearby' based on some criteria. For example, mutual awareness could be supported through video links between team members using the same files or programs [56], or simple textual lists based on current users within a specific virtual community [18]. Such functionality facilitates the opportunistic and spontaneous interactions that are so common and important to the collaborative design process.

Because these tools address only communication aspects of the Workstyle model, artifact related dimensions are not applicable and will not be discussed in the following analysis. Only the dimensions of the model that address collaboration style will be considered – Synchronicity, Location, Coordination and Group Size.

Synchronicity

These communication tools support a range of interactions, from synchronous to asynchronous. Media spaces that provide video view ports into remote offices [19,98,31] support both synchronous and asynchronous interaction by facilitating opportunistic synchronous interactions, as well as providing alternate communication mechanism such as instant messages or email. The same is true for tools supporting proximity awareness [56,18]. Additionally, contextual awareness about collaborators working on common tasks further supports asynchronous work by proving information about the activities of a remote party within the context of the shared task. Common communication tools tend to support either synchronous (e.g. telephone, videophone, real-time chat) or asynchronous (e.g. email, instant message, discussion groups) interactions. However, use of some asynchronous tools can approach synchronous interaction. For example, rapid email exchange or discussion group postings are very similar to real-time chats.

Location

Communication tools such as these are intended to support distance communication, and are generally unnecessary in co-located situations. More common communication mechanism (telephone, email, discussion groups, real-time chat) support greater and more flexible distribution because of the ubiquity of the devices required to use them. Tools that rely on video [98,19,31] require devices that may not be available everywhere, prior installation, as well as sufficient network bandwidth to supply smooth video streams. All of these requirements limit the distribution of collaborators that can be practically supported by these tools.

Coordination

Common synchronous communication tools such as telephone or videophone generally rely on social protocol to mediate conversation, while more asynchronous tools such as email, instant message or even real-time chat tools inherently impose time-based coordination on interactions. Media spaces that support awareness or use video to support synchronous interaction tend to rely on an electronically assisted social protocol. This means that interactions are based on social protocol and facilitated by the awareness cues that are available. For example, many tools [56,98,19] provide mechanisms to protect privacy, and prevent or dissuade interaction. Furthermore, video view ports allow remote users to know whether or not somebody is even available for interaction before attempting to make contact.

Group size

Generally, communication tools support all size groups of collaborators. Some tools, such as telephone or videophone, are simply point-to-point and support only two people. They can scale to slightly larger groups through conferencing mechanisms, though these may introduce coordination issues. Media spaces that rely in video [98,19,31] are limited in the number of separate video view ports that can be shown simultaneously by technical issues such as bandwidth, processor speed and display size, as well as the social impracticality of scaling these view port-based communication tools to larger groups. Tools that support awareness based on virtual proximity [56,18] support larger groups, as only those collaborators considered to be 'nearby', generally a subset of the total group size, are provided awareness of each other. Finally, common asynchronous communication tools (e.g. email, instant message, discussion groups) can support reasonably large groups.

7.2 Summary

The tools of this category provide team communication and awareness in support of collaborative software design. These tools are appropriate to many kinds of working styles. They do not support the production of artifacts. Instead, they enable synchronous and asynchronous communication, and/or provide contextual awareness cues in support of informal or impromptu interactions.

8 Conclusions

Software design and development is a collaborative activity often involving large numbers of people sharing information and resources to complete a variety of inter-related activities. As such, there are a wide variety of tools that are used to support collaboration in software design. They include dedicated communication tools, electronic collaboration environments for synchronous and asynchronous interaction, tools supporting informal interaction, as well as dedicated software design tools. This diversity is motivated by the wide range of activities and workstyles in which software designers engage; different tool types provide functionality to support the different collaborative styles in which designers work. Such diversity is also reflective of the relative failure of most dedicated software design tools to sufficiently address all aspects of design activities, particularly communication and informal interaction. Though some such tools integrate higher-level support for collaboration in software design, most provide only minimal facilities to allow teamwork.

Software modeling tools are most appropriate to a formal collaborative working style characteristic of later stages of development after the initial, creative design work is complete. Modeling languages such as UML can be too cumbersome to support creative design activities where specific details are unknown, semantics are undefined and designs change frequently. Additionally, the asynchronous interactions supported by these tools are better suited to these later stages when more of the relevant information has been precisely recorded within the shared design documents, reducing the need for extensive synchronous communication.

Process centered tools can be used to define and enact collaborative software processes that support a reasonably wide variety of collaborative work styles. Such a process may be designed to facilitate synchronous or asynchronous work, distributed or co-located interactions, as well as all sizes of groups. However, a process is designed to coordinate people's interactions as well as access to resources, and its enactment is intended to impose the specified coordination. Therefore, these tools can restrict people from a free-form style of interaction that may be most suitable to initial, creative design and/or the development of novel software systems.

Asynchronous tools support collaboration in software design in three ways; integration of document repositories and team communication facilities, configuration and/or version management facilities, or through implementation of a 'shared-place' metaphor. These tools support fairly restrictive workstyles. They are flexible in the nature of the artifacts that they support, however, interaction with those artifacts is highly moderated and restricted to small or medium distributed groups. Some 'shared-place' tools do support synchronous interactions, as well as transitions between synchronous and asynchronous work. However, this is at the cost of the size of the group that may be supported.

There are three kinds of tools supporting collaborative software design by enabling synchronous artifact sharing; meta-tools for sharing pre-existing, single-user applications, collaboration-aware CASE tools, and tools supporting co-located design. These tools support a fairly limited set of workstyles, as they are limited to supporting small groups of collaborators interacting synchronously. Concurrency control mechanisms may impose additional restrictions on how designers may interact with each other and the shared artifacts. This is especially true for meta-tools that must use restrictive concurrency controls to maintain the single-user semantics of the underlying application. Collaboration-aware tools minimize these restrictions, but are less flexible in the nature of the artifacts that may be shared. Tools that support co-located interactions further reduce these restrictions and rely on primarily social protocol to mediate concurrent interaction.

Tools that support interaction through informal media support collaborative software design by facilitating unstructured interaction. This informal workstyle allows users to interact naturally and to use the tools transparently with no unnecessary overhead, and is appropriate to the early, creative stages of the software design process. These tools support a small group of designers, mediated through social protocol producing an informal artifact of unbound semantics and free-form syntax. They also support synchronous

and asynchronous interactions through the notion of persistence, as well as distributed and co-located teams.

Communication and awareness tools support a variety of collaborative workstyles in software design. Their variety and ubiquity can make combinations of these tools appropriate to any and all stages of the software design process. Communication and awareness tool enable synchronous and asynchronous communication, as well as provide contextual awareness cues in support of informal or impromptu interactions.

8.1 Open Problems

Each category of tool discussed in this paper addresses a single or limited set of collaborative workstyles. This allows the functionality of individual tools to be appropriately tailored to a particular design context or range of contexts. In combination, these tools can support most common workstyles. This can be seen in the union of the Workstyle analysis plots as shown in Figure 22. This analysis depicts the combined functionality of all tools available to support collaboration in software design. It demonstrates how collaboration might be supported throughout the software design process by an appropriate set of compatible tools. It also reveals some open problems in area of tool support for collaborative software design.

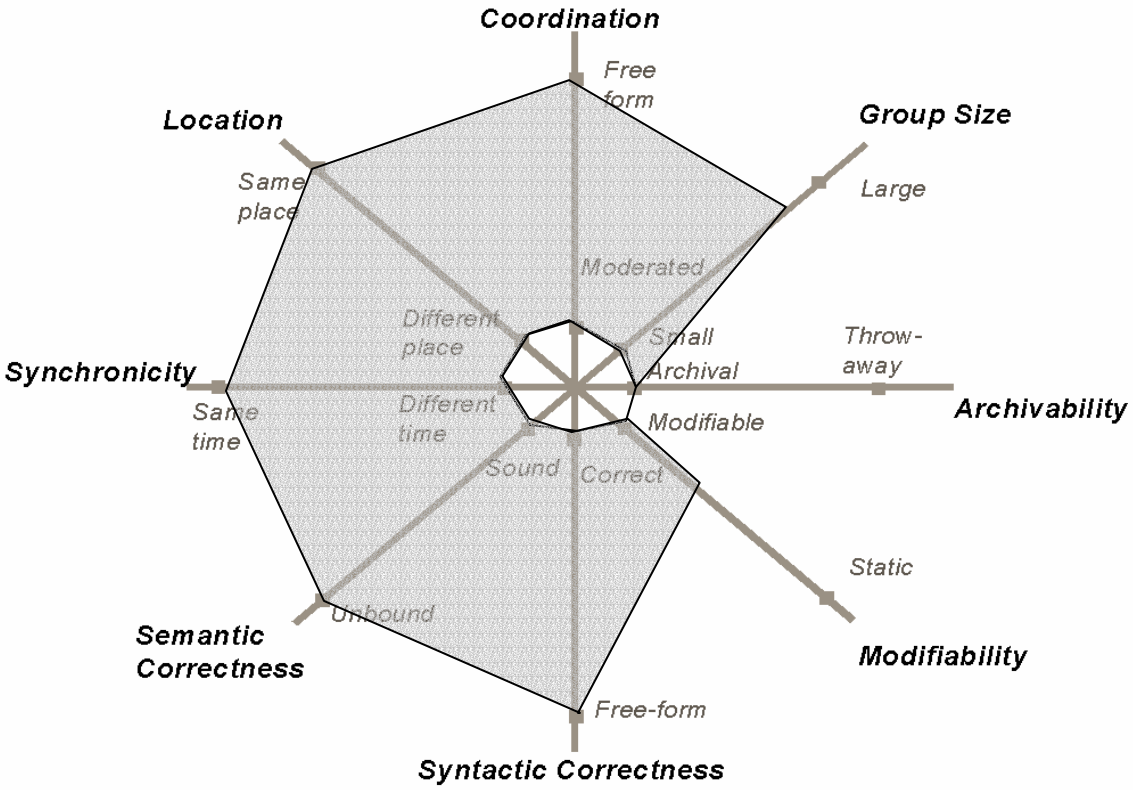

Figure 22: The union of Workstyles supported by collaborative software design tools supports most common styles of collaborative work.

- *Unsupported Workstyles:* Certain workstyles are not supported by any individual class of tools. For example, large groups of synchronous collaborators, whether distributed or co-located, are not well supported by any available tool. This may be a result of hardware restrictions, as well as limited applicability of such a workstyle in practice. Additionally, no environment allows free-form interaction while supporting the creation of syntactically

and semantically refined artifacts. Even informal CASE tools such as IdeogramicUML employ a gesture-based syntax that places restrictions on free-form interaction.

- *Lack of Support for Workstyle Evolution:* Despite the fact that designers frequently change the ways in which they collaborate [112], individual tools only support a single or limited set of workstyles. Furthermore, they provide little or no support for movement between work styles. Traditional software modeling tools typically provide rigid support for a single workstyle as outlined in Figure 3. Shared place tools such as TeamRooms [84] support limited transitions between synchronous and asynchronous collaboration styles, and collaboration aware tools such as JComposer [47] support varying degrees of coordination between collaborators. However, transitions between certain workstyles are not supported at all. For example, transitions along the syntactic and semanic correctness axes are not well supported by any individual tool, or class of tools. Consider a scenario where designers have brainstormed and developed an initial idea on a physical whiteboard. Such an artifact would be considered syntactically free-form and semantically unbound. No tool exists to support migration of this design artifact to a more formal environment where the syntax and semantics of the artifact may be refined. This forces the designers to do extra work in order to continue their design work. The artifact must either be recreated in a more appropriate tool, or abandoned all together.

  UMLRecognizer [66] attempts to address the syntactic aspect in the context of a limited subset of UML. User-initiated recognition allows more freedom during interaction than real-time interpretation [53, 24] because users remain unbound to any syntax until they are ready. However, UMLRecognizer operates in the limited context of a small subset of non-standard UML, and the recognition algorithm requires more accuracy and flexibility to better support transitions from free-form drawing to syntactically correct UML. Furthermore, the decision to use the UMLRecognizer (or any informal CASE tool), as well as the decision to work in a particular language such as UML, must be made beforehand. However, in informal design situations such decisions often cannot be made until after the informal artifact is created [64,111]. Finally, transitions between workstyles often involve changes in interaction devices. For example, moving from an informal to a more formal workstyle may involve switching from an electronic whiteboard to a PC. Available tools do not sufficiently support migration between devices.

- *Lack of Support for Multiple Collaborative Contexts:* In addition to frequently changing their collaborative workstyle, individual designers also switch amongst a number of concurrent collaborative contexts. For example, a given designer may be participating in a number of concurrent projects or tasks. Within each, the style in which the designer is collaborating may evolve over time. Additionally, designers may switch their focus from one project to another. This will move them from one context, involving a particular set of collaborators in some workstyle, to another involving a completely unrelated set of collaborators in a different workstyle. Furthermore, designers may participate concurrently in multiple collaborative contexts. For example, someone may begin to write an email message to a collaborator in one project, while waiting on the telephone for a collaborator in another project. Transitions between concurrent workstyles can be supported to a limited degree by shared place tools in which users may navigate between different active collaborative contexts. For example, different rooms within TeamRooms [84] can represent different collaborative contexts. However, users are still unable to fluidly participate in both rooms simultaneously, as moving from one to another involves a complete loss of context.

- *Lack of Support for Integration of Existing Applications:* Current meta-tools that support sharing of existing applications, such as Netmeeting, impose significant restrictions on collaboration that can be inappropriate for many workstyles. Additionally, they rigidly support only a synchronous, distributed, and highly coordinated workstyle between small numbers of collaborators. Collaboration-aware applications address some of these issues by supporting more flexible collaboration mechanisms, but force designers to give up their preferred tools. For example, JComposer allows designers to collaboratively edit design diagrams in a variety of workstyles. However, in order to reap these benefits designers must choose to use this tool, rather than their preferred diagram editor. In doing so, they must accept any other limitations or drawbacks related to that choice. Furthermore, collaboration-aware applications often address only a very limited task domain; for example, JComposer only supports design in a proprietary language rather than a standardized language such as UML. Mechanisms for integrating existing tools into a variety of collaborative workstyles would allow designers to collaborate on wide variety of tasks without giving up their preferred tools for accomplishing those tasks.

There are other open issues in this field that are not explicitly revealed by the workstyle analyses of the tools currently available to support collaboration in software design. For example:

- *Lack of Suitable Hardware:* Typical computer systems supporting interaction through keyboard and mouse are designed for single users. New hardware must be integrated to support collaboration amongst groups of people. Devices such as web cameras, microphones and speakers are becoming sufficiently inexpensive and ubiquitous that they can be reliably used to support communication and awareness in practical applications. However, supporting more advanced collaboration is more difficult. For example, tools such as InsightLab [65] or Clearboard [55] require such specialized hardware that only prototypes of these tools have ever been created. Similarly, tools that support interaction through informal media may require commercially available, but often prohibitively expensive, hardware such as electronic whiteboards. Even with such expensive hardware, interaction with these devices is typically cumbersome. Issues such as latency and screen resolution make natural interaction with an electronic whiteboard difficult. Similarly, some electronic whiteboards [89] do not support truly concurrent interaction. The solution to these hardware issues will come in time, when the technology evolves and demand increases.

- *UI Transparency vs. Functionality:* There is a trade-off in many of these tools between interface simplicity and functionality. In the early stages of development, interface complexity imposes overhead that can be counter-productive to creative design [107]. Informal tools that support lightweight interactions appropriate to these early stages of design often lack higher-level functionality that may make the tool more usable. Different tools have attempted to address this issue by striking a balance between interface transparency and functionality. For example, shared drawing tools make little or no attempt to provide enhanced editing features common to typical structured drawing tools. Conversely, informal CASE tools and enhanced electronic whiteboard tools may require a complex gesture-based interaction in order to apply higher-level meaning to the artifact. This dissonance between advanced features and ease of use may only be solved by the introduction of a new interaction paradigm for tools that support informal media.

8.2 Future Directions

The preceding discussion of open problems in this field motivates a variety of future research directions. Tools intended to support collaboration in software design could be constructed to support the entire range of working styles as defined by the Workstyle model. However, this will lead to a 'Swiss Army' design tool that supports all workstyles poorly, excelling in none. More appropriately, tools should be designed not only to adequately support a particular workstyle, but also to support common transitions into and out of that workstyle. This support should include migration between devices as well as concurrent collaboration contexts. This will allow designers to chose particular tools that support their preferences and current style of work, and facilitate movement between preferred tools as that style of work changes. This would reduce the overhead currently involved in such changes in workstyle, as well as promote the use of preferred workstyles, allowing software designers to work and collaborate more effectively.

9 References

[1] 3M – Ideaboard, http://www.3m.com/meetings/product_catalog/wd_ideaboard.jhtml

[2] AgileAlliance, http://www.agilealliance.org

[3] Alavi, M. (1993). "Making CASE an Organizational Reality: Strategies and New Capabilities Needed." Information Systems Management: 15-20.

[4] Allen, T.J., (1977). Managing the Flow of Technology. Cambridge, MA: MIT Press.

[5] Ambiente – I-Land, http://www.ipsi.fhg.de/ambiente//home.html

[6] Aonix – Software through Pictures, http://www.aonix.com

[7] ArgoUML – http://argouml.tigris.org

[8] Bandinelli, S., Di Nitto, E., Fuggeta, A. (1996). "Supporting Cooperation in the SPADE-1 Environment." IEEE Transactions on Software Engineering **22**(12): 841-865.

[9] Barghouti, N. S. (1992). "Supporting Cooperation in the Marvel Process-Centered SDE". Symposium on Software Development Environments, ACM Press.

[10] Begole, J., Struble, C.A., Shaffer, C.A. (1997). "Leveraging Java Applets: Toward Collaboration Transparency in Java" IEEE Internet Computing, (1)2, pp. 57-64.

[11] Ben-Shaul, I. Z. and G. E. Kaiser (1994). "A Paradigm for Decentralized Proccess Modeling and its Realization in the Oz Environment". 16th International Conference on Software Engineering, IEEE CS Press.

[12] Bly, S., A. (1988). "A Use of Drawing Surfaces in Different Collaborative Settings". Conference on Computer-Supported Cooperative Work, Portland, OR.

[13] Bly, S. A., Harrison, S. R., Irwin, S. (1993). "Media Spaces : Bringing People Together in a Video, Audio and Computing Environment." Communications of the ACM **36**(1): 28-47.

[14] Bly, S.,A. and S. Minneman (1990). "Commune : A Shared Drawing Surface." SIGOIS Bulletin: 184-192.

[15] Bogia, D. P. and S. M. Kaplan (1995). "Flexibility and Control for Dynamic Workflows on the wOrlds Environment". Conference on Organizational Computing Systems, Milpitas, CA, ACM Press.

[16] Booch, G. (1994). Object-Oriented Analysis And Design - With Applications Second Edition, Redwood City, CA.: The Benjamin/Cummings Publishing Company,Inc.

[17] Chmura, L. and A. Norcio (1986). "Design Activity in Developing Modules for Complex Software". 1St Workshop on Empirical Studies of Programmers. Norwood, NJ., Ablex Publishing Corp.,

[18] Cockburn, A. and S. Greenberg (1993). “Making contact: Getting the group communicating with groupware”. Conference on Organizational Computing Systems, Milpitas, CA, ACM.

[19] Coutaz, J., Berard, F., Carraux, E., Crowley, J. (1998). "Early Experience with the Mediaspace CoMedi." IFIP Working Conference on Engineering for Human-Computer Interaction. Heraklion, Crete, Greece September 14 - 18, 1998

[20] Craighill, E., Lang, R., Fong, M., and Skinner, K. "CECED: A System for Informal Multimedia Collaboration." Proceedings of ACM Multimedia. Anaheim, CA. August.

[21] Crowley, J., Coutaz, J., Berard, F. (2000). "Things that See." Communications of the ACM **43**(3): 54-64.

[22] CVS – Concurrent Versions System, http://www.cvshome.org

[23] Damm, C.H., Hansen, K.M., Thomsen, M., Tyrsted, M. (2000). “Creative Object-Oriented Modelling: Support for Creativity, Flexibility, and Collaboration in CASE Tools”. Proceedings of ECOOP'2000. Sophia Antipolis and Cannes, France, June 12-16. Pages 27-43.

[24] Damm, C. H., Hansen, K. M., Thomsen, M. (2000). “Tool Support for Object-Oriented Cooperative Design: Gesture-Based Modelling on an Electronic Whiteboard”. Proceedings of Conference on Human Factors and Computing Systems. The Hague, Netherlands.

[25] DeMarco, T. and T. Lister (1987). Peopleware. New York, Dorset House.

[26] Demers, A., K. Petersen, M. Spreitzer, D. Terry, M. Theimer and B. Welch (1994). "The Bayou Architecture : Support for Data Sharing among Mobile Users." Proceedings IEEE Workshop on Mobile Computing Systems & Applications.

[27] Dewan, P., Choudary, R. (1995) “Coupling the User Interfaces of a Multi-User Program”, ACM Transactions on Computer-Human Interaction.

[28] Dewan, P. Choudary, R. (1991). “Flexible user interface coupling in collaborative systems”. CHI ' 91, New Orleans, LA, ACM.

[29] Dewan, P. (1993). “Tools for Implementing Multiuser User Interfaces”. User Interface Software. L. Bass and P. Dewan, John Wiley & Sons Ltd.

[30] Dewan, P. and J. Riedl (1993). "Toward Computer Supported Concurrent Software Engineering." IEEE Computer. V26, pp 17-28.

[31] Dourish, P. and S. Bly, A. (1992). “Portholes: Supporting Awareness in a Distributed Work Group”. CHI '92, Monterey, CA.

[32] Embarcadero – Describe UML, http://www.embarcadero.com

[33] ERoom Technology, Inc.– Erooms, http://www.eroom.com

[34] Ergonis Software – Voodoo Personal, http://www.macility.com

[35] Excel Software – MacA&D, http://www.excelsoftware.com

[36] Fernstrom, C. (1993) “PROCESS WEAVER: Adding process support to UNIX”. 2nd International Conference on the Software Process: Continuous Software Process Improvement, Berlin, Germany, February 1993. IEEE Computer Society Press.

[37] Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., & Nuseibeh, B., “Inconsistency Handling In Multi-Perspective Specifications”, IEEE Transactions on Software Engineering, 20(8), pp. 569-578. 1994.

[38] Fitzpatrick, G. (1998). "The Locales Framework: Understanding and Designing for Cooperative Work." Ph.D. Thesis. The University of Queensland

[39] Fitzpatrick, G., W. J. Tolone, Kaplan, S.M. (1995). “Work, Locals, and Distributed Social Worlds”. Fourth Euopean Conference on CSCW, Stockholm, Sweden, Kluwer Academic Publishers.

[40] Galegher, J. and R. Kraut, (1990) “Computer-Mediated Communication for Intellectual Teamwork: A Field Experiment in Group Writing”. Proceedings of CSCW90. Los Angeles, CA

[41] Graham, T. C. N., Stewart, H. D. Ryman, A.G, Kopaee, A.R., and Rasouli, R. (1999). “A World-Wide-Web Architecture for Collaborative Software Design”. Software Technology and Engineering Practice, IEEE Press.

[42] Greenberg, S. and R. Bohnet (1991). “GroupSketch: A Multi-user Sketchpad for Geographically Distributed Small Groups”. Proceedings of Graphics Interface, pp 207-215.

[43] Grundy, J. C. and J. G. Hosking (1998). “Inconsistency Management for Multiple-View Software Development Environments”. IEEE Transactions on Software Engineering, 24(11):960-981

[44] Grundy, J. C. and J. G. Hosking (1998). "Serendipity: Integrated environment support for process modelling, enactment and work coordination." Automated Software Engineering **5**(1): 27-60.

[45] Grundy, J.C., Hosking, J.G., and Mugridge, W.B. (1996) “Supporting Flexible Consistency Management via Discrete Change Description Propagation”. Software, Practice & Experience, v.26 n.9, p.1053-1083, Sept. 1996

[46] Grundy, J. C., Hosking, J. G., Mugridge, W.B., Amor, R.W. (1996). “Support for Constructing Environments with Multiple Views”. SIGSOFT '96, San Francisco, CA.

[47] Grundy, J. C., Mugridge, W.B, Hosking, J.G., Apperley, M. (1998). “Tool Integration, Collaboration and User Interaction Issues in Component-based Software Architectures”. TOOLS '98, Melbourne, Australia, IEEE.

[48] Hammond, T. and R. C. Davis (2002). “Tahuiti: A Geometrical Sketch Recognition System for UML Class Diagrams”. Sketch Symposium, Stanford University, Palo Alto, CA.

[49] Harrison, S., Minneman, S., Stultz, R. (1988). “The Media Space - experience with video support of design activity”. Proceedings of International Workshop on Engineering Design and Manufacturing Management, Melbourne, Australia. pp. 114-126

[50] IBM Lotus Software - Notes, http://www.lotus.com

[51] IBM Lotus Software – QuickPlace, http://www.lotus.com

[52] Iconix Software – PowerTools, http://www.iconixsw.com

[53] Ideogramic – IdeogramicUML, http://www.ideogramic.com

[54] Iida, H., Takeshi, O., Inoue, K., and Torii, K. (1991) "Generating a menu-oriented navigation system from formal description of software development activity sequence." 1st International Conference on the Software Process: Manufacturing Complex Systems. Renondo Beach, CA. October

[55] Ishii, H. and M. Kobayashi (1992). "ClearBoard: A seamless medium for shared drawing and conversation with eye contact". Conference on Human Factors in Computing Systems, Monterey, CA, ACM.

[56] Issacs, E. A., Tang, J.C., Morris, T. (1996). "Piazza: A Desktop Environment Supporting Impromptu and Planned Interactions". CSCW '96, Cambridge, MA, ACM Press.

[57] Jarzabek, S. and R. Huang (1998). "The Case for User-Centered CASE Tools." Communications of the ACM **41**(8): 93-99.

[58] Johansen, R., Sibbet, D., Benson, S., Martin, A., Mittman, R. and Saffo, P. (1991) Leading Business Teams. Addison-Wesley.

[59] Jones, T. C. (1986). Programming Productivity. New York, McGraw-Hill.

[60] Kaplan, S., Mansfield, T., Fitzpatrick, G., Phelps, T., Fitzpatrick, M. and Taylor, R. (1997) "Evolving Orbit: a progress report on building locales". Proceedings GROUP'97, Phoenix, AZ, ACM Press, pp. 241-250.

[61] Kaplan, S. M., Fitzpatrick, G., Mansfield, T., Tolone, W.J. (1996). "Shooting into Orbit". Oz-CSCW '96, Brisbane, Australia, University of Queensland.

[62] Kraut, R. E., Fish, R. S. , Root, R.W., Chalfonte, B. L. (1990). "Informal Communication in Organizations : Form, Function and Technology". People's Reaction to Technology. S. Oskamp and S. Spacapan. Newbury Park, Sage Publications**:** 145-149.

[63] Kraut, R. E. and L. Streeter (1995). "Coordination in Software Development." Communications of the ACM **38**(3): 69-81.

[64] Landay, J. A. and B. A. Myers (1995). "Interactive Sketching for Early Stages of Design". CHI '95, Denver, CO, ACM Press.

[65] Lange, B. M., Jones, M.A., Meyers, J.L. (1998). "Insight Lab : An Immersive Team Environment Linking Paper, Displays and Data". CHI '98, Los Angeles, CA, ACM.

[66] Lank, E., Thorley, J.S., Chen, S.J. (2000). "An Interactive System for Recognizing Hand Drawn UML Diagrams". CASCON2000, Toronto, ON.

[67] Malone, T. W. and K. Crowston (1990). "What is coordination theory and how can it help design

cooperative work systems?". Proceedings of Conference on Computer-Supported Cooperative Work. ACM Press. pp. 357-370

[68] Marlin, C., B. Peuschel, McCarthy, M., Harvey, J. (1993). "MultiView-Merlin: An Experiment in Tool Integration". Proceedings of 6th Conference on Software Engineering Environments, IEEE CS Press.

[69] MetaCase Consulting – MetaEdit+, http://www.metacase.com

[70] Microsoft Corp. – Netmeeting, http://www.microsoft.com

[71] Microsoft Corp. – PowerPoint, http://www.microsoft.com

[72] Microsoft Corp. – Visual SourceSafe, http://www.microsoft.com

[73] Microsoft Corp. – Word, http://www.microsoft.com

[74] Miller, D. S., Smith, J.G., Muller, M.J (1992). "TelePICTIVE: Computer-Supported Collaorative GUI Design for Designers with Diverse Expertise". UIST '92, Monterey, CA, ACM.

[75] Mynatt, E. D., Igarashi, T., Edwards, W.K. LaMarca, A. (1999). "Flatland : New Dimensions in Office Whiteboards". CHI '99, Pittsburg, PA, ACM.

[76] No Magic – MagicDraw, http://www.magicdraw.com

[77] Object Management Group (OMG) 1998. Unified Modeling Language Specification. Framingham, MA: Object Management Group.

[78] Pankoke-Babatz, U. and A. Syri (1997). "Collaborative Workspaces for Time Deferred Electronic Cooperation". Proceedings of GROUP '97, Phoenix, AZ, ACM Press.

[79] Pederson, E. R., McCall, K., Moran, T.P., Halasz, F. G. (1993). Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings. INTERCHI '93. Amsterdam, Netherlands. April.

[80] Popkin Software – System Architect, http://www.popkin.com

[81] Rational Corp. – Rose, http://www.rational.com

[82] Rational Corp – ClearCase, http://www.rational.com

[83] Roseman, M. and S. Greenberg "Building Groupware with Groupkit". Tcl/Tk Tools, O'Reilly Press**:** 535-564.

[84] Roseman, M. and S. Greenberg (1996). "TeamRooms: Network places for collaboration". Conference on Computer Supported Cooperative Work, Boston, MA, ACM.

[85] Rudebusch, T. (1993) "CSCW: generische Unterstutzung von Teamarbeit in verteilten DV-Systemen", Deutscher Universistats-Verlag, GmbH, Wiesbaden. Univerisity of Karlsruhe, Germany. ISBN 3-8244-2043-0

[86] Rumbaugh, J., Jacobson, I., Booch, G. (1999). The Unified Modeling Language. Addison-Wesley-

Longman.

[87] Rumbaugh, J., Blaha, M.R., Lorensen, W., Eddy, F., Premerlani, W. (1991). Object-Oriented Modeling and Design. Prentice Hall.

[88] Seaman, C.B. and Basili, V.R. "Communication and Organization in Software Development: An Empirical Study". IBM Systems Journal 36(4), 1997.

[89] SMART Technologies, Inc. – SMARTBoard, http://www.smarttech.com

[90] SpeeDev, Inc. – SpeeDEV, http://www.speedev.com

[91] Stefik, M., G. Foster, Bobrow, D.G., Kahn, K., Lanning, S., Suchman, L. (1987). "Beyond The Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings." Communications of the ACM **30**(1): 32-47.

[92] Streitz, N. A., J. Geißler, Haake, J. M., Hol, J. (1994). "DOLPHIN: integrated meeting support across local and remote desktop environments and LiveBoards". Conference on Computer Supprted Cooperative Work, Chapel Hill. NC.

[93] Streitz, N. A. (1999). "i-LAND: An interactive Landscape for Creativitiy and Innovation". ACM Conference on Human Factors in Computing Systems, Pittsburg, PA, ACM Press.

[94] Swenson, K.D. (1993) "The Regatta Project", Proceedings of the First International Conference in Technologies and Theories for Human Cooperation, Collaboration, and Coordination, Applica `93, March 1993

[95] Tang, J., C. (1989). "Listing, Drawing and Gesturing in Design: A Study of the Use of Shared Workspaces by Design Teams". Ph.D. Thesis, Department of Mechanical Engineering, Stanford University**:** 172.

[96] Tang, J., C. (1991). "Findings from Observational Studies of Collaborative Work." International Journal of Man-Machine Studies. Vol. 34, No. 2, pp. 143-160

[97] Tang, J. C. (1990). VideoDraw: A Video Interface for Collaborative Drawing. CHI '90, Seattle, WA.

[98] Tang, J. C., Issacs, E.A. , Rua, M,. (1994). "Supporting Distributed Groups with a Montage of Lightweight Interactions". Proceedings of CSCW '94, Chapel Hill, NC, ACM Press.

[99] Tang, J. C. and L. J. Leifer (1988). "A Framework for Understanding the Workspace Activities of Design Teams." Proceedings of the Conference of Office Information Systems, Cambridge, MA. ACM Press, pp. 244-260.

[100] Tang, J. C. and S. Minneman (1991). "VideoWhiteboard: Video Shadows to Support Remote Collaboration". Conference on Human Factors and Computing Systems, New Orleans, LA.

[101] Tang, J. C. and M. Rua (1994). "Montage: Providing Teleproximity for Distributed Groups". CHI '94, Boston, MA, ACM Press.

[102] Telelogic – Tau, http://www.telelogic.com

[103] Tichy, W.F., (1985). “RCS--A System for Version Control, Software”. Practice & Experience. Vol. 15, No. 7, pp 637-654.

[104] TogetherSoft Corp. – TogetherSoft, http://www.togethersoft.com

[105] Tung, T. L. (1997). "Mediaboard: A Shared Whiteboard Application for the MBone.". Master’s Thesis, Computer Science Division, University of California at Berkeley

[106] Virtual Ink Corp – Mimio, http://www.mimio.com/index.shtml

[107] Wagner, A. (1990) “Prototyping: A day in the life of an interface designer”. The Art of Human-Computer Interface Design. Addison-Wesley, Reading, MA, pp. 79-84

[108] Wang, W., Dorohonceanu, B., Marsic, I. (1999). “Design of the DISCIPLE Sycnhronous Collaboration Framework”. Internet, Multimedia Systems and Applications, Nassau, Bahamas, IASTED Press.

[109] Whittiker, S., D. Frohlich, Daly-Jones, O. (1994). “Informal Workplace Communication: What is it Like and How Might We Support it?. CHI '94, Boston, MA, ACM Press.

[110] Wolf, C. G. and J. R. Rhyne (1993). “Gesturing with Shared Drawing Tools”. CHI '93, Amsterdam, Holland, ACM Press.

[111] Wong, Y.Y. (1992) “Rough and ready prototypes: Lessons from graphic design”. Short Talks Proceedings of CHI '92: Human Factors in Computing Systems, pp. 83-84, Monterey, CA,

[112] Wu, J., Graham, T.C.N, Everitt, K., Blostein, D. and Lank, E. (2002) “Modeling Style of Work as an Aid to the Design and Evaluation of Interactive Systems”. Proceedings of CADUI’02. Valenciennes, France.

[113] XTV (1993) http://www.cs.odu.edu/~waha_cit/XTV.doc/xtv.html