VaDeR: Visualizing a Development Record
A Study of Claims-Centric Scenario-Based Design

Brandon Berry

Submitted to the Faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Dr. D. Scott McCrickard, Chair
Dr. Chris North
Dr. Manuel A. Pérez-Quiñones

June 21, 2004
Blacksburg, Virginia

Keywords: scenario-based design, claims, case studies, problem-based learning, information visualization, human-computer interaction

Abstract

VaDeR: Visualizing a Development Record

A Study of Claims-Centric Scenario-Based Design

Brandon Berry

The scenario-based design process, particularly with an emphasis on claims, captures and shares design knowledge that has promise in teaching design. However, the best methods are not fully understood yet. This thesis examines the areas of scenario-based design, claims relationships, and learning through cases with the goal of learning what can be improved to better support these processes.

In investigating those areas, an information visualization tool is created for learning design through case studies. The tool combined claims from multiple visualization techniques to create an approach for supporting problem-based learning goals. In creating the tool, scenario-based design was used, with a particular emphasis on claims for driving the development and evaluation. All design rationale was captured in claims to create a tool that addressed the current problem issues identified. A recently developed claims relationship framework, used for the first time in development efforts, aided in structuring the design rationale. Using these relationships, the design was tracked from requirements analysis to problem claims to design claims and finally to claims that formed the basis of an evaluation.

This thesis presents a case study of using claims relationships to drive the development and evaluation of a tool for supporting problem-based learning through case studies of scenario-based design. In creating a novel tool for learning about cases, a unique example of claims-centric scenario based design emerges, presenting as open questions new directions for merging scenarios, claims, and cases in knowledge storage, access, and learning systems.

Acknowledgments

This work would not have been possible without the frequent guidance of Dr. McCrickard. The help of Shahtab Wahid and Jamie Smith was invaluable in the early stages of the VaDeR project. And everyone who participated in the evaluations provided valuable insight into the interfaces. So thank you to everyone who had any connection to the work presented here. Also thanks to members of my committee, Dr. North and Dr. Pérez-Quiñones.

The author also wishes to note that this thesis would never have been completed without the author. Thank you to myself.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Scenario-based design is a development process suitable for capturing and sharing design knowledge. But doing so is not easy as the methods are not fully defined yet. A recently developed claims relationship framework should aid in structuring the design rationale, but it has not been used yet for this purpose. This thesis presents a case study of using claims relationships to drive the development and evaluation of an information visualization tool for supporting problem-based learning through case studies of scenario-based design.

As such, this thesis offers contributions in three area outlined in this introduction: scenario-based design, problem-based learning with development records in education, and information visualization.

1.1 Scenario-Based Design

Designing an interface for a user is not trivial. Every decision made can have a major impact on the usability of that interface. The trick is to remain focused on what the user needs to perform their tasks. One such way to accomplish that goal is to base the design around realistic usage scenarios. These help envision exactly how a user will use the interface and the issues they may encounter. As an extension of that, key features from those scenarios are identified to draw out what the effective parts of the interface are. Those parts are then examined for possible upsides and downsides. Using those effects, the design is driven to create the desired upsides while minimizing the harmful downsides.

Scenario-based design [32] is centered around that idea of usage scenarios and the related upsides and downsides with respect to a certain feature, called a claim. The hope is that basing a design around claims allows the designer to focus on precisely what impact their design will have and thus enable them to create a system that meets their goals. This project will apply scenario-based design to creation of a tool for exploring development records. By doing so, the ability of this design process to enable the use of claims for driving the design and evaluation will be analyzed. Not only that, but a recently developed framework [46] for tracking the relationships between claims will be put to the test as to how much it aids in identifying the important claims and driving the design forward.

The problem domain in which this process will be used is the use of development records in aiding the education of an human-computer interaction student. Starting with this domain, a current tool will be examined for its successes and failures in supporting the user of development records within a specific teaching environment. Then a new system will be designed to improve upon that tool, with its design and evaluation driven by the claims. An introduction to the problem domain is provided in the next section.

1.2 Development Records and Education

When an interface is designed and implemented, there is some process that the developers follow to create that interface. There must have been some design decisions made. Prototypes were likely created. And there is usually a testing phase on the prototype or the final interface. This leaves a tangible trail that can allow others to follow the development of the interface. We can call that trail a development record. A development record with context and commentary provided is a case study.

More specifically, cases are “evocative narrative descriptions of a specific real-world activity, event or problem.” [9]. Used in education, they provide a model for the student to learn from, especially by analyzing the decisions and exploring what might result from different decisions. Their presentation, though, must be well designed to fully support that educational process.

Academically published papers about an interface development often hint at a development record but the paper can only reveal so much. The requirement of a descriptive, narrative structure pushes many details out of view. Developers are able to present the interface and discuss testing, design rationale, and implementation. But with having to consider space limitations, the intended audience, and a paper structure, many useful details are lost.

But what if the intent is to teach students about a design process? Without the full detail, learning about the complexities of the process is difficult. So the published paper is not the best way to present an entire development record. To counter that problem, an alternate approach was designed to facilitate access to development record information, called the Usability Case Study Tool (UCS) [31].

For teaching a course on human-computer interaction, development records provide examples of realistic interface design efforts. Teachers can use them to reinforce the concepts they are teaching by grounding them in actual use. Students can learn from them as references and explore how an interface can be designed. UCS has been successfully used for those purposes.

One effective teaching method, that can incorporate UCS, is problem-based learning. Unfortunately, UCS is not ideal for supporting the stages of that technique. It fails as a viable reference when the student is trying to direct their own learning. The format hinders the student’s ability to fully and efficiently explore all of the data. Frustrating the student can not bode well for their education. A new interface for exploring the information is needed, and this is called VaDeR: Visualizing a Development Record.

1.3 Information Visualization

There must be another way to present the information in such a way that it better fits the tasks UCS is meant to support within problem-based learning. By following a scenario-based design process and introducing strategies from the field of information visualization, a more efficient and engaging method of using development records is possible. VaDeR is the implementation of these strategies.

Information visualization deals with strategies and techniques for mapping data into some visual form that can then be interacted with. VaDeR's design will reuse claims from those techniques within the framework of scenario-based design. As a result, the design will be an example of how to reuse knowledge from other domains to help create an interface that addresses the problem domains.

1.4 Thesis Roadmap

The remainder of this thesis is structured as follows: Chapter 2 describes the background and motivation behind this work. Chapter 3 outlines the initial efforts, and requirements analysis, towards building a UCS-style tool, discussing successes and failures of using a claims-centric scenario-based design approach. Chapter 4 follows up with a more claims-driven second iteration of the VaDeR tool development, followed by an explanatory analysis of the tool in Chapter 5. Finally, Chapter 6 provides conclusions and future work.

Chapter 2

Background and Motivation

This chapter introduces the scenario-based design process that was used to create and test VaDeR. By using scenarios to envision the usage of VaDeR and using claims about those scenarios to drive the design and lead to an evaluation, the final product emerged.

Also introduced in this chapter is the educational technique VaDeR was designed to support. Constructivism, a theory about learning, can be applied as problem-based learning when teaching a class. One such class it is appropriate for is a course on human-computer interaction that teaches scenario-based design. And problem-based learning requires student access to reference sources.

UCS contains case studies of scenario-based design, but lacking is support for key tasks necessary for allowing full insight into the data. By applying information visualization, VaDeR addressed those needs by presenting the data in way that better support key tasks. Thus, applicable techniques from information visualization are also discussed and their relationships to this effort are described.

2.1 Scenario-Based Design

A development record, the collection of artifacts produced by the design team, can be created from any design process the developers follow. But it was infeasible for this project to create a visualization that could take into account every process possible and still be effective. Since the intent was to take the basic ideas behind UCS and improve their implementation, the visualization focuses on the design process that UCS handles. This is scenario-based design (SBD), as defined by Rosson and Carroll in their textbook "Usability Engineering" [32]. While the fundamentals behind the visualization remain flexible and appropriate for any development record, this focus on case studies within scenario-based design allowed valuable comparisons between UCS and VaDeR.

This process was also used to drive the design of VaDeR. A focus is placed upon the scenarios and the related claims. Those claims suggested how the system should be designed to improve upon UCS with respect to the identified tasks. Then by integrating the idea of claim relationships, the most important aspects of VaDeR were identified and evaluated.

2.1.1 Analysis

Root Concept

Without a shared vision, the development team will not be focused on achieving the same goals. It is important to decide first what the system is. There is at least some rough idea of why the system should exist, or else the development team would not be at this point. If the system is implemented, then it will have an impact on a group of people. Such people are called stakeholders. Note that a stakeholder need not be a person, or organization, that will physically use the system. It is possible that a stakeholder simply works with the actual user.

Thus the next step is to perform a stakeholder analysis. For each stakeholder, the development team can project how the system will affect them, including positives and negatives. Now the users go from being abstract concepts to real people whose lives could change if this system is built and used.

To finalize a shared vision, a set of assumptions must be agreed upon. Budget constraints, design techniques, performance goals, and other such realities factor into root concept. Combining these assumptions with the initial vision and stakeholder analysis creates a common ground [17, 19]. So within SBD, the root concept operates as the common ground.

Field Studies

Before any design work can begin, requirements must be collected and analyzed. Here, it is necessary to study the current situation. The proposed system must have some problem domain in mind. Unless the system is entirely novel, an unlikely and unrealistic occurrence, there must be current practices that allow users to work within that problem domain. So the development team must explore what those practices are.

One possible approach to collecting information about current practices is to conduct a field study. This allows the team to observe the practices and interact with the participants. Brainstorming is also useful, especially if an actual field study is impractical. Whatever is done, SBD advocates the analyzing work practices to learn about the activities, artifacts, and social context of the workplace. After all of that information is collected, it can then be summarized and analyzed to find important themes and issues. This leads to the creation of problem scenarios, as discussed in Section 2.1.2.

2.1.2 Design

After the initial background work is complete, the next step is to actually design the system based on the requirements that were created. Scenarios are stories that explore possible designs. Key features from those are analyzed as to their possible effects and this creates claims. Then prototypes are created to envision the design for testing purposes.

Scenarios

The general definition of a scenario is that it is "a narrative or story that describes the activities of one or more persons, including information about goals, expectations, actions, and reactions." [32, 50] Using scenarios allows designers to focus on what the user and system will do without having to specify all of the details about how the actions will occur. Each one allows for discussion and reflection to take place amongst the design team.

Corresponding with four major parts of SBD, there are four types of scenarios. After the requirements analysis is completed, problem scenarios are written. These scenarios simply describe the current practices, as collected through field studies or brainstorming. This allows the design team to highlight key features, such as important objects and actions that should be kept in mind when creating the new system.

Activity scenarios are next. With these, stories are told about supported activities that the user will expect from the system. No consideration to precisely how the interface will look or interact is given. Instead, they just define what the user can do with the system.

Designing the look of the interface coincides with information scenarios. This defines what the user will see on the interface and includes such types as text and icons. It is not described yet as to how the user interacts with what they see, but the design created here suggests possible interaction methods and leads directly into interaction design.

For that, interaction scenarios describe what the user performs and the feedback involved in some action. Here the user performs the supported activities based on the information they perceive. So there is a logical progression to the scenario types, although it is not a strict requirement. Indeed, one can certainly think of examples where changing the information scenario does not imply a change to the interaction scenario.

Finally, documentation design can be performed as an option. This can be performed separately by using all of the previous design phases to create a documentation system. In effect, it is just designing another interface. But it is important enough to note as a step in the process of creating a complete system.

Claims

Claims are positive and negative effects tied to a specific feature. [8] They can be extracted from scenarios or be reused from past work if existing features are being used again or modified to fit new interfaces. For that reuse purpose, a claims library [27] is in development to facilitate the capture of design rationale for targeted types of systems. Not yet well understood by this and related efforts, however, are all of the methods for identifying and evolving claims to be applied during design and evaluation. This work explored these issues, providing a case study of emerging methods and techniques.

An example of a claim is an activity supported or an artifact displayed on a screen. The design team can then think about the possible effects that the specific feature will have, both positive and negative. It is important that there be both positive and negative effects, as it is unlikely a feature will have no upsides or have no downsides. Such a balanced view will force the designers to consider the impact of their design decisions.

This list of claims is then analyzed to decide what features should be included in the design and how they can be altered to help increase the number of positive claims while eliminating or mitigating the negative claims. Note that since the initial claims were extracted from the scenarios, there is no guarantee they are valid. But that not need always be true if claims are reused from other design projects and those claims were tested.

If the scenario uses the same feature of another system, then upsides and downsides of that feature can be brought into the design rationale of the new system. Since features are not always used identically, and multiple features may interact to produce new upsides and downsides, the claims may evolve in the design of the new system. This is where the use of claim relationships [46] (as discussed in Section 2.2) aided in identifying these evolutions.

Prototypes

Throughout the design phase, it is helpful to create prototypes of the proposed system. A prototype can be created relatively quickly and cheaply, allowing for iterative design to flow easily. By demonstrating the design ideas through prototypes, the design team can discuss the ideas with potential users or test the design claims. Since each prototype requires much less time to create than it would to actually implement the system, there is no forced momentum where the team feels like it can not afford to change their ideas. This also allows team members without implementation skills to be heavily involved in the design.

Each type of design scenario can lead to prototypes. Activity scenarios promote prototypes where the look and feel of the interface is largely unimportant. Instead, they just demonstrate the supported activities and how the flow of work will occur. Information scenarios then require prototypes where the visual representation of information is emphasized. Now the design team can think about what data should be shown and how the layouts of each component fit together to form the interface.

Interaction scenarios encourage prototypes where the design team can work out how a user acts upon the information they perceive. How the interface will eventually look is not the focus here. The important factor here is that the user is able to carry out their goals using the provided interface.

2.1.3 Claims versus scenarios as a driving design entity

A key issue with scenario-based design is whether the focus should be on scenarios or claims. The textbook [32] by Carroll and Rosson emphasize scenarios as the key force behind driving the design. Claims then just spring forth from the scenarios, with little regard for future reuse possibilities. Reusing knowledge from scenarios is partly addressed by Carroll et al. [7], and claims are briefly mentioned, but again the focus is on scenarios as the key method for capturing design rationale.

Claims rise in importance in another Carroll et al. paper [10], where the use of claims is meant to aid theory development. Here, claims are not thrown away after a particular interface is done. Instead the lessons learned from them are taken for the purpose of applying them to broader contexts. This is of particular focus in Sutcliffe's and Carroll's efforts [43] to classify and relate claims for the purpose of reuse. In this approach, claims are the focus and driving force of the design.

This project used the approach of focusing on claims as the key method for capturing design knowledge and rationale. Scenarios were written to envision usage, but it is the claims that identify the upsides and downsides of design features. Those claims were then used to ensure that the design addressed the issues with the problem, and that the desired upsides were being introduced into the design of VaDeR. As a result, this project was able to reflect on this approach and gain more insight into if the technique of focusing on claims worked as intended.

2.1.4 Evaluation

Once a prototype or implementation is created, it is tested for its usability. Since claims were already attached to the design, they are used to guide the evaluation. By setting up the evaluation to explore each claim, the validity of those claims can be established. A positive claim might be false so it is imperative it be verified or else the design will continue to emphasize features that do not match their intended effect.

Since the prototypes are not the final system, testing them is a formative evaluation. This provides feedback that can be used to alter the design to enhance its usability [35]. Many problems can be identified, indicating to the design team precisely what should be addressed. Since the design is not final, now is the time to test the claims that were developed. Then the design can be altered to mitigate the negative claims and confirm or create more positive claims. Through the iterative design process, any number of formative evaluations can be performed.

At the end of the interface design, or at a critical point, a summative evaluation is performed. Here, the results tell the team if their system met their goals or how it compares to other systems. Thus the system undergoing evaluation should not be a prototype or else the test results may not apply to the finished system.

2.2 Claim Relationships

Using claims during scenario-based design helps in directing the design of the interface and ensuring the designers are aware of the effects each feature they integrate will have. But there is no established method for tracking how the knowledge within claims evolves throughout the process. Such a method will aid the designers in structuring their design rationale and identifying what is important to consider for future design iterations.

A proposed framework for that purpose is the idea of claim relationships [46] from Wahid et al. It not only aids an ongoing iterative design process, it helps in identifying and creating claims appropriate for reuse in a later project. The authors assert that "...explicit relationships aid in the transfer of claim-embodied knowledge in both the short-term and the long-term."

There are currently six claim relationship types introduced by the authors. The first is predication/postulation. This tracks the claims that go from the problem domain into the design claims. A problem claim will postulate a design claim. And going in the opposite direction, that design claim predicates the problem claim. Using this relationship allows the designers to track how the design claims match and address the problem claims.

To relate claims of the same basic interface feature, but of different scope, there is the generalization/specification relationship. For example, there could be a general claim about using an icon to represent information but a specific claim about using a certain shaped icon that is colored. So identifying the general claims result in knowledge useful over a wider range of design while the specific claims identify the precise effects a feature will have in narrower context.

Referring to Norman's stages of action [24], there are evaluating/executing claims to find the claims that address the gulf of evaluation or gulf of execution. Starting with a claim to belongs to either gulf, the destination claim is one that addresses the execution or evaluation gulf. Unlike the predication/postulation and generalization/specification relationship, this relationship does not have to show up in pairs. Each one may be used individually.

The fusing claim relationship identifies where aspects of multiple claims combine into one claim. Multiple interface artifacts may interact together to create a new claim that highlights the upsides of downsides of those artifacts working together. In the opposite direction, a diffusing relationship breaks down one claim into multiple, smaller claims.

Translating claims show the relationship between two claims from different domains. Similar interface features may be used in a PDA and a desktop, but the effects may not be identical within their context of use. This translating relationship highlights the connection between claims like those in the example, and may lead to more general insight about the effects of that feature.

Finally, mitigating claims are claims that manage the limitations of a different claim. One claim may have multiple downsides, and another claim has upsides that help counter those downsides. So the mitigating relationship flows in one direction and show the proposed solutions to problems.

This is not claimed to be an exhaustive list of claim possible relationships, but the authors believe it to be a useful start. Using these relationships introduce structure to the evolution of claims throughout a design process and should help designers track the important effects their interface features will have.

So by applying these relationships to the claims developed the scenario-based design of VaDeR, the evolution of the design was mapped. This not only ensured that all problem claims are addressed, but that the important contributions of the design were identified for evaluation purposes.

2.3 Constructivism

There are various pedagogical theories on how people learn. One such is called constructivism [2]. It says that a student will construct their own knowledge by actively reflecting on what they experienced and combining that with previous knowledge. This is in contrast to the student passively obtaining information by just reading a book or listening to a lecture. Ben-Ari [2] claims that integrating this theory into a computer science education will prove beneficial to students as it will allow them to form more correct mental models.

But UCS fails in supporting this learning technique because it does not allow for the creation of the most appropriate mental models, especially in examining an iterative design process. Through examining the key tasks that need to be supported to apply constructivism, VaDeR was designed to support those tasks and mitigate other related downsides of UCS. Specifically, VaDeR addressed the need for a reference tool within problem-based learning which is an application of constructivism.

2.3.1 Applied in the classroom

Parker and Becker [26] examined the difference between using a constructivist versus behaviorist approach to teaching a computer science class. Behaviorism is a simpler model than constructivism, as it deals with observable reactions to stimuli. Thus it fits a standard lecture-based approach to teaching. They found that students had to expend more effort using the constructivist approach, but there was no difference in the grading performance between that group and the group taught using the theory of behaviorism. This conforms to Ben-Ari's idea that the benefit may not be easily quantifiable. However, in the next class these students took, the students from the constructivist class had grades 18% higher than the other students. Parker and Becker suggest that the constructivist approach leads to increased performance by the students. It just may not be obvious early in the educational process.

One way to bring constructivism into a course is to use problem-based learning, as explained by Vat [45, 44]. This is where students are given a problem to solve and they learn in the process of trying to solve that problem. Computer science classes commonly use this approach by having students write a program or design an interface for a class project. Indeed, Vat discusses an approach to teaching human-computer interaction using a problem-based learning approach. Breaking down this approach into five stages, there is a stage for problem analysis, information gathering, synthesis, abstraction, and reflection. From these stages, the one that the teacher can easily support is the information gathering stage which is where students actively search for information relevant to their current goals. By pointing students to the appropriate references, such as books or websites, the teacher provides a knowledge base from which the student can draw from.

Savery and Duffy [36] also discuss the stages of problem-based learning. They emphasize the open-ended nature of this approach by allowing the student to determine what they need to learn and then proceeding to learn it. In support of those learning activities, reference materials such as case studies are provided. It is up to the student to decide how to use the case study and what knowledge they want to take from it.

Ellis et al. [15] reinforce the importance of information gathering for problem-based learning. They also insist there must be appropriate reference materials available to the student. This not only includes books and such, but interactive tools. Those interactive tools can even be visualization tools for the student to experiment with. Soloway et al. [40] also stress the importance of software tools for aiding problem-based learning.

UCS is a tool that can be used to aid problem-based learning. It provides a reference to past design efforts for the purpose of aiding a student's understanding of the design process and their own class project. Note that problem-based learning is not restricted to having a student attack an open-ended problem. They can also examine the problems others have encountered and how they were dealt with. Doing so still uses the constructivist approach of having the student use the reference materials to synthesize and reflect on past development records, instead of just being lectured to. It is still up to the teacher to design activities to encourage such reflection and not simply tell the student to read the record.

Problem-based learning, and how it would work within an HCI class, is discussed in more detail in the next section.

2.3.2 Problem-based learning and HCI

As in an example described by Vat [45], problem-based learning can be applied to courses where students must design and test an interface as part of their class requirements. These types of projects are common in human-computer interaction classes (HCI) [5]. By examining the stages of problem-based learning, it is seen how those design projects fit into them.

Problem analysis is the first stage. Students are presented a problem such as the need to develop a certain type of interface. Then they examine possible solutions based on their current knowledge and define what they need to know to approach the problem. This helps create an action plan for their proposed design activities. With HCI, they will realize they need to learn about the design process the course is teaching.

Information gathering is where the students find the knowledge they need to enact their plan. One way to learn about a design process is to look at case studies of previous design efforts using that process. With that, they can learn by example and see how steps progress in certain phases and the types of information produced.

After collecting that information, the synthesis stage occurs. The students combine the knowledge they gained from information gathering with their existing knowledge. Using this new information and perspective, they are able to re-examine the problem and adjust their approach to integrate what they learned. With the HCI example, students adjust their design process to reflect the process they are learning.

After finishing their solution to the problem, students enter the abstraction stage. By comparing the problem to other similar, or dissimilar, they are able to form generalizations that may be applicable to a wide range of problems. In the HCI class, students can compare multiple case studies along with their own efforts to learn about the design process in a more general sense.

Finally, the last stage is reflection. Students perform a self assessment to look upon what they did at a higher level and reflect on the new knowledge they have gained. This applies directly in an HCI class, as the student reflects upon what they learned about the design process either by performing it themselves or reviewing others through case studies.

2.4 Cases and Development Records

Development records can lead to case studies, so they are both described in this section along with their relationship. This gives an introduction to the case study model that will be focused on with the design of VaDeR, as opposed to all possible development records.

VaDeR is designed to improve UCS based on the key tasks that will emerge from analyzing UCS's place within problem-based learning. So the details behind UCS, including a motivation for its existence, are discussed in this section.

2.4.1 Development records

Whenever a design process occurs, there are related artifacts, such as writeups on background research, design rationale, prototypes, and evaluation results, as a few examples. Collected together, we can conveniently call these artifacts a development record. It provides a record of every tangible document or object that were involved in the design process.

A design team can create one by just collecting any related work they produce during their design process. At the end of the process, this will produce a collection that the team can look back upon. If this is not done during the process, it may be possible to create a development record after by finding the tangible artifacts from the process and collecting them together.

While this concept can apply to the production of any product, it especially applies to interface design within a specific process, such as scenario-based design. Here, the development record includes requirements analysis, scenarios, claims, prototypes, evaluations, and other related artifacts. Using this collection, an investigator might be able to follow and understand the process, but a case study would better support that task.

When it comes to publishing a development record, there is no standard as to what information should be included and how that information is best organized and presented. Instead, most development records are hinted about in part of a research paper suitable for journals or conferences. This imposes a narrative, descriptive structure around the data. While appropriate for readers whose goal is to learn about the theory or project, this approach prevents the reader from getting full access to all parts of the development record.

Thus, other methods are necessary. Case studies are one way to bring context to development records, and their presentation methods are well documented. So examining strategies for exploring them will lead to design possibilities to mitigate the downsides of those strategies and see what can be improved for their use in a particular context.

2.4.2 Cases

Cases in general are "evocative narrative descriptions of a specific real-world activity, event or problem." [9]. By being examples of real-world events, they are suitable for students learning about related processes. In other words, "they provide models of practice to students and other novice practitioners." [9].

They are widely used in academia for that purpose. Harvard Business School [37] classes are based around their own model of case studies, which provides a narrative description of a real-life business situation. Students study over 500 cases, analyzing what happened and focusing on what effects alternative decisions would have. Similarly, cases are used in medical [14] and law school [25].

Cases can be also be applied in computer science, as in the case of interface design. Wixon [49] proposes that cases be used to build up a collection of design knowledge. This is because he claims that real practice is the best place to extract valid design rationale. It also allows the case user to learn the resources used during the described process, allowing them to make informed decisions about whether they have the resources necessary to follow their model or if adjustments are needed.

With regard to presenting cases, Carroll and Rosson [9] emphasize that cases unfold in time and that any presentation should reflect that. To move cases towards more minimalist design, they also suggest that the volume of content be considered as large cases, presented improperly, can hinder the user's thinking about the case. About the structure of the case itself, it is also suggested that key design objects be drawn out, such as scenarios and claims for scenario-based design.

All of this relates to how a case is presented. As Carroll and Rosson say, "Just what and how case information is presented can have a significant impact on its effectiveness as a resource for active learning." [9] The design of VaDeR specifically addresses these issues.

2.4.3 UCS

As a supplement to their textbook [32] on scenario-based design, Rosson and Carroll created a website to present some case studies. It is called the Usability Case Study Tool [31] (Figure 2.1). Each case study is organized according to the defined structure of SBD. There are sections on requirements analysis, activity design, information design, interaction design, documentation, and usability testing.

The requirements analysis and usability testing sections then contain subsections on planning, methods and materials, information gathering, and synthesis. Each of the design and documentation sections contain subsections on exploration, envisionment, and rationale. All of this is organized in a hierarchical structure accessed through a menu on the website.

Within each of those subsections list a set of artifacts related to the topic. For example, the interaction design section contains possible interaction metaphors, interaction scenarios, claims, and prototypes, among other information. Therefore an artifact might be just a text document or it might be a screenshot, whatever is appropriate for that section.

While this approach does not leverage many information visualization techniques (as discussed in Section 2.5.1), it does at least present a structure around the data. And most importantly, it provides access to the lowest level details. The structure allows for the user to filter through data that is not desired and find a list of the desired data with only two mouse clicks on a website. With just one more click, the detailed data is available. This is an example of efficient filtering and access to information.

As an example of how UCS is currently integrated into a classroom, a class taught by the creator is described in the next section.

2.4.4 CS 3724: Introduction to HCI

During the Spring 2003 semester at Virginia Polytechnic Institute and State University, John Carroll taught CS 3724: Introduction to Human-Computer Interaction [5]. UCS was heavily integrated into many of the homework assignments.

For example, homework 1 asked the students to review the requirements analysis of a certain case study and then write their own problem scenario. Other homeworks required the students to access the various case studies and perform some tasks like review activity design, review interaction design, explore if the design phase addressed problem claims, contrast information prototypes versus interaction prototypes, and decide if the documentation effectively used a minimalist or systems model.

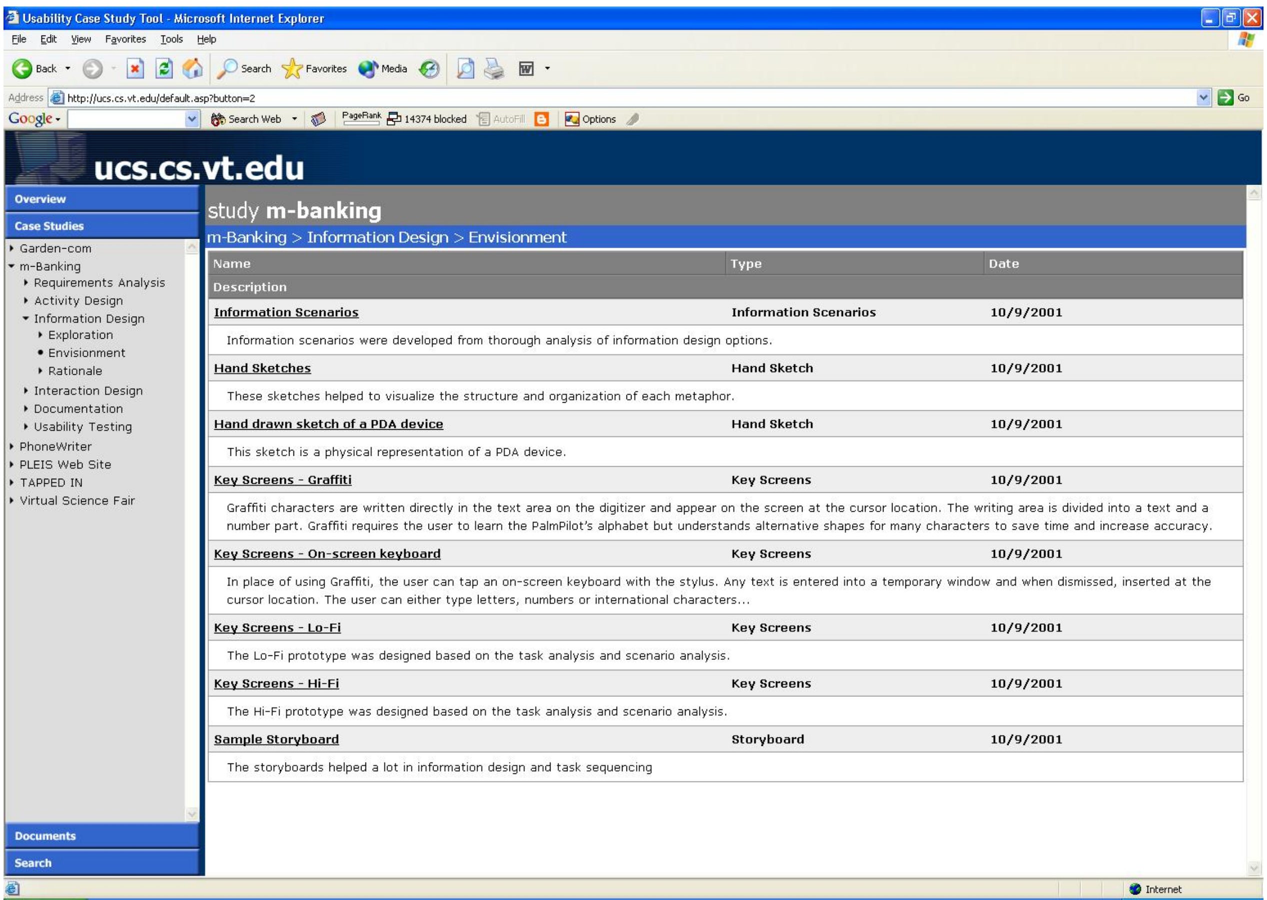


Figure 2.1: Usability Case Study Tool.

All of these tasks involve navigating and accessing the case study details as prerequisite for continuing with the activity. Assignments were not just limited to homeworks, but also led to class discussions and groupwork [33]. This fits with the constructivism theory discussed in Section 2.3 as the teaching is not just about lecturing, but forcing the students to reflect and discuss and the case studies in UCS. Problem-based learning is also connected here as the students, individually and within groups, try to address issues found within the case studies. Therefore UCS serves as a reference tool, which is important to the information gathering stage of problem-based learning.

2.5 Information Visualization

Since part of the interface project focuses on the visualization, concepts from the field of information visualization [11, 16] will be crucial to the success of the interface. This field focuses on the presentation of, and interaction with, data in an interface. A main goal is to allow the user to gain insight into the data, which is either impossible or extremely difficult with whatever methods are currently used to analyze that data.

2.5.1 General principles

As a central principle to designing an information visualization interface, Shneiderman [39] proposes a visual-information-seeking manta: overview first, zoom and filter, details on demand. An overview shows the entire collection of data in some representation. Perhaps, on a map, every house is shown as a point on the screen, or data is grouped where one point represents one neighborhood. From that overview, the user can zoom to gain more information about a specific portion of the data. For example, a city map could allow the user to zoom in to see a detail of the streets. Filtering allows the user to eliminate irrelevant data from their view. If the user only wants to view streets next to houses of a certain size, the visualization would only show that data.

Upson et al. [34] describes a process model for visualizing a set of data. First, if necessary, the data is filtered into a set of data suitable for visualization. Then, that data is mapped to geometric primitives than can be shown on a screen. These are marks or glyphs to represent the data and the attributes can alter the look of a specific data mark. Finally, the representations are rendered according to whatever rules are being used to map the data point and the corresponding attributes. This model highlights the most important part of any visualization, which is deciding what data to visualize and how to represent that data visually.

An overview and access to detail is required to comply with the mantra and using an overview + detail approach is appropriate. In this technique, the overview and detail are placed on the same display next to each other. One such example is a document reader [42] that has the entire document on the left, and a zoomed in section on the right.

When deciding how to map the data attributes to visual properties, some visualizations are easier to perceive than others. Cleveland and McGill [13] studied various methods on mapping quantitative data and came up with an ordering. They found that, in decreasing accuracy of performing quantitative tasks, the order is: position along a common scale, position along nonaligned scales, length, direction, angle, area, volume, curvature, shading, and color saturation. Thus, with respect to the supported tasks, the more important the quantitative data is, the higher it should be mapped within the order. Mackinlay [21] presented a similar ordering except for ordinal and nominal data. From his work, a simplified ordering is: position, color, shape, length, angle, slope, area, and volume. So there are guidelines when deciding how to map the various data attributes to a visual representation.

2.5.2 Inspiration from LifeLines

One example of an effective visualization is the LifeLines [29, 28] system developed at the University of Maryland. LifeLines focuses on personal histories similarly to how this project examines development records. A personal history is a collection of events and states that track key attributes of a person. Given as examples are such histories as medical records or financial dealings.

The authors take note of the problem that these histories are usually scattered and must be collected as one would a puzzle. Then the reviewer must browse through the collection to get an overall understanding, which can be difficult and time consuming. Problems like these are analogous to reviewing development records, as explained in Section 2.4.1.

Thus LifeLines (Figure 2.2) is their proposed solution. Starting with an overview, the information is arranged on a timeline, with different horizontal lines placed at different heights. There are rows that correspond to categories, so that the lines are not scattered arbitrarily. Each line represents some status, such as in jail, so the viewer can quickly determine when the person was in that status. Other lines that occur during the same time may represent related statuses, such as unemployment. Lines can also vary as to color or thickness to represent additional information. Icons can also be used to represent information, most likely one time events.

Zooming and filtering is supported by LifeLines. Each category of lines represents some hierarchy of facets which can be expanded or contracted to show more or less detail. Another type of zooming is what the authors call rescaling. The timeline shown on the screen can be altered to focus on some subset of the entire timeline, thus allowing the user to zoom in to detailed, dense sections. Filters then support the ability to just view certain types of information, hiding what is unimportant at the time.

Completing adherence to the information visualization mantra, details are available on demand. By clicking on icons or lines, additional information is brought up in a separate view. This could be a textual document detailing an arrest or a medical picture of an x-ray, for example. So the user is able to view as much data as they desire while using the LifeLines interface to guide them towards that data.

A key design choice to notice here is the main navigation strategy. It uses a zooming technique whereby the user can expand sections within the overview to gain a more detailed view. But this is used along with an overview + detail approach. The overview is always shown on the left side of the screen, and details appear in a separate area of the display. This is useful as it allows the viewing of the lowest level of detail without losing the larger context.

With multiple categories composed of singular events with a need to view detail, the general approach from LifeLines was useful in designing VaDeR. The artifact-based timeline approach was adapted to a timeline composed of scenario-based design artifacts. Having the details separate from the timeline overview was also an approach used in VaDeR. Section 4.2.1 discusses the design of VaDeR with respect to these techniques.

2.5.3 Exploring Elastic Windows

Another technique from information visualization that was instrumental in this work was the Elastic Windows concept [20]. This concept explored how tiled windowing compared to independent, overlapping windowing (as in pop-up windows). Instead of dealing with multiple windows that need to be moved and resized, Elastic Windows placed all windows into one area. Then resizing or closing one window resulted in the automatic adjustment of all other windows to fill up the space.

The idea was this windowing system would be more efficient for environment setup, window switching, and task execution than compared to using multiple, independent, overlapping

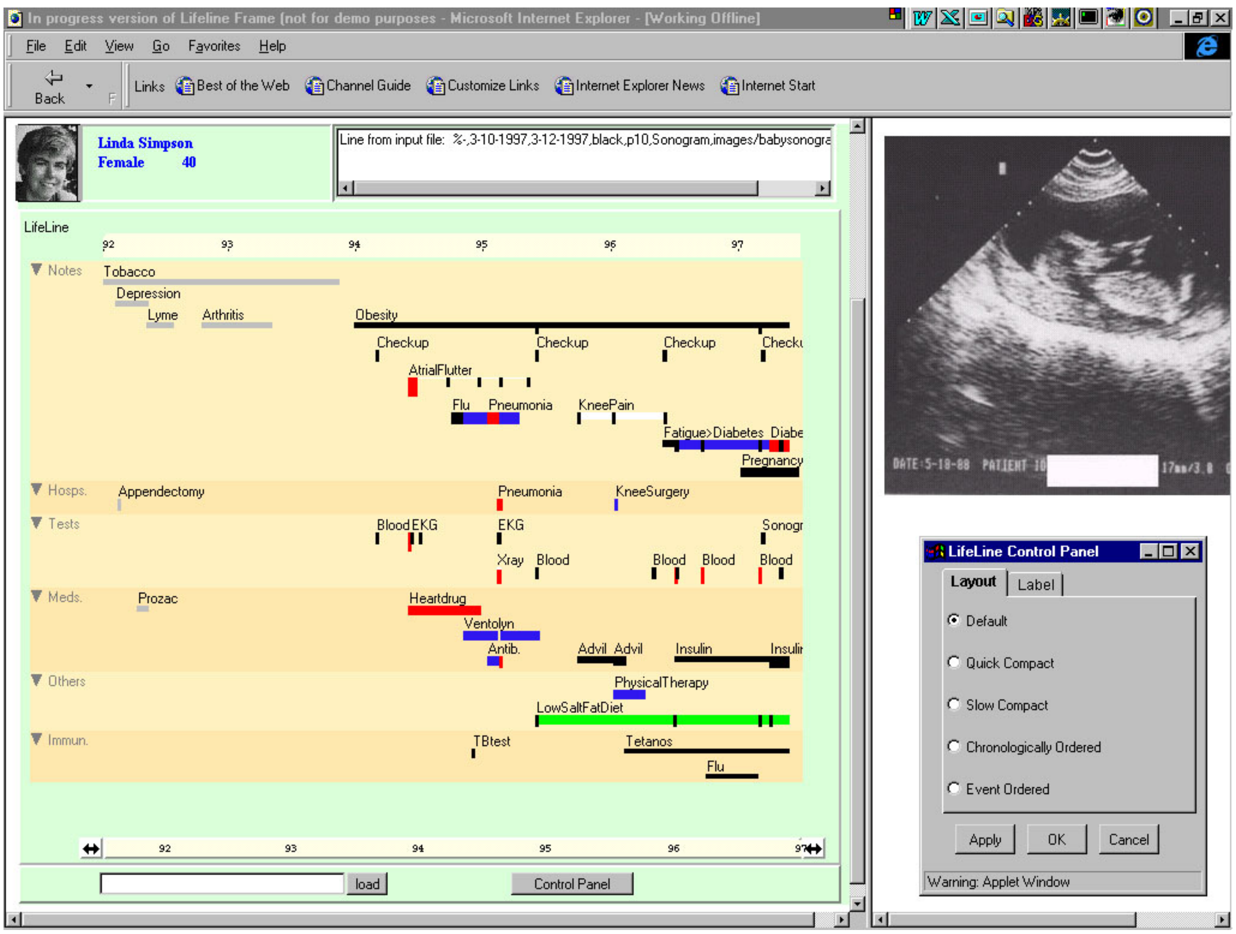


Figure 2.2: LifeLines for Visualizing Patient Records.

windows. Their four tasks were sequential scanning, comparison, determine context and scan, and recall context and scan. Of particular importance to this project were the sequential scanning and comparison tasks. Scanning involves looking through multiple windows for an attribute of the information and comparison is comparing attributes across multiple information sources. Note that to perform comparison, some amount of scanning is necessary.

In their evaluation, the Elastic Windows developers explored a wide variety of techniques within the tiled window domain. But we are not interested in their complex setup of many, hierarchical windows that can be resized or closed at will. It is simplified in this project because that complexity is not determined to be crucial to the design. But the core idea remains and the use of even a simplistic setup proved to be more efficient than pop-up windows.

They found that Elastic Windows met its goals and was significantly more efficient for all of the tasks, making for good claims on which to base our work. As we are considering a much more complex comparison task that is developed in Section 3.7, it was necessary to extend their work by combining it with other claims (see Section 4.2.1) and considering its utility as part of a complete system.

Chapter 3

UCS and Problem-Based Learning

This chapter describes the early development efforts of VaDeR, with significant focus requirements analysis phase. Towards the end the problem and scenarios and claims emerge, and those claims form the basis of what VaDeR addresses in Chapter 4.

It begins by examining UCS and its failures within the context of problem-based learning, including a class in which UCS was integral to the homeworks. This leads to scenarios demonstrating how one would use UCS to perform problem-based learning, and the breakdowns result from analyzing those scenarios with respect to the usability of UCS.

3.1 Approach

VaDeR must leverage the upsides of UCS while mitigating the downsides. By focusing on what UCS does well with respect to the tasks, VaDeR will attempt to not lose those advantages. But VaDeR must be designed to counteract the negative effects of UCS. The result will be an interface that is better able to support the tasks identified in the requirements analysis.

This will be accomplished by taking a scenario-based design approach. Requirements analysis identifies and explores the problem domain. Then the tasks within that domain are applied to the current tool and that leads to problem scenarios and problem claims.

Three design phases follow in Chapter 4 and attempt to address the problem claims and create a system that improves upon UCS. A discussion for each design phase is presented to demonstrate what the design decisions are. Then a scenario demonstrates the design in actual usage. Finally the claims analysis picks out key features and hypothesizes about their effects.

Throughout the process, it will be the claims and their relationships that push the design forward. Once the design is complete, those claims will help identify what is important for an evaluation.

Presented in this chapter is the background work and requirements analysis necessary before the VaDeR design process can proceed.

3.2 UCS Usability in the Classroom

Before designing VaDeR, the important tasks it must support need to be identified and explored. Thus we need to see why and how UCS fails in supporting problem-based learning. Discovering that will help reveal how VaDeR should be designed to maximize its beneficial impact upon problem-based learning.

So this section reinforces how UCS fits into a classroom environment. Then the usability problems are discussed based on what others discovered and a previous user evaluation. This leads into the problem scenario where those usability problems lead to problems when UCS is used within problem-based learning. Analysis of that problem scenario reveals the key tasks that VaDeR needs to support.

For teaching human-computer interaction classes, case studies have been shown to be useful [22, 33]. In these instances, development records can be viewed as case studies. But there is no standard method for presenting these case studies. One attempt, in the area of scenario-based design, is UCS [31]. This was discussed in Section 2.4.

But UCS is not without problems. Feedback that the researchers received complained about browsing difficulties related to the large amount of information available. Indeed, this was also found to be a problem by Somervell et al [41]. They found that there was no "...ability to reduce content to manageable pieces." So while there was a lot of information with an imposed organization, UCS remains hampered by its browsing and display functionality.

Students found the case studies on UCS as the best source of reference as compared to alternatives [41] and accordingly spent the most preparation time on those case studies. This indicates the ability of UCS to allow the answering of many questions. But just because it is able, does mean it makes it easy to answer those questions. Again, the possible insights into the data are restricted due to the presentation method.

3.3 Usability Problems

By applying UCS to problem-based learning within the context of a HCI class, a problem scenario can be written to explore the current usage and the problems that exist. This will focus where UCS fails in the stages of problem-based learning, guiding towards what type of improvements would be most beneficial.

Notice that UCS is comprised of three main features: document-centered design, organization of those documents, and the detail display of a document. Basing the system around singular documents, or artifacts, is a choice made to conform to the types of information a development record produces. Those artifacts are then organized into hierarchical menus, with each lowest level menu having a list of artifacts. That list is ordered by document type, with a title and date included. Accessing an artifact entails clicking on its link which produces a pop-up window.

Therefore the selection of an artifact means the user goes through three steps. First, they use the first view they see to decide which area of the case study they want to investigate further. Next, they remove the irrelevant information from their view by ignoring that information or literally hiding that information from their display. Last, they find the artifact they desire and access the detail.

UCS is not ideal at supporting any part of that process. Initial heuristic evaluations (data in Appendix A) gathered feedback such as "exploration part was a bit confusing", "Information was presented logically in the tree, but the pages were not so intuitive", "When I click on something say under Information design it will open in a new window. Then I will not be able to see the definition of Information design which I might want to refer to" , "Design is quite minimalist... It probably needs a little bit more to distinguish the difference between objects.", and "It is hard to distinguish objects, so details are harder to find than they should be."

Even the UCS creators acknowledge problems with it. They note that even though design is iterative and emergent, they imposed an artificial "phase-oriented hierarchy" [33] for simplification purposes. Using that hierarchy in combination with a document-centered view makes it difficult to follow any iterative process. For example, if a user wanted to follow activity design starting with initial design, prototyping, evaluation, results that lead to design changes, more prototypes, more testing, and so on, they would have to go back and forth between multiple levels of the hierarchy and use artifact dates as a guide.

Also noted by the creators were that students complained about "problems with browsing...or simply comprehending the very large amount of information presented by each case." [33] This corroborates the comments from the initial evaluation as noted earlier.

3.4 CS 3724 and UCS

As discussed in Section 2.4.4, UCS was used in a human-computer interaction course. The assignments there provide examples as to how students actually used UCS. There are six assignments in particular that required UCS. Each involved opening a specific example and following the instructions. The key features of these assignments are summarized in Table 3.1.

Table 3.1: Assignments in CS 3724.

1. Examine requirements analysis, write own problem scenario and claim.
2. Explain requirements analysis and activity design, explain how the activity metaphors clarify the requirements and suggest starting points for activity design.
3. Find issue in interaction design, analyze tradeoffs, describe if and how the tradeoffs were resolved.
4. Argue how well the problem claims were addressed in the design.
5. Describe and contrast how prototype was used in information design versus interaction design.
6. Explain how minimalist or systems model was used in documentation design.

For someone learning how to perform scenario-based design, these assignments help guide them to learn from examples how to apply the process. They also represent the type of tasks that a student undergoing problem-based learning would likely perform. For example, a user not familiar with writing scenarios would likely perform the example-based learning of assignment 1. If they want to know how a design is supposed to address problem claims, assignment 4 is appropriate. These types of assignments help inspire the problem scenarios in Section 3.5 and lead to analysis of what tasks are important.

They also suggest design elements that VaDeR needs to investigate. As in assignment 4, it would be helpful for the user to do a side-by-side comparison of problem claims and design claims. Finding both prototypes in assignment 5 would be aided by making it efficient to access that information.

With each assignment, there is also the searching that a user must perform to find the desired artifacts. Since there can be a lot of information in a single case study, it will help to examine information seeking strategies from prior information visualization work.

3.5 Problem Scenario with Problem-Based Learning

Putting all of the usability issues from Section 3.3 together in the context of actual class usage from Section 3.4 results in the problem scenario in Section 3.5.2. This problem scenario uses the stages [45, 36] of problem-based learning. Examining these stages in detail shows the process flow that a student goes through.

3.5.1 Problem Analysis

A student enters the problem analysis stage with an open-ended problem. They start by brainstorming about possible solutions and identifying what is important. Whatever the student feels they should know, but do not currently have a good understanding of, is identified as a learning issue. An action plan is then developed to address those learning issues and solve the problem. This is shown in a problem scenario:

> Derek needs to design a sports score notification system for his HCI class and he is required to use scenario-based design. His initial idea is to make a small display for a desktop system, but any design he makes will have to come out of the required process. Since he is still learning about the process, he knows he needs to know more about how the design occurs. Class has already taught him the

> basics behind scenario-based design, such as terminology and concepts, but he is not sure how everything fits together. He is confident he knows enough about requirements analysis, but is unclear on how to perform activity, information, and interaction design. So his learning issues are those design phases. Once he learns about those design phases, his plan is to follow by example to ensure he does not miss any important steps and that the work he performs is indicative of scenario-based design.

The main learning issue Derek identified is with the design phases. He needs to know how they are used so that he can use them in his work. Without that knowledge, he will be unable to apply scenario-based design, and will not meet the requirements of the class. His professor has provided the class with UCS as a main reference source.

So far, Derek is doing fine. But he will run into problems with the next stage: information gathering.

3.5.2 Information Gathering

Learning issues play into information gathering. Now the student will use any relevant references available to gather and explore the knowledge necessary for meeting those learning issues. So this is where the student learns enough to feel ready to attempt a solution. The student exits with the new knowledge necessary to implement their action plan. Derek's attempt at this stage is shown in a problem scenario:

> Derek uses UCS and opens up a case study that used scenario-based design. He sees a list of categories including requirements analysis and the design phases. Since his class already covered requirements analysis in depth, he is currently more interested in the activity, information, and interaction design. Starting with information design link (because Derek likes pictures), he clicks on the appropriate link. This reveals the subcategories of exploration, envisionment, and rationale.
>
> Now he is confused and is not sure where he can find the information scenarios. Figuring that scenarios explore design possibilities, he clicks on 'exploration' and

a list of artifacts appears on the screen. Seeing nothing that indicates a scenario, Derek then chooses ‘envisionment.’ Reading down this list, he sees artifacts for sketches, but finally sees ‘information scenarios’ at the bottom of the list. Despite having an earlier date, the scenarios are listed last and Derek has no idea why the artifacts are ordered the way they are. He manages to notice that they are actually ordered by type, but that does not strike him as particularly meaningful.

Wanting to find the claims next, he looks through the other categories in information design until the claims are located. Derek then becomes curious as to what information prototype was created after the claims were written. He goes back to other categories and finds a list of prototypes, then examines the dates against the date of the claims. Looking down the list of dates, he mentally compares each date to the claim date, and finds the earliest prototype. Interested in the design, Derek wants to know what the final evaluation results were for the information prototype. He clicks on the usability testing menu and looks for the testing related to information design. Finding a few, he compares the dates between each one until he is sure he has found the latest one.

Moving on to another design phase, Derek clicks on the link for ‘interaction design’ and sees another list of artifacts, none of which say ‘interaction scenarios.’ Instead he sees ‘online shopping scenario script.’ So he reads that short script and decides it is not enough and he wants to read longer documents. So he clicks on the ‘exploration’ category and sees ‘interaction metaphors.’ He reads those and is much more satisfied with the amount of information. Expecting to find claims in the ‘rationale’ category, as there are in the information design, he looks in that section but finds no such claims.

Now wanting to know about the metaphors used in interaction design, Derek clicks on the link with that title. This brings up a new window with the text about those metaphors. He reads through them and now wants to see the artifact about interaction technology options, but does not want to close the current window because he wants to compare the two artifacts to see if he can relate the two. Minimizing the metaphor artifact, he again sees the list of artifacts and clicks on the technology options link. Another window pops up. Derek now brings back the window with the metaphors, and moves both windows so that they are next to each other. He can now easily look back and forth between the two windows, trying to think how he would use the metaphors and technology options together to create an interaction design.

Continuing on in this manner, Derek slowly navigates the sections. He is unsure of what each category contains until he selects them, and then is unsure of the ordering scheme. But he pushes on and is able to read the documents he wants. Later on he wants to look back at a document he already read, remembering there was something important in it. But he can’t quite remember which was one it was, only that it was in the information design section. So Derek goes

> back and clicks on the artifacts until he finally finds what he wanted.
>
> Eventually, Derek thinks he has a grasp on the process of the design phases. He had to pay close attention to the dates of the artifacts to understand what came before what, but he realizes the difficulty of placing those artifacts in order and does not bother spending too much time worrying about it. Having to deal with multiple pop-up windows was frustrating for Derek, as it made it difficult for him to compare and contrast multiple artifacts. Derek also wishes there was a better way to search through the case study data without encountering so many dead ends or not finding what he expects in a category.

Derek ran into many problems. It took him a while to find exactly what he was looking for, and in that process he started losing the context of what he was viewing. The organization of the data also made it difficult to see the sequential and iterative nature of design, which means his understanding of the process can easily be incorrect. With a shaky, and incomplete, knowledge base the next stage is also affected.

3.5.3 Synthesis

Once the student has the knowledge gained from their information gathering, they enter the synthesis stage. Using their newly acquired knowledge, combined with any previous knowledge, the student re-examine the problem and their approach to it. Further work on the problem reflects what they learned in information gathering. Derek's problems in information gathering spill over into the synthesis stage as shown in a problem scenario:

> Derek thinks he has a good enough grasp on activity, information, and interaction design to continue with his project. He tries to model his work after the example he saw, such as making sure he creates appropriate scenarios, uses claims to analyze key features, and brings in HCI knowledge to back up his design decisions. Doing so, he is able to continue his work on creating a sports score notification system. Unfortunately his frustrations with UCS cause him to periodically refer back to ensure his project is not skipping any steps or leaving out important details.

> Because of the way UCS structures the information, Derek does not notice that he should be performing an evaluation and redesigning based on the results. Instead, he just performs an evaluation and moves on. So his process does not match what his professor intended him to perform and his learning progress suffers in conjunction.

3.5.4 Abstraction

After the work is completed, the student is able to compare the problem to other problems for similarity and make generalizations. Coming out of this abstraction stage is a new perspective on how a broad range of problems is related to each other.

Derek never bothered to look at multiple case studies through UCS due to his frustrations just using one. But now he is forced to since his class test will evaluate his general understanding of scenario-based design. So it is back to UCS:

> Since his project is complete, Derek wants to go back and review how his work compared to others in preparation for his test. In the course of doing so, he compares those other projects to each other. Using UCS, Derek realizes that comparing two case studies is difficult since he can only open one menu at a time. And every time he uses UCS, he still runs into the same problems encountered during his initial information gathering. Thus trying to compare two case studies is annoying to Derek. He is therefore slow at developing generalizations across the general problem domain and the process of scenario-based design.

Trying to compare multiple case studies was very difficult for Derek. He could have easily missed important similarities or major differences that would have led him to make more realistic generalizations. But with no other options, he pushes on into the reflection stage.

3.5.5 Reflection

Finally the reflection stage is any sort of assessment based on the work the student performed, especially what they learned from information gathering. Since the student was the one who identified what needed to be learned, the assessment is based off of that.

> Derek's professor gave him a test based to evaluate his knowledge of scenario-based design. Because of his frustrations with UCS, Derek's knowledge of how the design phases progress is limited and his performance on the test is less than what he hoped for. His project grade also suffered because he did not fully grasp how to implement those design phases and combine them with evaluations.

Derek's journey through problem-based learning is finished. He was not able to learn as much as he intended as evidenced by his performance. Pinpointing the main point of failure leads Derek to UCS. Every problem he encountered led back to his issues in information gathering. He wishes he had a reference system better equipped to handle his needs.

3.6 Problem Scenario Analysis

Since UCS is a reference tool, it is not surprising the main breakdown is with information gathering. Problems within information gathering can lead to more problems in future stages. Therefore addressing the issues in that stage will be most beneficial. A summary of the stages is presented in Table 3.2.

Based on the problem scenario, which was inspired by usability issues noted by others, there are three key tasks that need to be further analyzed. Since a development record is a series of sequential events and the iterative nature of events is hard to see with the phase-based hierarchy, the first task is organizing the artifacts by time. Concurrent to that is the way in which the hierarchy is navigated. Since selecting menus within an organization level is what

UCS supports, the second task can be referred to as filtering artifacts based on attributes. After the desired artifact is found, getting the full detail within it is the final task.

By analyzing why these tasks are so important and what about UCS fails in supporting them to their full potential, a new system can be designed that will better support these key tasks. In turn, this will aid information gathering which helps prevent problems in the later stages of problem-based learning.

3.7 Tasks

Each task introduced in Section 3.6 is now analyzed for their importance and the ways the UCS fails in supporting them. A problem scenario is presented for each to illustrate how the task is performed. Each of these takes place within the information gathering stage of problem-based learning.

3.7.1 Use temporal ordering in search

Being able to put the events of the development record in order is a step towards putting all of the data into context. This corresponds to getting an overview of the data as recommended by Shneiderman [39]. When this is done, the student is able to follow the events from one to next, creating a narrative story within their mind. It also provides a consistent viewpoint for discussion amongst multiple students, as everyone will understand how the design process progressed.

Mostly it helps to answer some basic questions about the events. A student can look at one artifact and wonder what came next or previously. They can then track how a design progresses, from scenarios to claims to prototypes and so on. Events that occurred early or late, or any other general time period, can be easily identified. In general, performing this

Table 3.2: Stages of problem-based learning, context, and UCS breakdowns.

Stage	**Entry**	**Exit**	**Example**	**UCS breakdown**
Problem Analysis	open-ended problem	brainstormed solution, learning issues, action plan	desktop display, need to know about information design, modify own plan to fit scenario-based design	not applicable
Information Gathering	learning issues	new knowledge to aid in action plan	learn information design	usability issues in Section 3.3
Synthesis	new knowledge	re-examination of problem and action plan	alter process to match SBD	incomplete or inaccurate knowledge from information gathering
Abstraction	finished solution	generalizations about various problems and solutions	see similarities and differences in use of information design	difficult to compare multiple case studies
Reflection	finished all work	assessment of knowledge gained	realize knowledge of how to perform information design successfully	based on problems in previous stages

task creates an overview of the data that allows the student to know where to look for the artifacts they are interested in finding.

Users reported they had problems distinguishing artifacts did not quite understand how to navigate the information. This is also connected the problems the UCS creators noted in that their organization hides the iterative nature of design, as discussed in Section 3.3. Performing this task addresses those issues. Time is a unique, distinguishing feature among many of the artifacts. And organizing by that allows the user to follow the iterative process of design by piecing together the progression of design.

This task is demonstrated with a problem scenario in the next section. Derek is attempting to perform this task with UCS and his experiences are not ideal.

3.7.2 Problem scenario

> Derek is not sure how to perform information design as he is required to do so for his class project. He does not fully grasp the progression of events and how iteration factors into the design process. So he goes to UCS and opens an example case study. Going to the information design section, he browses the artifacts just to get a sense of what is present. After finding the information scenario, he becomes curious as to what occurred next. Not having a full understanding of scenario-based design, he is not sure where to look for that next artifact. Derek is forced to use the dates he sees besides each artifact to find the one that comes right after the scenario and he must go through each menu to ensure he has found the correct one.
>
> Continuing his search in this manner, Derek slowly pieces together the events of information design. Unfortunately, he misses the evaluations performed on the information prototypes because he was not looking in the evaluation section. Therefore he remains unaware of the reasons he sees a progression of information prototypes in the information design section.
>
> Modeling his own project activities after what he understands, Derek is able to go from scenarios to claims to prototypes. But he only understood this progression after comparing their associated design and wasting much time in the process. And he misses out on how formative evaluations are integrated into the information design process. Derek's understanding of scenario-based design and

his project suffer because of that.

3.7.3 Filter artifacts based on attributes

With the large amount of data available from a development record, it is helpful to filter out irrelevant data when looking for a specific artifact. UCS allows this with the case studies by placing a hierarchical menu organization upon the artifacts. When the user is looking at the contents of one menu, the contents of all the other menus are hidden and the user can only read the title of their highest menus. But this has the drawback of removing the artifacts from their context when combined with all of the remaining artifacts

These filtering operations allow the student to quickly focus in on the data they are interested in. It is the equivalent of the student issuing a dynamic query upon the data. Dynamic queries have been shown to be useful in information visualization [38]. And this also meets the requirement of Shneiderman's [39] insistence on supporting some type of zoom and filter operation.

UCS presents a top-down model of filtering. A few attributes are given, and the user must increasingly select additional attributes to get to the details. This method makes it very difficult for a user to realize how much data a case study contains. A user would have to open each menu, something they can only do one at a time. Organization like this does not readily hint to the user what sections were emphasized in the design process. Perhaps the design team spent most of their effort on interaction design, and relatively little on everything else. Without going through every menu, a user would never know this and it is this inefficiency that inhibits the usefulness of the development record.

With a similar situation from the previous task's problem scenario, Alex is attempting to filter the data to find what he desires in the next section's problem scenario. He, like Derek, runs into problems.

3.7.4 Problem scenario

> Alex is in the same HCI class as Derek. He needs to study the information design and interaction design of a case study. On UCS, he clicks on the Garden.com link and this brings up a list of categories, two of which are for information design and interaction design. First choosing information design, he clicks on the link and this reveals the subcategories of exploration, envisionment, and rationale. Now he is confused and is not sure where he can find the information scenarios. Figuring that scenarios explore design possibilities, he clicks on 'exploration' and a list of artifacts appears on the screen. Seeing nothing that indicates a scenario, Alex then chooses 'envisionment.' Reading down this list, he sees artifacts for sketches, but finally sees 'information scenarios' at the bottom of the list.
>
> Clicking on the link for 'interaction design,' Alex sees another list of artifacts, none of which say 'interaction scenarios.' Instead he sees 'online shopping scenario script.' So he reads that short script and decides it is not enough and he wants to read longer documents. So he clicks on the 'exploration' category and sees 'interaction metaphors.' He reads those and is much more satisfied with the amount of information. Expecting to find claims in the 'rationale' category, as there are in the information design, he looks in that section but finds no such claims.
>
> Continuing on in this manner, Alex slowly navigates the sections. He is unsure of what each category contains until he selects them, and then is unsure of the ordering scheme. But he pushes on and is able to read the documents he wants. Later on he wants to look back at a document he already read, remembering there was something important in it. But he can't quite remember which was one it was, only that it was in the information design section. So Alex goes back and clicks on the artifacts until he finally finds what he wanted. After finishing up his work, Alex wishes there was a better way to search through the case study data without encountering so many dead ends or not finding what he expects in a category.

3.7.5 Access artifact data for comparison

Accessing the lowest level of detail for the artifacts is the final task. Without this ability, students will never be able to read scenarios, see prototypes, or do anything else requiring that level of detail. It is a relatively simple task as compared to others, but it still presents interesting design choices. The detail can be placed in a pop-up window, next to the overview, on another display, or other possibilities.

The importance of this task is more obvious than the other two for without the details the student could never fully understand the development record. It completes Shneiderman's [39] information visualization mantra by allowing details on demand. UCS handles this by popping up a new window when the user clicks on an artifact link. As many windows as the user desires can be open and they are handled by whatever operating system the user has.

Using pop-up windows is potentially distracting and annoying the user. A window will cover up whatever the user was currently looking at, taking away the context of the artifact. It also discourages accessing multiple artifacts at once as then multiple windows have to be dealt with. Testers did report their displeasure and confusion with pop-up windows as discussed in Section 3.3.

Again, this task is demonstrated in a problem scenario. In this one, Joe is forced to work harder than is really necessary to compare two artifacts.

3.7.6 Problem scenario

> Joe is in the same HCI class as Alex and Derek. He needs to study the interaction design of garden.com. So he clicks on the interaction design link and sees the submenus of exploration, envisionment, and exploration. Starting with the exploration, he clicks on that link and gets a list of the artifacts within that category. Joe sees a link for interaction metaphors and clicks on it. This brings up a new window with the text about those metaphors. He reads through them and now wants to see the artifact about interaction technology options, but does not want to close the current window because he wants to compare the two artifacts to see if he can relate the two. Minimizing the metaphor artifact, he again sees the list of artifacts and clicks on the technology options link. Another window pops up. Joe now brings back the window with the metaphors, and moves both windows so that they are next to each other. He can now look back and forth between the two windows, trying to think how he would use the metaphors and technology options together to create an interaction design.

3.7.7 Claims

With all of the problem scenarios written, they can be analyzed for features. Those features can then be hypothesized about with respect to their upsides and downsides in relation to the tasks. This is the claims analysis of the problem scenarios. Together, all of these claims show how UCS supports the tasks. Table 3.3 summarizes the claims analysis.

From this analysis, it is seen that the first two tasks have the most room for improvement. It is difficult to order any collection of artifacts by time. Filtering works, but it is too easy to get lost in the menus since you can only go by titles and can only open one menu at a time.

3.8 Initial prototype and evaluation

Before the final design is discussed, it is prudent to show the prototype that inspired this project to move forward. It was created for a class project [4] in Information Visualization of Fall 2003, taught by Chris North. The intent was to create a basic design, prototype it, and test that design for usability issues against two other interfaces. If testing showed the prototype was not a regression in usability, then it was worth the further effort to improve the design. This work was accomplished in tandem with Jamie Smith and Shahtab Wahid, also in the class.

It was mostly the data layout that was being studied. Interaction was limited to clicking on icons. A main difference for this prototype is that clicking on an icon zoomed in on what data icons it represented, which could then be clicked on for the actual data. But otherwise, the interface layout basics are similar to the design presented in this chapter

Table 3.3: Claims for problem scenarios.

Situation Feature	**Upside and Downsides of Feature**
Date associated with each artifact	+ shows when artifact was created
	+ can put events into temporal order
	- artifacts not ordered by time, so takes effort to understand progression
	- difficult to look for artifacts based on time
Organization of data into categories	+ allows focusing of search
	+ demonstrates what type of information is available
	- organization may not be clear
	- can not know what is in category until is opened
Navigation based on clicking through hierarchical menus	+ efficient filtering of information
	+ simple interaction method
	- lose context of bigger structure
	- menu titles may not be meaningful to user
Access to detail by clicking link	+ allows full detail to be shown
	+ simple interaction method
	- opens up new window, covering the other view
	- opening up multiple windows requires more work to arrange

3.8.1 Interfaces

We first chose two common interfaces in current use. Note that every interface was given an arbitrary letter label for the purposes of testing. First, we chose a standard textual document. Many case studies are presented in standard textual forms. Since the ACM uses PDF files, we chose Acrobat Reader and a PDF file. This is labeled interface B and is pictured in Figure 3.2. Next, we created a webpage modeled after the Usability Case Study Library (UCS) [31]. This organizes the case study information into scenario-based design sections and forces a hierarchy on the data. At the lowest level, users can access artifacts and some details on the project described in the case study. This is labeled interface C and is pictured in Figure 3.3.

Finally, we have our own prototype created in PowerPoint. Here, there are three main categories: Design, Implementation, and Evaluation. A timeline stretches across the top of the interface. Icons representing events are placed within the relevant category and at the correct place on the timeline. When a user clicks on an icon, a detailed view of what it represents appears right below the icon. Subsequently, the user sees more icons that represent artifacts within a zoomed-in timeline section just for that selected event. Clicking on these new icons brings up the details in the lower portion of the display. The details range from design documents to prototype screenshots to evaluation materials, and anything else the case study contains. Notice this prototype starts with an overview process, allows the user to zoom and filter unto a specific event, then provides details on demand. This prototype is labeled interface A and is show in Figure 3.1.

Each interface contained the same basic data taken from a single case study of an interface. Interface B had the research paper written about the interface, while interfaces A and C used data taken from that paper and the author's knowledge. The actual data was not crucial, as the experiment was testing the interface usability, so it's important that the data did not influence the results.

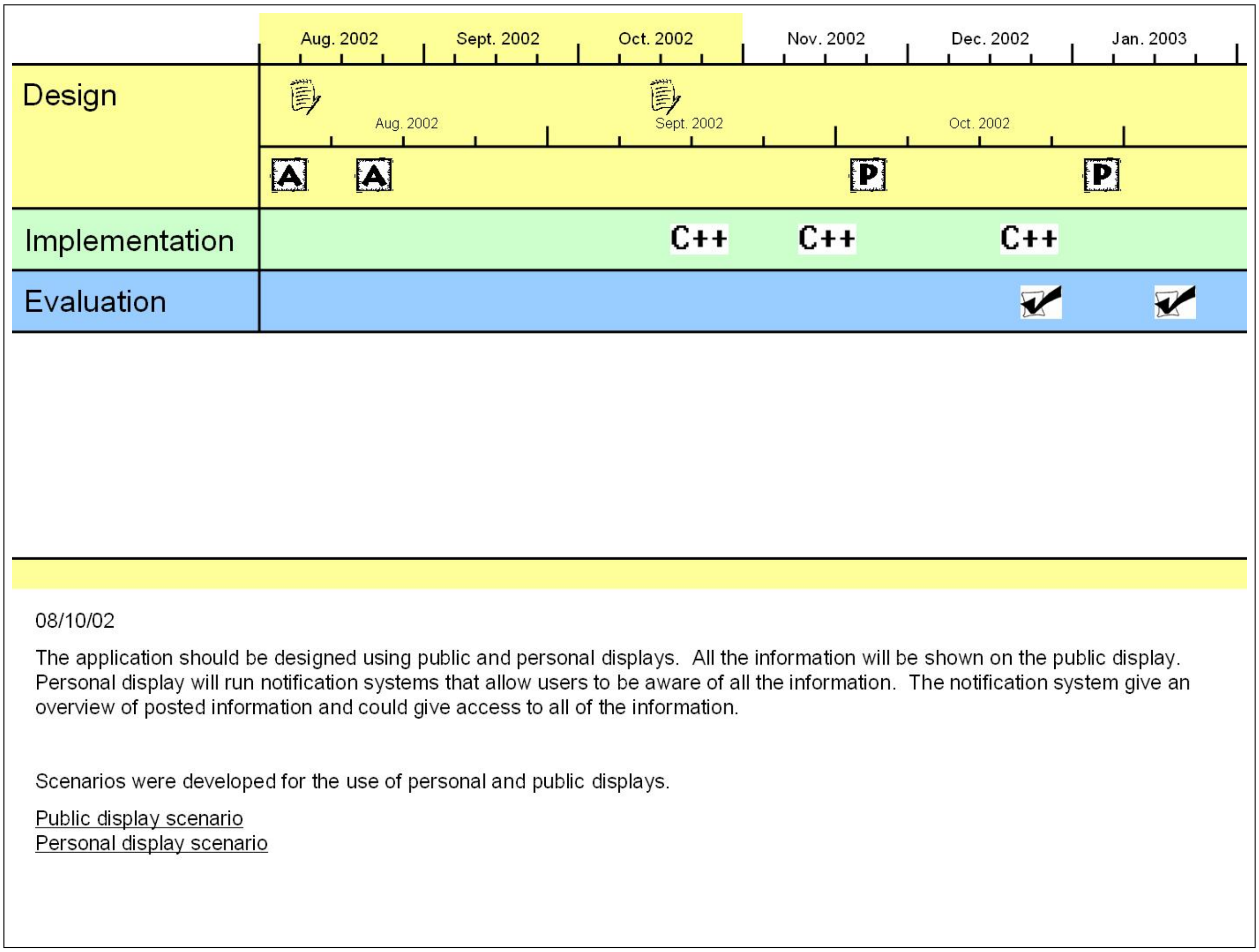


Figure 3.1: Interface A: Prototype in PowerPoint.

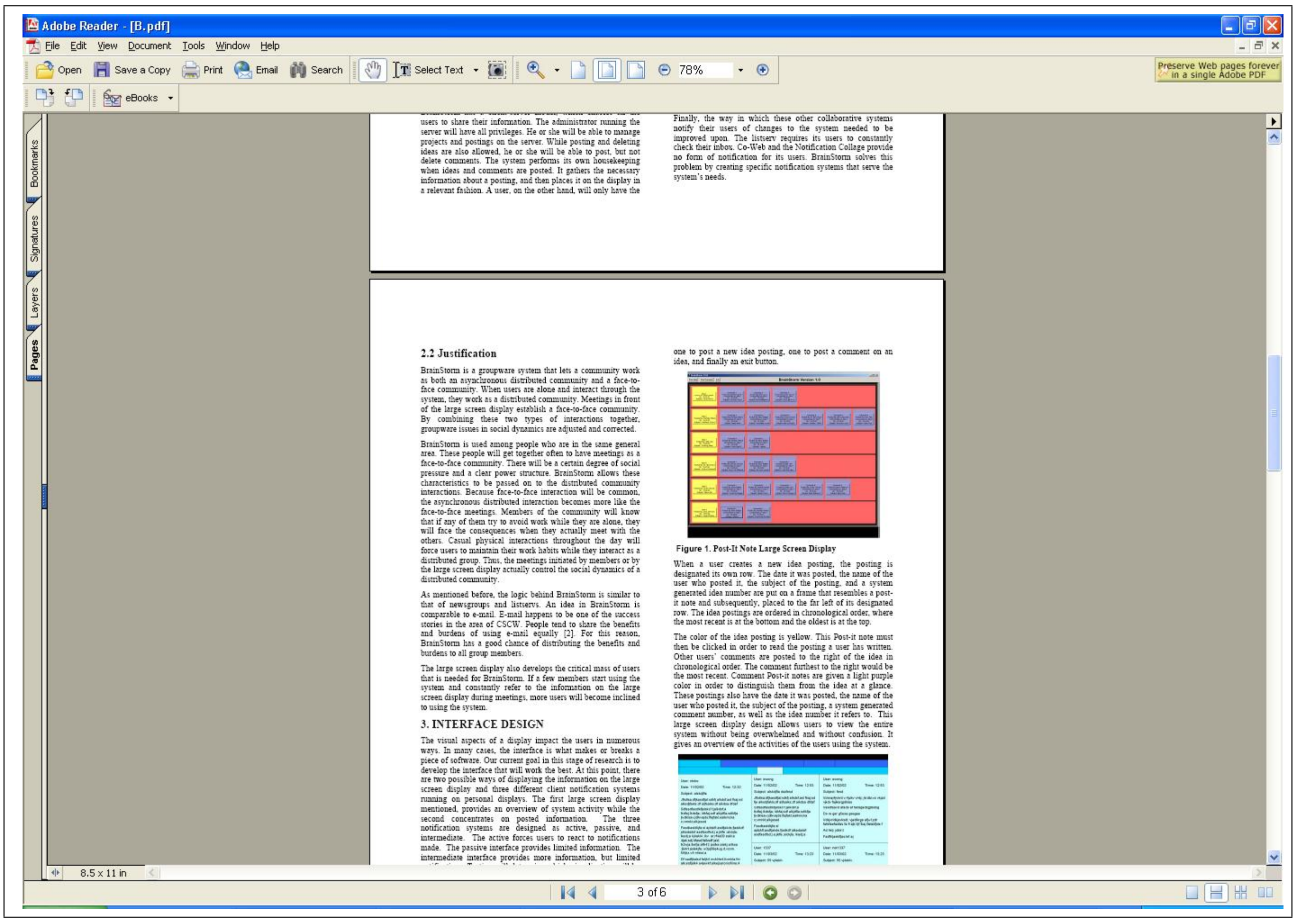


Figure 3.2: Interface B: Acrobat Reader.

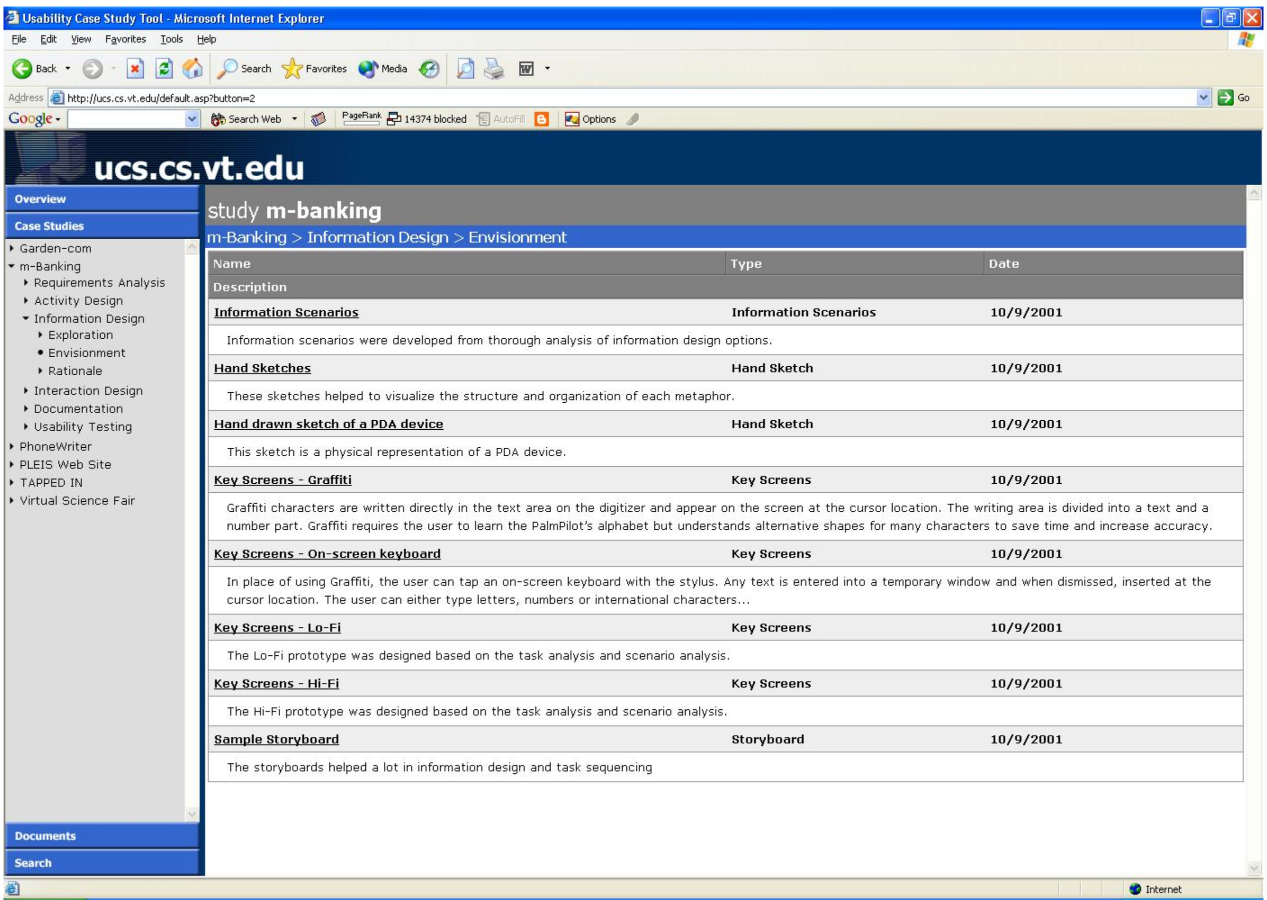


Figure 3.3: Interface C: UCS.

3.8.2 Experimental design

The null hypothesis is that there is no difference between any of the three interfaces with their ability to adhere to the heuristics and guidelines we are evaluating with. Our intent was to investigate whether this hypothesis could be disproved through our user testing.

We wanted to compare the usability of the three interfaces. Heuristic evaluations [23, 3] provide a useful method to test for adherence to general usability guidelines and to find major problems with the interface. As mentioned earlier, the new interface is only a basic prototype, so precise task evaluation and task timing is not useful for this current experiment.

Therefore, our independent variable is the interface being evaluated. The dependent variable is then the responses to the form we provide based on heuristic evaluation.

We chose a relevant subset of Nielsen's heuristics [23]. Then we created a set of four guidelines that were specifically related to this project and the tasks we wanted to support. These heuristics and guidelines now are shown in table 3.4.

A form was created to collect the desired information. Each user was allowed to use an interface for as long as they desire, and ask questions to the evaluators. They then filled out the form for that interface. For each heuristic or guideline, they rated their response to the statement "The interface follows this guideline" with the choices being among Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree. Those responses map to integer values -2,-1,0,1,2 respectively for the purposes of data analysis on quantitative data. This data points to where the interface met certain guidelines and where it failed at meeting certain guidelines.

Next, the user describes, in words, why they chose their response by pointing out specific elements of the interface. This allows for the collection of qualitative data. These comments are usually very useful in learning precisely what is usable and not usable with a system and points the way toward possible improvements.

Table 3.4: Heuristics for testing initial prototype.

- 1.1 Visibility of system status
- 1.2 Match between system and the real world
- 1.3 User control and freedom
- 1.4 Consistency and standards
- 1.5 Recognition rather than recall
- 1.6 Flexibility and efficiency of use
- 1.7 Aesthetic and minimalist design

- 2.1 Overview: I was able to achieve an overall understanding of the events that occurred.
- 2.2 Navigation: I understood how to navigate the case study using the interface.
- 2.3 Details: I can access any details that I want in an efficient manner.
- 2.4 Organization: I was able to easily understand how the information was organized.

Each participant repeats this process for each interface as they all evaluate every interface. For each participant, an order of the interfaces was given. This was different for each new participant and ensured that very few people would evaluate the interfaces in the same exact order. This minimizes any learning curve as a factor in the experiment.

3.8.3 Results and analysis

There were 17 participants. The quantitative data indicated no significant difference between all three interfaces. This could be because all of the interfaces were too simple to reveal major differences or that the experimental design was not enough to reveal the differences.

Free response comments gave more useful feedback about the interfaces. Evaluators generally liked UCS's simplistic design, but had problems with it opening new windows and covering what they were looking at previously, found navigation sometimes confusing, and could not find much distinction between the artifacts without accessing their details. With the VaDeR prototype, evaluators approved of the minimalist design and found it easy to use after a few minutes of exploration. Some had problems with not knowing what the icons mean, as this was not explained. Of the problems found, the icons were found to need the most improvement. This was not unexpected, as the goal was to compare the basic data layout. But this is considered more in the current design.

With these results, we concluded that VaDeR showed promise for further development and was not a regression in usability as compared to two other interfaces. The thinking was that with further development, VaDeR would prove to increase efficiency and simplify the process of gaining insight into the data. It was also thought that because it used information visualization techniques, it would scale better for future development. Thus VaDeR was chosen for continued development with the goal of maintaining the upsides of UCS while providing the ability to integrate the long descriptions that a written description can provide.

3.8.4 Reflecting on the claims-centric scenario-based design approach

This initial development style was not as successful as we had hoped. Was this because of the scenario-based design process? Did the claims focus help or hurt the success of the effort. Would a more rigorous approach lead to greater success?

In some ways, it seems that the scenario-based design approach, and in particular the claims focus, may have steered the project team away from some important issues. Because scenarios focus only on a particular path through a system, by definition other paths are ignored. As even noted by the creators of the scenario-based design approach, it is difficult to know how many scenarios is sufficient and when a developer can stop writing them. [32]. A claims focus does not help with this problem, in fact, in some ways it may increase the problematic effects by creating many small focus points, thus making it difficult to make broad conclusions about the overall usability of the system.

Upon closer inspection, perhaps the claims focus should have been applied more rigorously. Under time constraints of a class project, that included deadlines and requirements not scenario-based design related, the project team perhaps scoped the project too broadly. In the next phase, this thesis uses claims to make progress towards a much more focused contribution.

Chapter 4

Scenario-Based Design of VaDeR

Building upon the requirements from the previous chapter, this design iteration focuses on activity, information, and interaction design. The focus is on using claims to drive the design and capture the design rationale. After the design work is completed, a prototype implements the design for the purposes of testing.

4.1 Activity Design

The first design activity is activity design. Before the look of VaDeR is explored, the tasks it will support must be set. The tasks are closely tied to the problem-based learning approach.

4.1.1 Discussion

In designing activities for VaDeR, there is already a head start due to the task analysis in Section 3.7. But those tasks now need to be supported by activities that can be performed with the interface. From the problem scenarios with UCS, the activities of filtering and accessing detail stand out as being well supported. Getting any sort of overview of the data (such as for organizing artifacts by time) is difficult, so that seems the area where there is the biggest potential for improvement.

Within the context of problem-based learning, these tasks are key for information gathering. The reference must provide an efficient mechanism for exploring and learning about the case study data. UCS's lack of a viable overview is troubling as it forces students to do a bottom-up exploration approach to even begin to piece together the details and guide them into synthesis and making generalizations. Then when those details are found, it is difficult to perform any synthesis due to the interaction overhead and loss of context that pop-up windows create.

So for the first task, using temporal ordering in search, VaDeR will present some sort of overview so that the user will not have to perform any, or at least very little, interactive work to complete this task. To filter artifacts based on attributes, the user will be able to select what data they want to see within the overview. Those attributes will include what type the artifact is and what it contains. The user's final activity is to access the detail for comparison, contained within an artifact, for viewing purposes. As the activities are simple, they are best summarized in one scenario.

4.1.2 Scenario

> Alex is in an HCI class and needs to study the information design for a certain development record because he needs to perform his own information design but is not sure how to do it. He opens up the record in VaDeR. The first thing he sees is an overview of the data already arranged temporally. Knowing that the information design likely occurred somewhere in the middle of the design phase, because the designers used scenario-based design, he focuses in on the middle of the design phase by using the *temporal clues from the overview*. Alex then *uses the filters to remove activity and interaction design from his view*. It is now clear precisely where the information he desires is. So he *accesses the detail from the scenarios and claims* of information design to read what the designers had for

usage scenarios and how that led to claims analysis

4.1.3 Claims

Analyzing the activity scenario reveals three key features. Alex uses a time-based overview to guide his search for data related to information design. In this case, it is not quite enough and he wishes to be sure he is finding the correct items so he uses filters to remove irrelevant information from the display. Once he is certain he knows which artifact he wants, he accesses the detail. This claims analysis is summarized in Table 4.1.

Notice that with UCS, it is not easy to order artifacts by time as noted in the problem claims from Section 3.7.7. This design addresses that issue by making that a primary activity. And the upsides of UCS are retained by still allowing for a quick focusing of a search combined with efficient access to details.

4.2 Information Design

Now that the activities are set, how the interface will look is the next item to be determined. Whatever appears on the display should in some way support the activities while also following general usability principles.

4.2.1 Discussion

Presentation of the visual information is key to VaDeR's success, as it is meant to be an interface for information visualization. Since it has already been identified that time-based artifact ordering is important, the main layout should include some sort of timeline. LifeLines (Section 2.5.2) presents a successful timeline approach, with multiple categories and event

Table 4.1: Claims for activity scenario.

Situation Feature	**Upside and Downsides of Feature**	**UCS relation**
Using temporal overview to aid search	+ can see what data is available with minimal effort	difficult to order artifacts temporally in UCS
	+ easy to use ordering to accomplish any time related task	
	- must be aided by visual design to go along with time data	
	- must recognize what the timeline is	
Filtering artifacts to remove from display	+ able to focus search quickly	similar to hierarchical menu navigation of UCS
	- user may accidentally remove what they are really looking for and become lost	
Access to details from overview	+ efficient access to all available details	UCS uses pop-up windows to show details
	- may not know how to find to specific required data	

representations, so that technique can be adapted for use here. Elastic Windows [20] are applied within the detail area to facilitate efficient comparisons.

Within problem-based learning, this improved information layout will allow the student to efficiently find the data they need and filter out what is irrelevant. If they need to perform a comparison of multiple design phases, the overview and design component menus will make it much more efficient to follow design threads when combined with the comparison frames. And more broadly, the overview will present a more realistic representation of how the design occurred, so the student does not create a false mental model of the data by assuming every design was perfectly sequential through every step of scenario-based design.

Timeline

UCS's help files indicate that each case study contains three main categories of artifacts: analysis, design, and evaluation. Analysis is everything from requirements analysis. Design contains all of the design phase artifacts, such as scenarios, claims, and prototypes. Evaluation holds any of the testing materials used, results, and analysis that leads to design recommendations. So create a timeline divided into those three categories. This use of a timeline based around singular artifacts, separated into categories, is adopted from LifeLines (discussed in Section 2.5.2). On top of that timeline, place the column titles and link them to the help files from UCS that explain what they are. Also place the title of the case study and link that to the introduction page from UCS.

For the timeline artifacts, earliest to latest goes from top to bottom in three vertical rows with space for the artifact representations. Put this timeline across the left side of the interface, in place of where the UCS menus used to be. The space used for the timeline should be smaller the UCS menus, since replacing text with icons allows a more dense use of area.

Each artifact will be represented by an icon placed on the timeline at the appropriate temporal location within the correct categorical row. This is because time is the most important ordering for the tasks, therefore the best visual mapping is position as Cleveland, McGill [13], and Mackinlay [21] showed. But the placement is not within a strict timeline, where a certain length represents a fixed amount of time. Instead place and space the artifacts so that the temporal ordering is preserved and comparison across the columns is possible.

For coloring certain icons, the design phase the artifact is associated with will be used. Using simple and distinctive colors, activity will be red, information green, interaction blue, and documentation yellow. Mapping a property to color is the second best mapping for categorical data [21]. This reduces the number of icons a user will have to recognize, lowering their cognitive load.

Menus and filters

Across the top, the menu titles from UCS are placed in the same order as UCS. They are: Requirements Analysis, Activity Design, Information Design, Interaction Design, Documentation, and Usability Testing. By using the same menu titles, the structure of UCS is maintained. Each title will be a link to the menu contents to be displayed in the detail area. But instead of hiding the submenu contents below another level, everything is shown at once with the submenu titles providing the separation. This reduces the amount of navigation necessary to explore the contents.

For more refined filter control with a design phase, a separate set of menus are provided: Exploration, Scenarios, Claims, and Prototypes. This represents the artifact types found in the design phases. These menus will also allow a user to more efficiently follow the evolution of the design as a whole, instead of going into each design phase menu separately.

Each timeline category will have a unique set of icons. The analysis category has icons for requirements analysis and explorations from the design. Design has icons for scenarios, claims, and prototypes. Evaluation has all of the usability testing artifacts. The only difference in the categorization here is that design exploration goes into the analysis column. This is to prevent the design column from containing an overwhelming number of artifacts, as well as because the exploration acts like the requirements analysis for each design phase.

To support filtering, simple checkboxes are placed underneath each category name. Each checkbox represents some type of data to view in the timeline. If the box is checked, the data will appear. Initially, all of them are checked. The requirements analysis and usability testing categories category have a filter for each type of artifacts they contain. Each design phase has only checkbox, but this is offset by creating a separate grouping of filters for exploration, scenarios, claims, and prototypes. Combined with the design phase filter, these allow the user to control what icons they view in the timeline.

Detail display area

Below the menu titles, and to the right of the timeline, is the area reserved for artifact detail. Whenever the detail is accessed, this area will show the text, pictures, and links associated with that artifact. This prevents details from covering up any portion of the timeline. Two detail areas are placed side-by-side to facilitate comparing two artifacts. This area takes up about the same space as UCS has for detail, since the overview and menu titles do not need a large amount of space.

Those two detail windows are an instance of elastic windows [20]. Compared to pop-up menus, they allow for more efficient comparison between two artifacts. This is because they reduce the amount of work required to manage the windows.

A control is needed for the user to indicate which of the detail frames they want new information to appear. This is accomplished by a small button in the upper-right-hand corner.

An icon that is split into two colors (chicago maroon and burnt orange [30]) will have the active frame noted by a yellow border around the color. To map that color to the actual frame, a strip of that color is placed above the appropriate frame.

Synthesis

Together, all of these features implement Shneiderman's [39] mantra: overview first, zoom and filter, details and demand. The timeline and menus provide an overview of all the available date. Using the filters and menus allows the user to focus their search and find the artifact they are seeking. Then the detail area provides a place to view everything contained in the artifact.

To make the interface fit well into a web browser, all links will use whatever color the browser uses. This includes the different color used when the link has been visited. Doing that aids the user in knowing what they have already visited. UCS, by contrast, uses only black-colored links.

All of these features and their use are demonstrated in a problem scenario.

4.2.2 Scenario

> Alex is in an HCI class and needs to study the information design for a certain development record. He opens up the record in VaDeR. Noticing the vertical *timeline* and a lot of *icons* in it, he quickly locates the design column and looks toward the middle figuring that information design is there. There are green icons there, but he is new to the system and is not quite sure if he found the right items. Seeing the *menu titles* across the top, he sees the same green color below 'Information Design' so he is convinced he was looking at the right icons. To get rid of the other design icons, he uses the *filters* to unselect them. Alex selects the first green icon and the detail appears in the left *detail frame*.
>
> Once he has the information scenarios in the left frame, he uses the *'active frame' button*, resizes the right detail frame to be larger, and opens the information

claims. Interested in reading more scenarios, he uses the *design component menus* and opens 'scenarios.' This brings up a list of all scenarios in the case study and he proceeds to read all of them.

4.2.3 Claims

Along with the scenario, the more detailed discussion provides many features for analysis. There is a timeline organized into three categories with artifacts placed within each. A list of menus go across the top with filters are placed below them and a detail area is reserved to the right of the timeline. Claims analysis for these features is summarized in Table 4.2.

Referring back to the claims with UCS (Section 3.7.7 shows that this design keeps the upsides of organizing the data into categories and using filters to help aid a search in place of hierarchical menus. The downside of having multiple pop-up windows is addressed by having an area reserved for two detail views. And as in activity design, the use of time as a primary visual mapping mitigates the downside of the difficulty in using time data in UCS.

4.3 Interaction Design

The activities and visual look are now determined. Next is to design how a user will interact with what they see to achieve the activities.

4.3.1 Discussion

Finally, the interaction with VaDeR must be defined. We have the supported activities and what VaDeR looks like, so interaction scenarios will show how the interface is actually used. Assume VaDeR is being used on a desktop computer with a keyboard and mouse. Interaction is being kept intentionally simple as it is the information design that is intended

Table 4.2: Claims for information scenario.

Situation Feature	**Upside and Downsides of Feature**	**UCS relation**
Timeline of artifacts on left sidebar	+ divided into the three components of UCS	shows all UCS data in temporal order
	- but moves design exploration into analysis section	
	+ shows all available artifacts	
	- is based on temporal ordering and not a strict timeline scale	
Menu names across the top	+ access to existing UCS categories	maintains UCS organization
	+ indicates type of information present	
	- must open menu to see submenu titles	
	- text smaller than UCS for space purposes	
2 elastic detail windows	+ dedicated area does not cover up navigation	replaces pop-up windows
	+ used for seeing menu or artifact	
	- but dual use may confuse users	
	+ efficiently compare 2 menus or artifacts	
	- but can not view > 2 menus or artifacts at once	
	+ user can control target window	
	- but requires an on-screen control	
Filters for each data type	+ removes undesired icons from the timeline	same effect as menu navigation
	- must learn what each icon represents	
Icons used in timeline	+ simple, consistent representation	visual representation of artifact
	+ colors represent associated design phase	
	- does not have ideal graphic design	
	- learning curve if user has never seen these icons	
Design component menus	+ efficient following of design progression	gathers all design artifacts together
	- breaks from UCS organization	

to be the biggest contribution of VaDeR. Therefore this design is relatively simple compared to information design.

For problem-based learning, this simplistic interaction ensures the students are focused on the data and not the overhead of interaction by having to move, resize, and close pop-up windows. Reducing the number of clicks the student performs will allow them to maintain their context more easily, thus allowing for the information connections to be explored more fully. Reducing the physical workload of the student should enable their brains to focus on what is important.

Each artifact icon on the timeline will be clickable. That will bring up the detail onto the display. Bringing up another artifact's detail is done by clicking the associated icon. Filters are checkboxes, so the user simply has to click within the checkbox to turn on or off the data type that filter represents. Within the detail area there may be more links that the user can click on. Anything clicked there will cause the information to appear in the active frame.

The most complicated interaction is with the 'active frame' button to control where new information appears when the menus or artifacts are clicked. A user must press that button whenever they wish to switch the active frame.

4.3.2 Scenario

> Alex is in an HCI class and needs to study the information design for a certain development record. He opens up the record in VaDeR. Wanting to find only information design data, he uses the filters and unchecks, by *clicking with his mouse*, everything expect those for information design. Now the only thing icons left on the timeline are for information design. Alex *clicks the first scenario icon* and the details appear in the left detail frame. Then he *presses the 'active frame' button*. In the scenario detail, *he sees a feature link and clicks it*. This brings up the claim about that feature in the right detail frame. Alex quickly and easily moves through all of the information design data by simply using his mouse to click on the icons.

4.3.3 Claims

A simplistic interaction design still leads to some tradeoffs. Everything is performed by clicking and there are no concurrent of actions. This keeps the interaction as simple as UCS and introduces no meaningful new complications. The claims analysis for this interaction technique is summarized in Table 4.3.

The main issue here is to keep the upsides of UCS having a simple interaction method (Section 3.7.7). By keeping the interaction to clicking objects on the screen, there are no new downsides introduced by this design.

4.4 Prototype

With the design finished, a prototype must be created for evaluation purposes. Using frames within a webpage allow the prototype have a relatively high fidelity compared to what a final system would be. A single case study was taken from UCS and converted to how it would act within VaDeR. In this case, it is the Virtual Science Fair case study.

When the system (Figure 4.1 first starts up, the user gets the timeline overview of artifacts with the introduction provided in the detail area. In the upper left is the title of this particular development record or case study. Below that are the titles of each column in the timeline. Each column contains icons 4.3 for each artifact, placed relative on the timeline according to their date. Because there are many artifacts with identical or similar dates, the icons are not always placed exactly where they should be, but they are in order. When an artifact is related directly to a specific design phase, the background is colored to match the color assigned to that design phase. Some examples are provided in Figure 4.4.

All of the category names and design phase titles are links. Clicking on them shows an explanation of what they are in the detail area. For example, clicking 'Activity Design'

Table 4.3: Claims for interaction scenario.

Situation Feature	Upside and Downsides of Feature	UCS relation
Click checkbox to filter data	+ checkboxes are a familiar metaphor	same as clicking menu title in UCS
	- does not allow for a fine-grained filtering	
Click icon to get detail	+ simple interaction method	same as clicking text link in UCS
	+ familiar metaphor that clicking an icon opens up the content	
	- action not reversible, can only click another icon to remove current detail	
Use 'active frame' button	+ simple interaction method	replaces using pop-up windows
	+ allows full user control as to where new information appears	
	- requires a lot of clicking if user needs to frequently change the active frame	
Clicking links within data	+ familiar hyperlink method	same as UCS, just does not use pop-up windows
	- user must change active frame to change target of link	

provides a definition. The title of the development record is also a link and it provides an introduction that is shown by default when VaDeR is first opened.

Below the menu titles are the filters as described in Section 4.2. Notice the design categories have their associated color next to their name to reinforce the mapping. Every filter is on by default. For this particular case study, the filter for evaluation analysis is grayed out because no such artifacts exist.

By clicking the checkboxes for activity and documentation design along with the claim filter, all claims and all information related to design exploration and prototypes are removed from the overview. This is shown in Figure 4.6.

Below the menu titles the detail area is split into two. Whenever an artifact icon is clicked, the associated detail will appear in one of these two areas, depending on which is active. In Figure 4.6, the activity scenarios are shown on the left while the activity claims are shown on the right. This facilitates viewing the context of design features while also examining the effects those features might have. Any other combination of two artifacts is possible.

Because the prototype uses frames within a webpage, users can adjust the size of any section to their liking. This allows them to hide category rows or remove the filters, among similar possibilities. A finished system would provide such flexibility in the interface, so this prototype emulates such a feature.

Now that prototype is finished, it is ready for evaluation, as discussed in Chapter 5. But in deciding what to test, the claims relationships must be examined to see if they suggest the important claims.

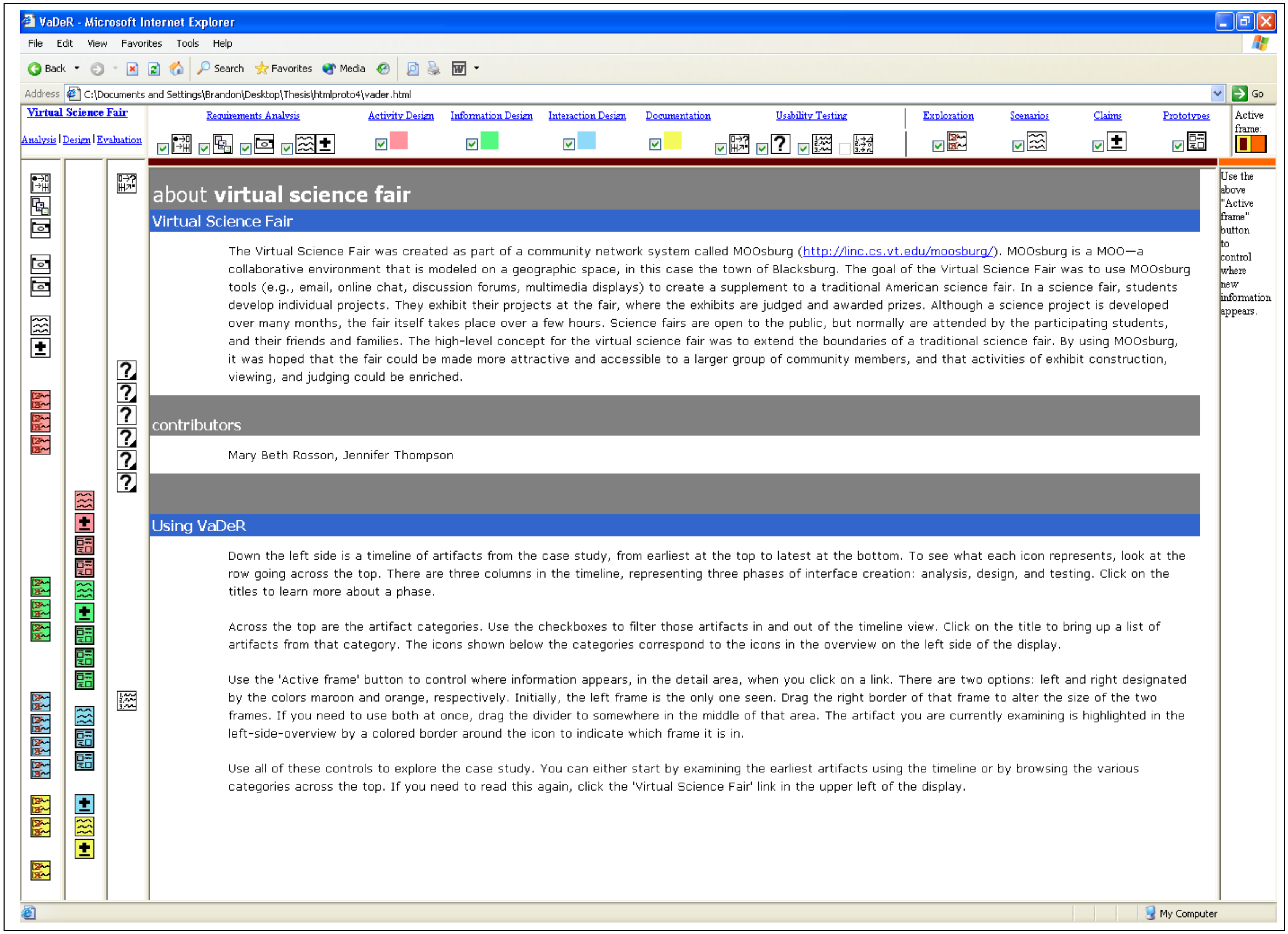


Figure 4.1: VaDeR displaying the Virtual Science Fair case study.

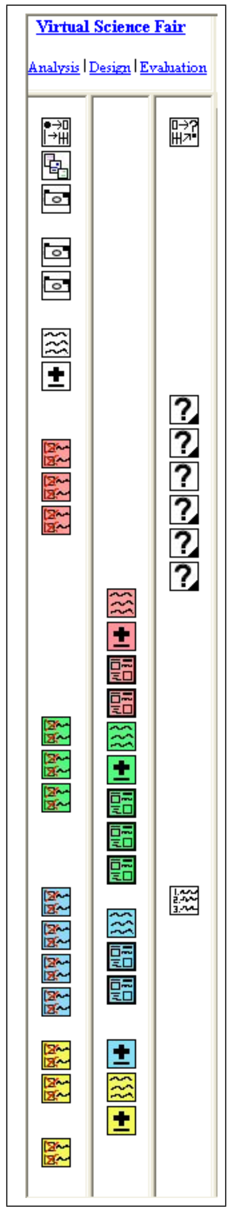
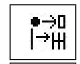
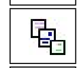
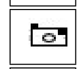
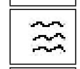
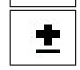
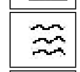
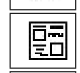
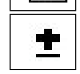
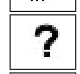
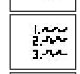
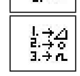


Figure 4.2: VaDeR timeline overview.

Figure 4.3: Icons in VaDeR.

Category	**Sub-category**	**Icon**	
Requirements Analysis	Planning		
	Method & Materials		
	Information Gathering		
	Synthesis		- Scenario
			- Claim
Design Phases	Exploration		
	Prototype		
	Envisionment		- Scenario
			- Prototype
	Rationale		- Claim
Usability Testing	Planning		
	Methods & Materials		
	Information Gathering		
	Synthesis		

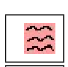
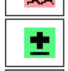
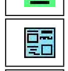
Icon	**Artifact type**
	Activity scenarios
	Information claims
	Interaction prototype
	Documentation design exploration

Figure 4.4: Examples of colored icons for design phases.

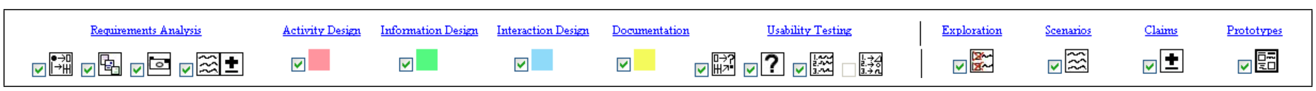


Figure 4.5: VaDeR data filters.

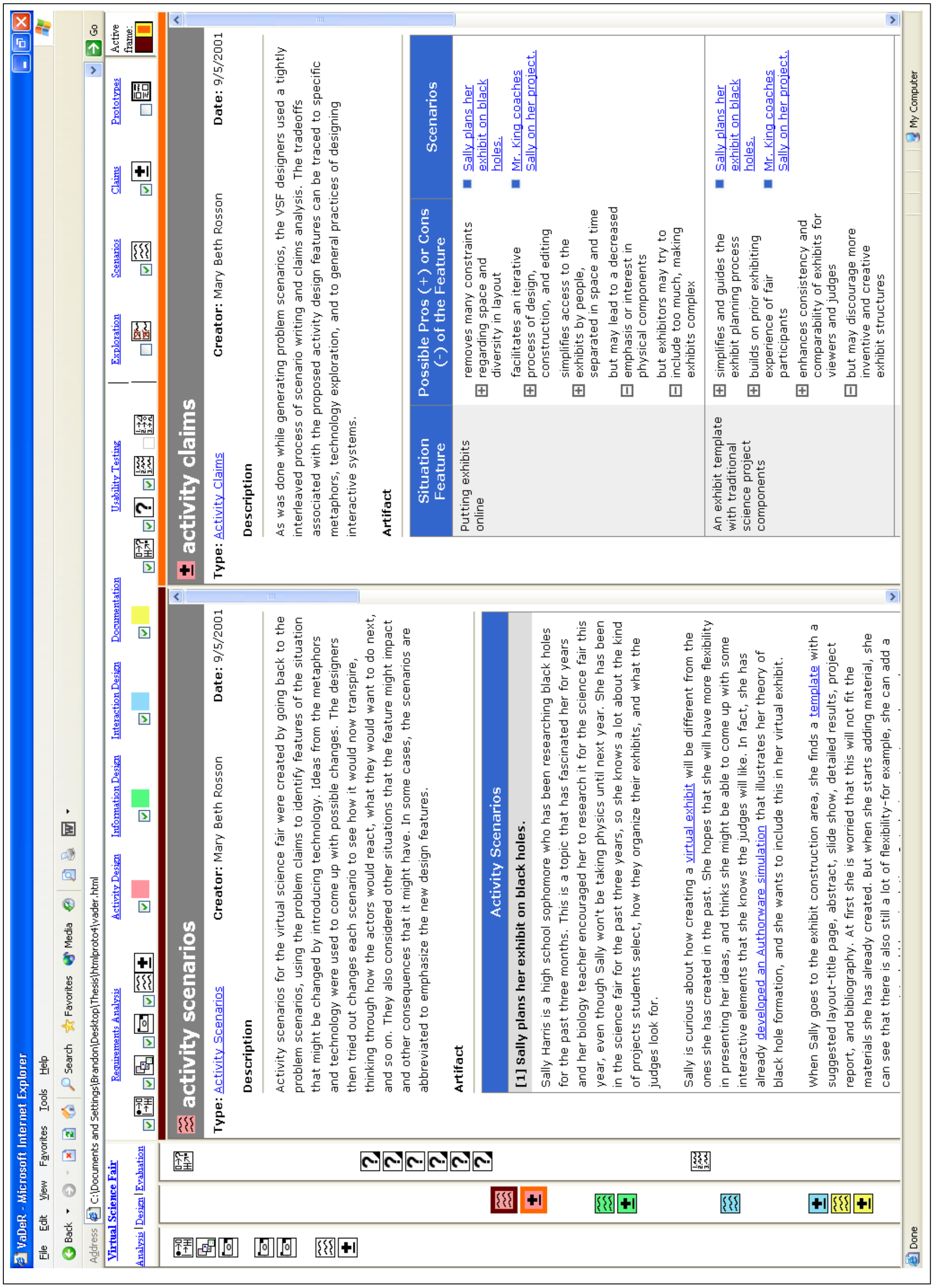


Figure 4.6: VaDeR with some data filtered from view and two artifacts opened.

4.5 Design scalability

The current prototype of VaDeR shows only one case study from UCS. Since most of the case studies are of similar size, scalability is not an immediate issue. But it needs to be addressed if VaDeR is ever to be used for larger case studies or expanded outside of the education domain.

There are two main scalability issues. One is the use of only two elastic windows. Users may desire more windows, even for small case studies, and this can be addressed by investigating the usability of implementing elastic windows as Kandogan and Shneiderman [20] did. There, the user has full control over how many windows they want, what their size is, and the use of hierarchical windows.

But the main scalability issue is with the timeline and the icons. Presently, VaDeR can handle a case study with about 100 artifacts but that also depends on when those artifacts are dated since the timeline concept must be maintained. One simple option is to reduce the size of the icons, but that can cause recognition problems. Therefore other techniques must be integrated if more artifacts need to be handled.

With smaller icons, a zooming [1] technique would allow the user to magnify the sections of the timeline they are interested in, bringing the icons to a recognizable size. Adding to that, a focus + context [1] approach would ensure the icons around the edge of the magnification would not vanish from sight. As another option, icons could be aggregated to a smaller set of icons, such as representing all activity design artifacts as one icon and then revealing the entire detail when that section is zoomed in.

If these scalability issues are addressed, VaDeR will be able to handle much larger case studies. Perhaps this will allow students to learn from much involved design examples as opposed to the smaller cases used presently.

4.6 Reflecting on Relationships within the Claims-centric Scenario-Based Design

Tracking the entire design process, from requirements analysis to final design, should reveal the important contributions of the design and lead to what needs to be evaluated to determine if the design goals were met. In this project, the tracking method used is the claims relationships framework [46]. By formalizing the relationship between claims, a structured view of the design process emerges and hopefully areas ripe for testing and new claims that can be reused are identified.

A design overview is presented in Figure 4.7. This is not intended to be a complete list of claims, but mostly the important ones as identified through the design. To contrast, unimportant claims deal with minor interface issues that should not have a noticeable effect when evaluating if the interface supports the tasks it was designed for. They are not completely ignored during the design or evaluation, they are just not the focus. It is of little interest to retest claims that have been verified thoroughly in other work, such as what mapping is best for qualitative data in a visual representation.

The design starts with the analysis 3.5 of problem-based learning with UCS, including the CS 3724 homeworks as real examples of the connection. From there the breakdowns emerge with three key tasks identified in Section 3.7. Those tasks flow directly into activity design (Section 4.1). Analyzing those scenarios results in problem claims(Section 3.7.7). Those claims are mostly addressed in the information design, which then has interacting claims resulting in the important claims to test with claim X, Y, and Z, as they mitigate the problem claims. A legend is provided in the upper-right to follow the claims relationships. The letters are references used in this document, especially with the results in Section 5.6.

Throughout the design phases, new claims were developed to address the problem claims. Within the claims relationship framework, these are postulating claims. Each problem claim is addressed somewhere in the design. Using the postulating relationship showed that this was the case. In reverse direction, the problem claims predicate the design claims.

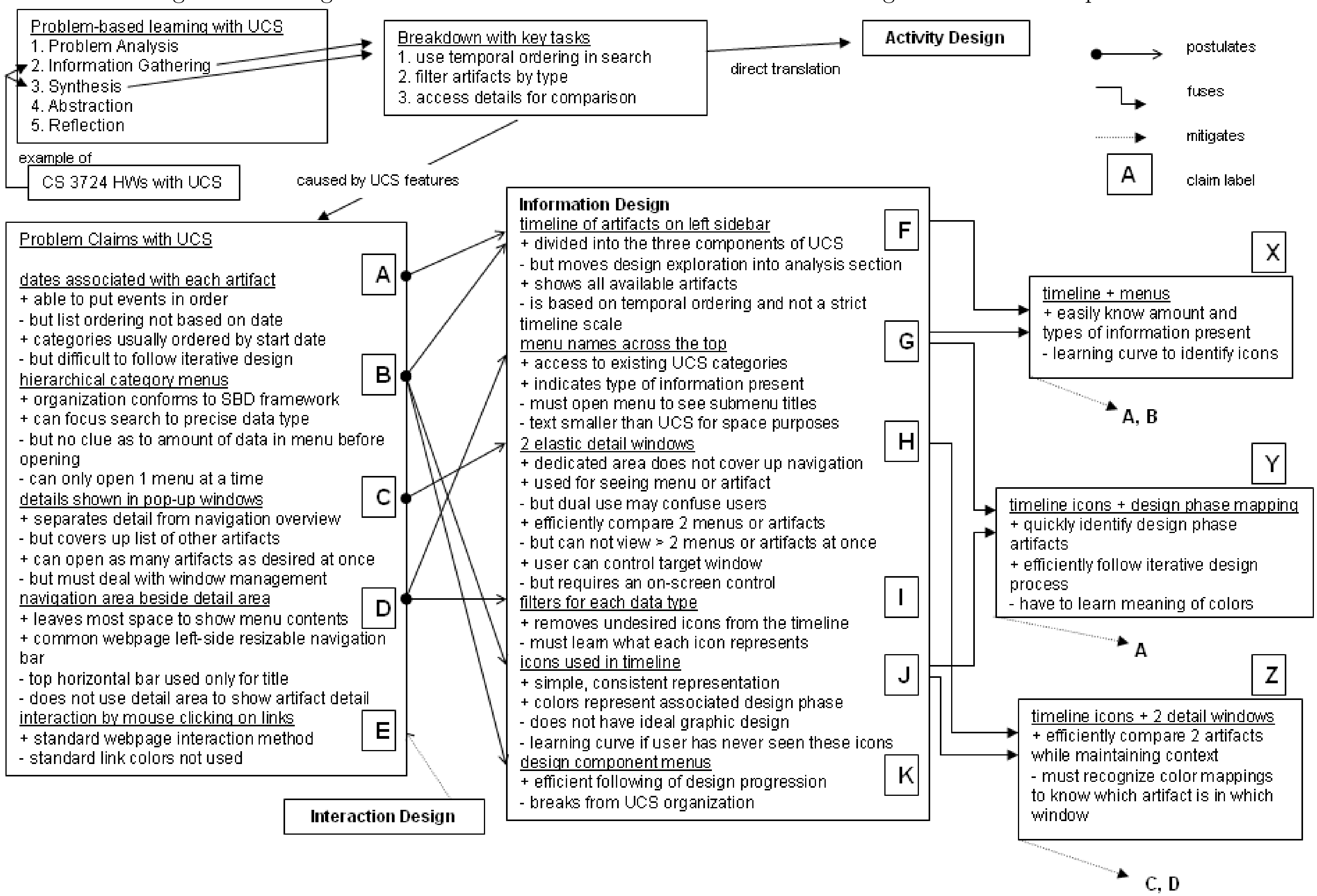
So after it was ensured that all of the problem claims were addressed in the design, the design claims were analyzed for important interactions. This analysis focused on the information design, as this represented the key departure from UCS. There are three key combinations, all involving the timeline since that is the most radical change from UCS. The timeline combined with the menus (claim X), design phase coloring (claim Y), and detail area (claim Z) to produce three new claims. Each new claim inherits the upsides and downsides, but the interaction leads to new upsides and downsides.

This is where the fusing relationship is used. The idea of that relationship encouraged the analysis that led those new claims. So tracing through the claims using the relationships ends with these three new claims. In turn, those claims trace back to the problem claims and mitigate the problem claims. Since these three claims are fused from the design claims, and they mitigate the problem claims, then it follows that testing based on these new claims would produce the most insightful results.

Using these claim relationships allowed us to see the progression of claims from the problem claims all the way to evaluation claims. But applying these relationships took time and effort because the definition of a claim does not immediately suggest these relationships should exist at all. It may even be that knowing these relationships influenced the design itself, for allowing isolated claims may have resulted in features with no connection the problem claims or evaluation. Just having the thought that some claims might fuse together may have resulted in the intentional creation of low-level claims, knowing we could always combine them later. And there is no way to be sure the fused claims are soundly based on the principles of the claims that were fused.

But the relationships did show a seemingly natural progression from problem claims to evaluation, suggesting that the claims requiring testing were indeed the important contributions of the interface. This project will only be able to examine one such instance of that progression. Other efforts are needed to see if using these relationships forces a well-connected design rationale collection and results in appropriate evaluation claims.

Figure 4.7: Design overview that shows the evolution of claims through their relationships.



Chapter 5

Exploring Tasks and Claims

Designed to support problem-based learning, there is currently only theoretical support that VaDeR succeeds in its goals. So we go back to scenario-based design for guidance in developing some kind of evaluation. The claims, through their evolution, should be able to point to what needs to be evaluated in order to learn if the design truly addressed the problem claims.

From the design map in Section 4.6, the important contributions of VaDeR over UCS were wrapped in three key claims. First, use of the timeline plus the menus should enable full and efficient navigation through a case study. Second, having the design phases colored in the timeline should aid in following an iterative design process. Third, having the timeline with highlighted icons in tandem with using the detail area should enable the user to compare artifacts while maintaining context.

But these are new claims that have no tested validity behind them. Claims need to be evaluated, and only then can they gain some level of validity. As they now exist, these claims are not believable enough to be useful for reuse. No one will trust them to the extent that they are willing to use them in the process of their own design. Unlike the previous evaluation, which broadly considered the usability of the prototype, this evaluation examines the key claims.

So complete a scenario-based design process that took a claims-centric approach, we must capture within the claim any rationale for its believability. That rationale can come from any evaluations, empirical or otherwise. This would satisfy the claim model of Payne et al. [27] that includes the theoretical background for each claim. If more thorough and formal evaluations validate the claims within this project, then the captured design knowledge can become a trustworthy background to the claims. With this limited context and low number of participants, it will not be possible to state definitively that they are true but they will be strengthened.

This chapter considers this process of evaluating claims for validity and future reuse, and reflects on its strengths and weaknesses.

5.1 Claims under Review

We have, through use of the design map (Section 4.6), identified the claims that need to be investigated. They involve the interaction of features as in the timeline plus the menus, timeline plus design phase colors, and timeline with highlighted icons plus the detail area.

Tracing those claims back demonstrates how they attempt to address the key tasks that were identified. Use of the timeline plus the menus and icons (to help recognize what is in the timeline) is designed to aid any searching that uses temporal information. The menus and their filters, along with the design phase coloring, help the user focus their search by filtering even if they only perform a mental filtering by ignoring the icons they have no interest in. And the detail area allows an efficient comparison of artifact details. So it appears these claims are appropriate for a relevant study.

Now the purpose of testing those claims must be determined. Since the design claims from VaDeR evolved from UCS problem claims, a comparison between those two interfaces will provide valuable results. This will enable the evaluation to see if VaDeR truly accomplishes its goals. If the study gives any strength to the claims then it becomes likely that VaDeR better supports the tasks than UCS, and should be developed further.

5.2 Approach

Since scenarios of usage were created as part of the design, the testing procedure would benefit from being based on scenarios that the evaluator acts out. A cognitive walkthrough [48, 18] evaluation approach is therefore appropriate. In this technique, the evaluator is set up with a task to accomplish, then tries to accomplish that task with the interface. Along the way, they note discrepancies between what the interface provides and user goals, knowledge, and expectations. The evaluator can also note usability problems.

In this particular test, the cognitive walkthrough will be performed by recruited participants. They will be set up with a scenario and told to attempt to accomplish a goal with the interface while talking about what they are doing and noting what interface features help or hinder their task. This is combined with the principle evaluator interrupting with questions to ensure they understand the thought process the participant is undergoing.

As an offshoot of that walkthrough, the participant is interviewed to get their opinions on various interface features or the interface's ability to support a task. Those questions arise from the claims and will be the same for both interfaces. Additional questions will be asked specifically about VaDeR to ensure specific feedback is collected about some of the more minor features. This is done to see if the participant thinks a feature aids in accomplishing their tasks or is a detriment that should be redesigned.

To finish the study, the participant will be asked about how they would compare both interfaces in their ability to accomplish a task. They not only are able to express preferences, but can also note what features led them to their decision. All of these questions are also related to the claims developed during the design of VaDeR. If users consistently note a preference for UCS for the tasks, then many of VaDeR's claims will need to undergo a serious rethinking.

Using an interview format will also allow for new questions to be asked when the need arises. Mostly this is for situations when clarifications are needed about what the participant notes about interface features or improvements that they suggest. With this ability, the evaluator can be satisfied that they understand the participant's answers and are not left with sparse feedback that gives little or confusing insight.

5.3 Scenarios

From the three key claims identified earlier, there are two tasks that encompass the important perceived upsides of VaDeR. The first is comparing multiple artifacts and the second is following an iterative design process. Comparison entails finding the appropriate artifacts and using the elastic windows in the detail area in tandem with switching the active frame. Following the iterative design process entails discovering if one exists by using the timeline and learning what the icons represent. So performing these tasks guides the participant to using these main VaDeR features, allowing them to evaluate those features in the context of realistic usage scenarios.

Participants also have to explore the UCS features in those scenarios. With this interface, the comparison scenario forces the participant to find desired artifacts, open the pop-up windows, and go back and forth between the windows. Most likely, there will be some window management involved. To follow an iterative design, the user must explore the menus and look for some indication of design changes. In the process, they will possibly use the artifact dates to aid their exploration.

Both scenarios are now set up by analyzing the involved goals and design issues that are being explored. This is all shown in Figure 5.1 and Figure 5.4.

5.4 Guide to Informal Interviews

Before the interfaces are introduced, a quick pre-questionnaire is used to get an estimate of the participant's knowledge in related areas. Selection from not at all, somewhat, very, each participant rates their familiarity with scenario-based design, UCS, and information visualization. This is done to see if there is any connection between their comments during the walkthrough or usability concerns with their knowledge of those areas. For example, people very familiar with scenario-based design, UCS, and information visualization might present a large preference for VaDeR and people who know nothing about those topics might prefer UCS for the tasks and its clean interface. But there is no way to infer relationships like those without collecting this data.

All of the interview questions asked have some relation to the claims or general usability concerns. These questions take place after the participant has had the chance to attempt tasks using the interface, which focus on the key claims identified in Section 4.6. If the participant already gave a direct response to the questions during the walkthrough, then the question is discarded so that the participant does not have to keep giving the same answer repeatedly. Additional clarifications can still always be asked for, though. The entire guide to these questions and the claims they are investigating are listed in Tables 5.1, 5.2, and 5.3.

Start	Alex only has cursory knowledge of how scenarios and claims are used in design, and how they lead to prototypes.
High-level task	Compare and contrast the design phases of information vs. interaction from an existing case study.
Lower goals	•Examine the connection between the scenario and claims. •Discover how the design claims addressed the problem claims. •See how the prototypes were used to envision the claims. •Learn how the scenario and claims differ between the two design phases.
Example problem scenario	Figure 5.2
Result	Alex has a clearer understanding of how to proceed with his own information design and have it lead into interaction design. ·knows what his scenarios and claims should focus on in each design phase ·knows how the prototypes should address the claims ·knowledge of useful methods for creating prototypes in both design phases
UCS issues	•Must go through menus one at a time. •Each new artifact detail creates a new pop-up menu; must deal with window management and resizing. •Have to open an artifact from one design phase, go back to other phase and find other artifact for comparison. •When dealing with multiple windows, easy to get lost as to where each one fits within the overall case study.
VaDeR design	•Places submenus on one page; can go through 2 menus at once. •Puts details in 2 elastic windows. •Provides constant context through timeline icons and highlighting.
VaDeR scenario	Figure 5.3

Figure 5.1: Comparison testing scenario.

Alex opens a case study in UCS. He clicks on information design menu, then clicks on envisionment. Finding the information scenarios, he clicks on link then reads the scenarios in pop-up window then minimizes that window and goes back to UCS and clicks on interaction design menu, then clicks on envisionment. Alex opens those scenarios in a pop-up window, minimizes all the windows, brings back up both scenario windows, moves them next to each other, and makes them bigger to see more text. Next he goes back and forth between the windows, noting the different focus of the scenarios. Now Alex has much better idea of how to write his own scenarios to focus on information or interaction, depending on the design phase he is in. Moving on, he closes both windows, and brings back up the main UCS window.

Figure 5.2: Example problem scenario for comparison task.

Alex opens a case study in VaDeR. He clicks on information design link, the clicks on the scenario artifact listed in the detail. Next he presses the 'active frame' button then slides window border to middle of detail area. Clicking on the interaction design link, he then clicks on the scenario artifact listed in the detail. Now he goes back and forth between the windows, noting the different focus of the scenarios. Moving on, Alex clicks on other menu links or artifact icons while only dealing with minimal frame management.

Figure 5.3: Example VaDeR scenario for comparison task.

Feature	**Related claims from Figure 4.7**
Ability to search by date	A
Useful to open 2 menus at once	B
Use of flat menus (VaDeR only)	G,K
Pop-up windows expected, annoying (UCS only)	C
Window management (UCS = pop-up, VaDeR = frames)	C and H
Empty menu representations	B and K
Ability to place artifact within context	A,B,C,D and X,Z
Ability to follow iterative process	D and X,Y
Other interaction or design issues	all

Table 5.1: Guide to questions for both UCS and VaDeR.

Start	Derek has limited understanding of how an iterative design process occurs.
High-level task	Examine a case study and learn from an example how to perform iterative design.
Lower goals	•Determine the first steps taken in a design •See how it progresses into scenarios and claims. •Examine how those scenarios/claims lead to a prototype. •Examine evaluations done on that prototype to see how the claims were used. •Look for revisions to the scenarios/claims/prototypes after the test results and testing done on those.
Example problem scenario	Figure 5.5
Result	Derek has a clearer understanding of how to perform an iterative design process. ·understands how the iterative design progressed ·saw chain of scenarios → claims → prototype → evaluation → analysis → altered scenarios/claims/prototypes → new test → ... ·knows how to perform his own design method to ensure he has an iterative process
UCS issues	•Artifacts not always ordered by date. •Categorization lumps all scenarios/claims/prototypes/evaluations into one place regardless of where they came in the design process. •Must pay close attention to dates and go back and forth between multiple menus to follow design in order.
VaDeR design	•Orders artifacts by time in a timeline going from top-to-bottom within a column. •Maps icon color to associated design phase. •Analysis → Design → Evaluation going from left-to-right with columns. •Can generally follow iterative design by 'reading' artifacts left-to-right, top-to-bottom.
VaDeR scenario	Figure 5.6

Figure 5.4: Iterative process testing scenario.

Derek opens case study in UCS. Clicks on the information design menu, then clicks on exploration. He clicks on artifacts there, reading each in pop-up window and closing the window. Then he clicks on envisionment, and examines those artifacts. Then he does the same for rationale. Now he opens the usability testing menu, and goes through the submenus there looking for anything related to information design. Eventually he finds an evaluation form used for the information prototype. Then he looks in the info gathering menu and finds the results amongst other unrelated results and similarly finds the analysis in the synthesis menu. Noting the dates for the analysis, he goes back to information design section, and looks for the claims. This time he knows which claims came before the evaluation and which came after so he sees how the design was altered based on the analysis. He also looks for the prototype that came after the evaluation analysis by looking at the associated date. Going back to the usability testing section, he looks at the dates and finds an evaluation made on that second prototype. He is able to better understand how claims lead to evaluations to analysis to design revisions and so on.

Figure 5.5: Example problem scenario for iterative process task.

Derek opens case study in VaDeR. Noticing that information design is noted by a light green color, he focuses on artifacts of that color. Using the timeline, he finds the first exploration and reads those artifacts by clicking on the icon. Then he reads the scenarios, claims, and prototypes by finding them below the exploration and in the next column. Looking in the last column, he reads about the evaluation and analysis. Going down the timeline, and left-to-right, Derek quickly finds the scenarios/claims/prototypes created after that first test's analysis. Continuing this way, Derek efficiently goes through the iterative process that occurred during information design.

Figure 5.6: Example VaDeR scenario for iterative process task.

Feature	**Related claims from Figure 4.7**
Icon design learnability	J
Use of color to designate design phases	J
Use of color to represent active frame and highlight icons	H, J
Filters for removing icons from timeline	I

Table 5.2: Guide to additional VaDeR usability questions.

5.5 Experimental Design

Since the key claims being tested state that VaDeR better supports the tasks than UCS, the goal is to show some difference between the interfaces. So the null hypothesis is that both interfaces are identical with their task support and that users will have no preference. The first part of the evaluation has users perform the same tasks and they are asked the same questions for both interfaces. This means the independent variable is the interface used and the dependent variable is the responses to the questions and their observations during their walkthroughs.

After that, the second part of the evaluation gets some feedback about the usability of VaDeR for the purpose of directing further design. Then the user is asked to compare the ability of both interfaces in various facets and also state any preferences that result.

Participants will all be graduate students who have at least some knowledge of scenario-based design and human-computer interaction in general. Using them reduces the amount of instruction needed to fully grasp what UCS and VaDeR are for. Their human-computer interaction knowledge will aid in giving valuable usability feedback.

Testing will take place in the same lab with the same physical setup for each participant. Each test must be done individually due to the walkthrough and interview format. One hour is the expected amount of time each participant will take to finish the entire process.

Feature	**Related claims from Figure 4.7**
Ability to obtain overview of case study	X,Y
Ability to use time in aiding search	X
Navigation to find particular artifact	X,Y
Maintaining context during exploration	X,Y,Z
Reading artifact details	Z
Comparing artifacts	Z
Following an iterative design process	X,Y

Table 5.3: Guide to comparison questions between UCS and VaDeR.

5.6 Results

Across the board, all seven participants provided valuable insight into the claim validity and usability of both interfaces. Everyone had similar overall opinions with most variation occurring in their design improvement suggestions. Based on these results, VaDeR is a success even when considering it only exists in prototype form. Suggestions for future versions also point to areas in which VaDeR can improve, but the problems with the interface appeared to have not provided a major impedance for the tasks the participant performed.

Two of the participants were very familiar with scenario-based design, UCS, and information visualization. Just like the other participants, these experts were enthusiastic about VaDeR's task support, usability as a whole, and potential for future development. This is promising as the people whose criticism would carry the most weight were still interested in VaDeR.

It is also encouraging that every participant had positive reactions to VaDeR, especially when considering the tasks it was designed to support. After evaluating the suggestions for improvement and going through another design iteration, VaDeR should be ready to test in a much more formal evaluation with more participants. None of the results so far indicate any fundamental flaws exist, so the basic design shows promise.

Each area of the study is now broken down for further analysis. Claim references refer to the design map in Figure 4.7.

5.6.1 Comparison task

With UCS, participants typically followed the same process for this task. They browsed through the menus looking for artifacts to compare, usually scenarios and claims, and brought up the associated pop-up windows. Then the windows were resized and moved around to allow for comparison between the details. Most participants expressed dissatisfaction with having to deal with this window management, with a few being extremely against pop-up windows. This strengthens problem claim C (pop-up windows), so this appears to be a true deficiency in UCS. For changing UCS, some suggested having the option of using pop-up windows or opening the detail in the same area.

With VaDeR, participants were able to find the artifacts they desired more efficiently. The ability to directly access artifacts by use of the icons was found very useful. Using the elastic windows and changing the active frame, they were also able to compare artifacts efficiently with less interaction than UCS and simple window management. Along with the timeline icons, the participants were also able to maintain the context of what they were comparing. Claim Z (artifact comparison), thus, has some validity. The upside that elastic windows are more efficient than pop-up windows for comparison was borrowed from Kandogan and Shneiderman [20]. Results here further strengthen their claim.

5.6.2 Iterative process task

Participants, with UCS, found it very difficult to discover if an iterative process occurred and to follow it if it did. The lack of a meaningful overview and no indication of versions in the details was the first difficulty encountered. Then using the dates as a guide was frustrating because they had to go into each menu and read the dates of only the few artifacts present. This strengthens the argument that claim A (dates) and B (menus) had valid downsides. One demand was to have some sort of introduction or overview that would indicate an iterative design, as the menus implied there was always a straightforward process.

But with VaDeR, they found the timeline extremely helpful in accomplishing this task. Because of the colors mapped to a design phase, participants were looking for colors to be mixed further down the timeline, which would indicate revisions to the scenarios, claims, and prototypes. They also expected to possibly find more requirements analysis or testing done sometime after the initial design artifacts. So just by looking at the timeline, the participants would be able to find an iterative design and follow the process if it existed. Claim X (timeline + menus) and Y (icons + colors) thus gain some validity.

5.6.3 Comparing UCS and VaDeR

In comparing UCS and VaDeR with respect to the main tasks and many secondary tasks, responses overwhelmingly favored VaDeR. Participants thought the increased efficiency of VaDeR combined with a visual overview made it much more usable. One quote that seemed to sum up many of the thoughts was that "UCS is less intimidating, but VaDeR is more powerful."

A major improvement noted was the elimination of pop-up windows in favor of frames. Still, many noted that an option to create more frames would be useful. But it appeared that the two given by default were sufficient for many of the tasks, so it was not a major issue. It is something to be explored in future versions, though.

It is encouraging that although many participants found the VaDeR interface overwhelming at first, they all said they were able to quickly learn how to use it. Considering there are still multiple improvements that can be made for help purposes, it is clear that even a little exploration results in a lot of learning. Watching the participant learn resulted in remarks like "wow" and "cool" as the participant was exploring VaDeR.

5.6.4 VaDeR usability

Of course, there were still plenty of suggestions to improve the usability of VaDeR. Mainly the option to create more frames was noted. Then there should be more visual cues to indicate which frame is active. Being able to click on the color slits next to the frame was one suggestion that could easily be implemented, but then some affordance must be explored to indicate that is possible. Also, the use of a hotkey to change the active frame would reduce the amount of mouse movement needed.

The icons below the menu titles were also an area where changes were suggested. They could either be linked to the checkbox to make them easier to click or have them linked to their appropriate submenu so that users could just view one portion of a category, as in UCS. Tooltips for the icons also would help in more quickly learning their meaning.

Other than those issues, many of the suggestions were minor improvements that new versions could incorporate. This includes using better affordances and clearer icons, which could be accomplished by consulting designers experienced with graphic art in user interfaces.

From analyzing all of the results, it is clear that VaDeR is a success so far. It met its goals of supporting the key tasks and improved on UCS in the process. And with the subjective opinions, participants expressed a uniform preference for VaDeR. One person noted that it required "less reading, but you gain more insight" than UCS. A selection of other quotes includes "really well designed," "it's a winner," "seems pretty good," "like this a lot", "much better," "this is nice," and the designer's personal favorite, "much cooler."

5.6.5 Reflecting on claims-centric evaluation

By promoting claims to a level of prominence not visible in many scenario-based design approaches [32, 7, 6, 50], we showed how evaluation plans can emerge naturally from earlier design stages in the scenario-based design process. While our evaluation was rather simple, consisting of walkthroughs and interviews with expert graduate students, it serves as a demonstration as to how more complex evaluations would be developed.

Scenarios were able to capture realistic usage and the claims identified the effects of key features. While, of course, there were variations in the way that participants used the interfaces, the scenarios described their activities. And many of the hypothesized upsides and downsides of claims gained some validity through testing based on those scenarios. So the developed scenarios and claims were realistic assumptions about how people use the interfaces and their reactions to them. There were no big surprises that would indicate the scenarios and claims had no connection to reality, so scenario-based design succeeded in the attempt to be user-focused.

Applying the claim relationships highlighted the important claims for testing, and those supposed key contributions of VaDeR were noted by the participants. When comparing UCS to VaDeR, participants pointed out those areas in which VaDeR improved upon UCS and those corresponded to the key claims used in testing. This is true even when they were able to freely explore the interfaces with no prodding from the principle investigator. Features were merely pointed out, and the participants came to their own conclusions and believed that VaDeR was better suited to perform the tasks.

But there were downsides to this approach. First, how can we be sure we evaluated the correct claims in an appropriate manner? There could be other interactions not considered that were not revealed during the testing, resulting in new claims that introduce serious downsides. But without testing, those downsides might not be revealed until the interface is unable to be altered to address those downsides without causing a major shift in the overall design. And if the most relevant claims were tested, how can we be sure they were tested in a manner that would increase the validity of their upsides and downsides? A different type of evaluation, based on the same claims, might create different results. Perhaps there needs to be some introducing claim testing into any sort of evaluation. This would mean there would be techniques whereby you take a claim, examine its upsides and downsides, choose an evaluation method, and adapt your approach to ensure the claims are correctly evaluated within that method.

Next, there is no existing tool support to facilitate claim testing while incorporating reuse. LINK-UP [12] provides some help in evaluation, but is not able to automatically use the related claims library to tell the designer that they do not need to evaluate a claim, as it already has been validated. As an extension to that idea, it could be able to inform the designer what claims have low validity and are therefore ripe for testing.

Lastly, the validity of each claim is not something trivial to quantify. If we say a claim is "true," what does that mean? How much and of what type of evaluation needs to take place to ensure that the claim is trustworthy enough to be reused for other design efforts without the designers having to worry if they are considering false upsides and downsides with respect to a feature? Perhaps there needs to be some sliding scale to represent claim validity. The lowest value would represent the claim has undergone no testing while the highest value represents it is accepted worldwide without question. The claim model of Payne et al. [27] provides a start to this, but is not far enough along to be of the use discussed here.

These are not easy downsides to address. But if they are, claims evaluation can become a more refined process. Designers will have more faith in their results because they can be sure they are testing the claims that need testing and they can then know what validity of their claims are.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The process of developing a visualization for browsing cases, particularly with a focus on supporting problem-based learning, revealed lessons about the scenario-based design process as detailed to this point. The main contribution of this work is as a case study on applying claims relationships within scenario-based design to drive the development and evaluation of an interface. Those relationships have not been previously used in an actual design process and so far it appears they do positively augment scenario-based design.

In addition to that conclusion, there are smaller, related conclusions which provide an in-depth reflection on the main components of this work. They are summarized as follows:

1. **A claims-centric scenario-based design focuses the design process:** By forcing the designer to put themselves inside the user's mind and write scenarios, the interface design tends to be based upon realistic usage paths. With claims, the designer is encouraged to consider the effects, both positive and negative, of every interface

feature. Used properly, this ensures that features are added for specific reasons related to improving usability and not just because the designer thinks it will be neat. It takes time to learn how to properly write scenarios and claims, but the effort is well worth it. The second version of VaDeR appeared to have increased usability, perhaps because it focused on realistic usage scenarios and formalized the upsides and downsides of each feature.

But it was still not easy. Writing scenarios can be challenging because you have to be sure to mention all of the features without getting lost in narration. There is also the problem of not being sure how many scenarios are needed to be certain the design space was explored sufficiently, as one scenario only explores one usage path through the interface. This is related to the problem of not knowing when to stop writing claims. Knowing what exactly constitutes an individual feature, and not some simple combination, is not well defined. Keeping track of the claims that came from the scenarios, or past work, and their evolution was also tedious, as there is no automated tool to assist with this activity.

2. **Claim relationships aid in structuring the design, ensuring all problem claims are addressed, and identifying key claims for evaluation:** Putting a structure on the claims used allows the designer to trace their design rationale from the initial problem claims all the way to final design claims. And by encouraging the investigation of how claims interact and mitigate other claims, the key claims suitable for evaluation are revealed. So every time the design process moves forwards, these relationships highlight where the progress is taking place. Then if claims need to be altered, the other claims that might be affected are easily identified and examined for their possible alterations. This ensures design changes become fully accounted for within the entire process.

   But this process was also time-consuming. Each claim had to be compared to many other claims to examine if there was a connection. And because the guidelines are not strict enough yet, there is nothing stopping you from applying incorrect relationships which might lead to inappropriate designs and evaluations. Adding this layer of complexity onto an already complex tapestry of scenarios and claims may introduce too much structure, leaving the designer lost amidst a sea of interconnected claims.

3. **Visualizing case studies aids in problem-based learning:** VaDeR was a success when compared to its inspiration, UCS. The evaluators preferred VaDeR for performing the key tasks, which were identified as key for information gathering in problem-based learning. Using a visual timeline with elastic windows was seen as very useful when they wanted to explore a case study. For providing a reference to case studies for problem-based learning reasons (and any related reasons), a finished version of VaDeR should prove usable and popular. While it was specific to scenario-based design case studies in this instance, the approach developed can easily be adopted to serve any type of development record. The information visualization mantra proved to be a useful guide.

   But VaDeR can be improved by examining the design recommendations given by the evaluation participants. The interface is more complex than UCS and evaluators did note it was intimidating at first. Some preferred UCS when examining navigation issues because it was cleaner and had depth to its information. VaDeR can also benefit from being adapted to become more flexible and handle any type of development record, because the basic features are appropriate for any artifact-centered history of a design. Hopefully, in the future, anyone needing to examine the details of a development record will shun the academic publication or UCS and turn towards what they really want: VaDeR.

Overall though, scenario-based design combined with claims relationships appears promising for guiding structured design processes. It succeeded in this case in developing an interface that met its design goals and is suitable for future development. Claims seem to be useful in capturing design knowledge in manageable chunks thereby facilitating a detailed design process while not losing sight of the big picture.

6.2 Future Work

There are still interesting topics left to be explored about scenario-based design, claim relationships, and visualization for the purposes of problem-based learning. Tool support will be invaluable in managing larger projects as well as encouraging others to use scenario-based design, hopefully resulting in more criticisms and refinement of the process. The following gives several key efforts that will assist with this.

1. **Develop a visual tool for managing scenarios and claims, connected to a claims library:** LINK-UP [12] is a good start at providing some of this capability, but is nowhere near what is possible. A system where the user can arbitrarily insert scenarios and claims, then easily connect portions of the scenarios to claims and vice-versa, would be helpful in managing constantly evolving large projects. At any time, when reading a claim, the user can jump to the related scenario section. This linkage is provided in UCS and VaDeR, but the data behind it is not automatic and therefore would not scale to larger projects. Treating scenarios and claims as individual objects than can be manipulated visually on-screen would greatly increase the efficiency of dealing with them. Using a claims library, reusable claims can be brought in at any time, and connected to the appropriate scenarios.

2. **Develop a visual tool for managing claim relationships:** Similar to the broader tool for managing scenario-based design, some tool is needed to manage all of the claim relationships inherent in a design. There is already an attempt to visualize claim relationships for the purposes of browsing and locating claims [47] but this is not meant for managing a design process. Using this tool, one could treat all their claims as visual objects, and easily make connections when new claims are added. Then they

can better see how new claims will fit into the overall design. Depending on where and what types of claims are inserted, the tool may even be able to automatically suggest relationships that might be valid. When combined with a claims library and reusing claims, it will enable the designers to efficiently insert existing design knowledge into their present design structure, allowing them to see what effects that new claim may have on the design.

3. **Explore the use of visualization when using development records as references:** Problem-based learning is only one approach for a classroom. But it is almost certain that the techniques explored in VaDeR can be expanded upon and improved so that it can be used as the main source when a student needs to get any information from a development record. Those records can contain a lot of data, and information visualization is meant to deal with large data sets. It can be difficult extracting design knowledge for the purpose of reuse when one must read through many academic publications or go through UCS. By applying more efficient navigation techniques along with the ability to gain more insight into the data, the true work and contributions of a project are more quickly revealed, instead of just relying on short summaries that may not represent the complex value a claim can have.

These and similar efforts will yield potentially useful tools for managing a software project that uses scenario-based design with claim relationships. The resulting claims for the purpose of reuse are then ready for insertion into a claims library, sharing the design knowledge with the rest of the world. And if the world wants to explore the details of the project from where those claims were born, they use VaDeR.

Bibliography

[1] Patrick Baudisch, Nathaniel Good, Victoria Bellotti, and Pamela Schraedley. Keeping things in context: a comparative evaluation of focus plus context screens, overviews, and zooming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 259–266. ACM Press, 2002.

[2] Mordechai Ben-Ari. Constructivism in computer science education. In *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 257–261. ACM Press, 1998.

[3] Brandon Berry. Adapting heuristics for notification systems. In *Proceedings of the 2003 ACM Southeast Conference (ACMSE '03)*, pages 144–149, March 2003.

[4] Brandon Berry, Jamie L. Smith, and Shahtab Wahid. Visualizing case studies. Tech Report (number pending), currently available at http://filebox.vt.edu/users/brberry/VCS.doc, 2003.

[5] John M. Carroll. Cs 3724 course website. http://courses.cs.vt.edu/ cs3724/spring2003carroll/.

[6] John M. Carroll. Making use: a design representation. *Commun. ACM*, 37(12):28–35, 1994.

[7] John M. Carroll, R.L. Mack, Scott Robertson, and Mary Beth Rosson. Binding object to scenarios of use. *International Journal of Human-Comupter Studies*, 22:243–276, 1994.

[8] John M. Carroll and Mary Beth Rosson. Getting around the task-artifact cycle: how to make claims and design by scenario. *ACM Trans. Inf. Syst.*, 10(2):181–212, 1992.

[9] John M. Carroll and Mary Beth Rosson. Cases as minimalist information. In *To appear in: Proceedings of the 2005 Hawaii International Conference on System Sciences (HICSS 2005)*, 2005.

[10] John M. Carroll, Mark K. Singley, and Mary Beth Rosson. Integrating theory development with design evaluation. *Behavior and Information Technology*, 11:247–255, 1992.

[11] Tiziana Catarci and Isabel F. Cruz. Information visualization. *SIGMOD Rec.*, 25(4):14–15, 1996.

[12] C. M. Chewar, Edwin Bachetti, D. Scott McCrickard, , and John Booker. Automating a design reuse facility with critical parameters: Lessons learned in developing the link-up system. In *Proceedings of the 2004 International Conference on Computer-Aided Design of User Interfaces (CADUI '04)*, pages 236–247, January 2004.

[13] William S. Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.

[14] Yale Medical School Department of Laboratory Medicine. Case studies, deparment of laboratory medicine at yale. http://info.med.yale.edu/labmed/casestudies/casestudy.html.

[15] Ainslie Ellis, Linda Carswell, Andrew Bernat, Daniel Deveaux, Patrice Frison, Veijo Meisalo, Jeanine Meyer, Urban Nulden, Joze Rugelj, and Jorma Tarhio. Resources, tools, and techniques for problem based learning in computing. In *Working Group reports of the 3rd annual SIGCSE/SIGCUE ITiCSE conference on Integrating technology into computer science education*, pages 41–56. ACM Press, 1998.

[16] Nahum Gershon, Stephen G. Eick, and Stuart Card. Information visualization. *interactions*, 5(2):9–15, 1998.

[17] Steve Greenspan, David Goldberg, David Weimer, and Andrea Basso. Interpersonal trust and common ground in electronically mediated communication. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 251–260. ACM Press, 2000.

[18] Robin Jeffries, James R. Miller, Cathleen Wharton, and Kathy Uyeda. User interface evaluation in the real world: a comparison of four techniques. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 119–124. ACM Press, 1991.

[19] Hank Jones and Pamela Hinds. Extreme work teams: using swat teams as a model for coordinating distributed robots. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 372–381. ACM Press, 2002.

[20] Eser Kandogan and Ben Shneiderman. Elastic windows: evaluation of multi-window operations. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 250–257. ACM Press, 1997.

[21] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, 1986.

[22] D. Scott McCrickard, C. M. Chewar, and Jacob P. Somervell. Design, science, and engineering topics – teaching hci with a unified method. In *Proceedings of the 2004 ACM Technical Symposium on Computer Science Education*. ACM Press, 2004.

[23] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 249–256. ACM Press, 1990.

[24] Donald Norman and S. Draper. *User Centered System Design: New Perspectives on Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1986. Chapter: Cognitive Engineering.

[25] Board of Trustees of Leland Stanford Junior University. Stanford law school. http://www.law.stanford.edu/casestudies/.

[26] J. R. Parker and Katrin Becker. Measuring effectiveness of constructivist and behaviourist assignments in cs102. In *Proceedings of the 8th annual conference on Innovation and technology in computer science education*, pages 40–44. ACM Press, 2003.

[27] Catherine Payne, C. F. Allgood, C. M. Chewar, Chuck Holbrook, , and D. Scott McCrickard. Generalizing interface design knowledge: Lessons learned from developing a claims library. In *Proceedings of the 2003 IEEE International Conference on Information Reuse and Integration (IRI '03)*, pages 362–369, October 2003.

[28] Catherine Plaisant, Daniel Heller, Jia Li, Ben Shneiderman, Rich Mushlin, and John Karat. Visualizing medical records with lifelines. In *CHI 98 conference summary on Human factors in computing systems*, pages 28–29. ACM Press, 1998.

[29] Catherine Plaisant, Brett Milash, Anne Rose, Seth Widoff, and Ben Shneiderman. Lifelines: visualizing personal histories. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 221–227. ACM Press, 1996.

[30] Virginia Tech University Relations. Virginia tech - visual standards manual. http://www.unirel.vt.edu/style/vsm10.html.

[31] Mary Beth Rosson and John M. Carroll. Usability case library. http://ucs.cs.vt.edu/.

[32] Mary Beth Rosson and John M. Carroll. *Usability Engineering: Scenario-Based Development of Human-Computer Interaction*. Morgan Kaufmann, 2002.

[33] Mary Beth Rosson, John M. Carroll, and Con M. Rodi. Case studies for teaching usability engineering. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 36–40. ACM Press, 2004.

[34] R.J. Rost, J.D. Friedberg, and P.L. Nishimoto. Pex: a network-transparent 3d graphics system. In *IEEE Computer Graphics and Applications, vol.9, Iss. 4*, pages 14–26. IEEE, July 1989.

[35] Traci Royer. Using scenario-based designs to review user interface changes and enhancements. In *Proceedings of the conference on Designing interactive systems*, pages 237–246. ACM Press, 1995.

[36] John R. Savery and Thomas M. Duffy. Problem based learning: An instructional model and its constructivist framework. *Educational Technology*, 35:31–38, 1995.

[37] Harvard Business School. The case method. http://www.hbs.edu/case/index.html.

[38] Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, pages 70–77, November 1994.

[39] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison Wesley Longman, 1998. Chp 15: Information Search and Visualization.

[40] Elliot Soloway, Shari L. Jackson, Jonathan Klein, Chris Quintana, James Reed, Jeff Spitulnik, Steven J. Stratford, Scott Studer, Jim Eng, and Nancy Scala. Learning theory in practice: case studies of learner-centered design. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 189–196. ACM Press, 1996.

[41] Jacob Somervell, C. M. Chewar, and D. Scott McCrickard. Making a case for hci: Comparing materials for case-based teaching. To appear at 34th ASEE/IEEE Frontiers in Education Conference, October 2004.

[42] Bongwon Suh, Allison Woodruff, Ruth Rosenholtz, and Alyssa Glass. Popout prism: adding perceptual principles to overview+detail document interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 251–258. ACM Press, 2002.

[43] Alistair G. Sutcliffe and John M. Carroll. Designing claims for reuse in interactive systems design. *International Journal of Human-Comupter Studies*, 50:213–241, March 1999.

[44] Kam Hou Vat. Teaching software psychology: expanding the perspective. In *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pages 392–396. ACM Press, 2000.

[45] Kam Hou Vat. Teaching hci with scenario-based design: the constructivist's synthesis. In *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 9–12. ACM Press, 2001.

[46] Shahtab Wahid, C. F. Allgood, C. M. Chewar, and D. Scott McCrickard. Entering the heart of design: Relationships for tracing claim evolution. In *Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE '04)*, June 2004.

[47] Shahtab Wahid, Jamie L. Smith, Brandon Berry, C. M. Chewar, and D. Scott McCrickard. Visualization of design knowledge component relationships to facilitate reuse. Publication pending in the Proceedings of 2004 IEEE International Conference on Information Reuse and Integration (IEEE IRI-2004), November 2004.

[48] Cathleen Wharton, Janice Bradford, Robin Jeffries, and Marita Franzke. Applying cognitive walkthroughs to more complex user interfaces: experiences, issues, and recommendations. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 381–388. ACM Press, 1992.

[49] Dennis Wixon. Evaluating usability methods: why the current literature fails the practitioner. *interactions*, 10(4):28–34, 2003.

[50] Richard M. Young and Phil Barnard. The use of scenarios in human-computer interaction research: turbocharging the tortoise of cumulative science. In *Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface*, pages 291–296. ACM Press, 1987.

Appendix A

User comments from initial heuristic evaluation

These comments are from the evaluation that took place during the early stages of this project in the Information Visualization class, as explained in Section 3.8. No attempt was made to fix any spelling or grammar mistakes in the comments.

A.1 UCS

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

DOSENT APPLY, this is not a some real time system , just stuff that i am reading

clicking on a link will provide feedback based on the users action.

Don't think this really applies, as there is nothing currently happening

When I clicked on a link I received an explanation quickly.

you see the page you are at

the system didn't really have any real time interaction so there was no need for feedback...

I didn't know that when I clicked for more information, It took me to a new window.

There is a real problem with not making links known without mouseover

It explains every parts well.

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

Does that , I got the organization pretty quickly

Good explanation of all the definitions. The user can thus understand whats going on.

I have used a similar interface before in the "real world" so I guess this could be considered a fairly decent match.

Language was perfectly understandable by the target audience.

i understood the language (pretty much).

The language was good. However, the exploration part was a bit confusing.

you could easily tell what is under what with the links and the frame

the web page had a navigation bar on the left which gave the information the natural and logical order. The language was easy to understand.

Information was presented logically in the tree, but the pages were not so intuitive

This is easy to do

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

There is no emergency exit however it is easy to navigate back to the previous page since it is in the form of the familiar webpage frame.

May be confusing sometimes.

No dead ends in the program, and navigation was simple and easy

Links were accessible to anywhere I wanted to go.

the frame let you see which is the next level by clicking the higher link

the web page never led to such a situation. Undo and redo could in a sense have been supported by the back and forward buttons.

When new window opened, took up whole screen and wasn't sure if I could close the window or not.

It is easy to go to a different page

every action is pre determined and supported by specific paths, so it is hard to get a program error.

I can easily click other parts.

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

Users familiar with the terms will not have to worry however the terms used are not common to the lay-man...ie non cs majors.

Nothing was ambiguous that I could see

interface is not confusing.

Everything was clearly shown.

The webpage interface is conventional, but the fact that links are not easily distinguishable present a major problem.

no ambiguous terms or different terms for the same thing are used.

I understand what everything means

Make objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

When I click on something say under Information design. it will open in a new window. Then I will not be abel to see the definition of Information design which I might want to refer to.

If you recognize the words that are on the frame then it is easy to recall but to recognize is more difficult because if not familar with the word than you must recall.

While there were no instructions on how to use the system, the general layout was highly intuitive and simlpe to use.

It was easy to get around the site. Everything was recognizable.

you didnt have to rember much just click the link to go back to some info you needed

when following a link that opened in a seperate window, i searched for a link back to the original window before realizing a new window had opened and closing it. (no taskbar).

Information is repeated when necessary to provide orientation.

Well known paradigms and pre-concepts are followed strictly, so there are no "surprises"

I can just navigate down the left side

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

They dont

The information contained was straight to the point

interface provides relevant information.

All the information seemed relavant.

didnt have lots of extra info

the links were short and concise as to what the pages they were linked to were about.

Design is quite minimalist... It probably needs a little bit more to distunguish the difference between objects.

No unneeded info here everything looks good

I think it contain all relevant information.

I was able to achieve an overall understanding of the events that occurred.

Yes I am familiar with the format so I could understand the feedback clearly

This visual display was much better than pdf file. It was structured in a much better way. Easy to understand also. One reason may be we have already encountered same web site design to represent our project in our HCI class

The site was set up to explain what each step meant in the brainstorming phase.

easy to see the whole structure fromt he frame

Organization was easily understandable.

Opening pages made sense

It is really easily to see overall because of left box.

I understood how to navigate the case study using the interface.

Simple to understand. Follow the hyperlinks.

Just had to point and click.

being similar to the standard interface for a webpage , it was easy to navigate.

Everything was easy to navigate through, except when needing to go back.

Once I realized where the links were, it was easy.

It explains about each section, but I am not sure how to navigate the study using it.

I can access any details that I want in an efficient manner.

I can do this easily.

Yes, because it has a Windows Explorer like view and very intuitive.

There was an easy way out of everything and everything available on the site was located on the left hand side of the site.

just click the link and it tok me there

all the links were concise and the webpage was well organized in a top down design making it easy to find what exactly i wanted.

Details took over the entire screen and was difficult to go back.

It is hard to distunguish objects, so details are harder to find than they should be.

I can easily click any section from left box.

I was able to easily understand how the information was organized.

There could be a more concise organization of the information with the frames pages...ies nesting and etc

Each part had clearly indented sub parts which shows that what goes under what topic.

Everything was organized very well.

the indents helped show what belonged under what

once again, the standard web page design made it easy since i was familiar with it.

Clearly marked categories and sub-categories.

I easily see the what it is about from overall and can esaily get information from two different boxs.

A.2 VaDeR prototype

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

the system had all pertinent information. Layout took a minute to figure out, but all information was presented.

With each click the system responds so there is immedeate feedback from the users action which is hlepful.

information is clearly provided on each item.

as soon as you click you got results

When clicking on certain buttons feedback was given on what happened at certain times.

The full time line is laid out

It did not explain well to the users.

double clicking on a slide, immediately pulls it up enlarged.

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

The language was simple, and easy to understand. The layout was logical

The particular icons and letters used to represent task are common say the letter to represent the beginning of each word that the icon does.

language used are appropriate and easy to undestand.

didnt really see the exact order

The letters are easy to catch on to, but are not imediately obivous.

The communication wsa done well.

The letter icon representations were not immediately understood

No real complex words or terminology.

Looks simple enough

I do not know what is c, s, or other letters in there

the interface was similar to a navigating the webpage, with thumbnails of the slides on the left. slides on the left.

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

There were no "dead-ends" in the interface. Any click would take you to a certain screen.

There are no emergency exits back to a previous page that I can see.

It is easy enough to change to any desired state (or back from an undesired one).

One is able to go from one aspect to the next however it was not very clear at first how and not very clear how to exit when finished.

Undo and redo were not listed, but there was a back button available as well as the main interface that stayed on the screen

Was easy to move around in system, but no real exits for different parts (only exit was clicking new part). No exit for entire program.

I don't see any emergency exit option?

I can go any slides eaily.

i never had encounter this situation, but the system seems to have the capabilities

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

The repitition of symbols (notepad, 'c++', checkmarks) don't give too much information.

Each letter and icon seems to be different than the next.

some sysbols are the same.

The meanings of everything was fairly consistent with the rest of the program and Windows in general.

The same icon was used several times in the timeline. This could be confusing.

The interface was consistent

It's pretty straight forward

the icons were similar to other editing platform icons.

all terms and situations are consistent.

Make objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Going from place to place only required a click, and was easy to do.

Although the objects are visible perhaps it would have been more reasonable to make an option where more than one categorey is open at a time.

objects, actions, and options are very visible.

it was clearly laid out

I did not have to remember much to use this interface

A person has to click several times to obtain information. At first it is not clear how the information is to be accessed

Again, the icons were not immediately understandable

ything is very visible to me.

all the controls were at the top of the screen, making it easy to access.

The abbreviated letters, like P and A needed to be remembered.

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

The interface was pretty sparce. Provided only the important information.

All of the information I saw was relevant.

no irrelevant info.

it didnt have extra junk on screen

Only essential information was presented.

The information presented was necessary, though the interface itself could potentially me shortened in height.

Was nicely laid out and easy to comprehend.

I could not find any irrelevant things.

every communication is kept to a minimum

I was able to achieve an overall understanding of the events that occurred.

After figuring out the system with some exploratory clicks, seeing the timeline and reading the descriptions of events was easy.

I could figure out from the description of each icon what the system was attempting to show me.

Information could be accesses by months or by stages etc. Easy to see everything and also visually appealing.

Overall, the interface presented information well.

Click actions made sense

Confusing at first, but then things began to make sense.

It is kind of easy to see because it divides by month and sections by picture, but i do not know what is the letters mean.

I understood how to navigate the case study using the interface.

Again, took some exploratory clicking, but in the end using the interface was generally simple.

I understood what to do and what icons to click to obtain given task.

Simple click on an item to get more detail about it. Everything was hyperlinked in powerpoint.

i had to click around at first i wasnt sure what things like the ”S” button do

The icons could have been more descriptive

Was easy to navigate through.

I can access any details that I want in an efficient manner.

It's possible to click anything you want at any time

This is true but I cannot access multiple sets of data that I may want to look at the same time.

More tha one ways to access the same item , use whichever is comfortable .

when i knew what i wanted to do it was fast and easy

It is easy to get to any piece of information.

By clicking on the icons I was able to access information.

You have to click twice on an icon once something is already opened

I can easily see what happened which date.

I was able to easily understand how the information was organized.

The layout was pretty logical and easy to understand

Yes I was it took some random clicking and whatnot to get hold of

very organized.

it was clearly laid out into different sections

The information was clearly broken down into categories.

Everything was organized well.

Information was well organized

Kind of, but not really

Appendix B

Mapping UCS data to VaDeR

While the prototype was coded for only one case study, the techniques it uses can be extended to work for any case study. The main area where further development is needed is with the timeline. If there are many icons and their ordering won't permit each to display fully, strategies for dealing with overlapping icons will need to be explored. There are plenty of techniques to choose from, such as making the icons smaller or grouping a set of icons into some singular representation, that can be found in the information visualization field. But for now, this mapping assumes it is possible to fit the artifact icons as the example was able to do.

The guidelines for mapping UCS data into VaDeR thus follows:

- Place the title in the upper left frame, link it to the introduction
- Use the case study introduction as the default item shown when VaDeR is opened
- Starting with the 'Requirements Analysis' menu, search through each submenu
- Build up a page that contains a list of each artifact from every submenu

- If a menu is empty, leave that title in the page, but lighten the color
- If there are no artifacts of a certain type, turn off the corresponding filter but leave it on the display
- Link the menu title (from the top bar) to that page
- Within the design phases, also build up a page each that lists exploration, scenarios, claims, and prototypes
- Map each artifact to the appropriate icon
- If the artifact contains a link to an external file, give the artifact icon an extra distinguishing mark
- If the artifact belongs to a design phase, also give it the assigned color as a background
- Place the earliest, date-wise, artifact at the top of the appropriate column
- Place the latest, date-wise, artifact at the bottom of the appropriate column
- Place the remaining artifacts so that temporal ordering is preserved, even across columns

Automating virtually the entire process should be possible, although not trivial. It may be necessary to manually find if any evaluations match to a specific design phase and to ensure the scenarios and prototypes were found properly since they both are in the 'envisionment' submenu.

Obviously, a much easier solution is to create the development record in such a manner that VaDeR can easily parse it, instead of going through UCS. Creating an XML schema to capture artifact data is one viable method.

Appendix C

Tasks and claims exploration results

Comments and observations from the evaluation in Chapter 5 are collected here. They are separated into each part of the evaluation. Spaces between the comments indicate the different participants and they are in the same order noted in Table C.

Participant	SBD	UCS	Information Visualization
1	very	very	very
2	very	very	very
3	very	somewhat	very
4	very	somewhat	somewhat
5	somewhat	somewhat	somewhat
6	somewhat	not at all	somewhat
7	somewhat	not at all	somewhat

Table C.1: Participants familiarity rating.

C.1 UCS

C.1.1 Comparison task

- browse options, menus appear in order
- description/artifact may be confusing model to new user
- false affordance with some non-hyperlinks
- closes pop-up windows each time
- no big picture, seems disjointed
- no overview, bottom-up organization
- hard to connect details
- need better layout - use mantra
- can have ton of windows open
- navigation should expand out to right

- straight to RA, AD, found claims
- went to resize but accidentally closed window
- opened 2 windows, resized, moved
- used internal links
- expected to find int claims but weren't there

- looked for scenarios, claims
- opened 2 windows, compared, resized windows

- used internal links in scenarios
- managed windows, compared contents

- closed pop-up windows, read separately
- brought up multiple windows, moved around

- browsed multiple menus to find scenarios + claims

- opens multiple windows, goes back and forth
- problem with window management, pop-ups

- searched through menus
- opened multiple menus, moved around
- really like internal links
- organization wasn't intuitive

C.1.2 Iterative process task

- very difficult to see
- don't see how classification is useful
- no version number indication
- timeline with link representations would help

- want to see an overview
- a table of contents, process description would help
- menu order might suggest temporal order
- expected to find results, but got empty menu
- must click to a 3rd level to see dates

- browse through menus
- go back and forth between menus
- dates not a guide, but indicate possibility

- opened testing, looked for results
- hard to see connections
- must look at dates
- structure + dates = hard to follow iteration

- want to go from prob scenarios to all design scenarios, in order
- looked for different levels of design
- went back and forth comparing dates

- looked for overview
- expect to go through linearly
- must search for next date
- hard to differentiate iterations

- look for details and dates
- look for some change indication, version details
- must compare dates, lots of clicking
- not easy or intuitive
- would expect multiple dates in categories for iteration

C.1.3 Usability

- very difficult, visualization needed for time
- marker needed for empty menus
- want to control when menu closes
- windows little annoying, gets in way if just clicking around
- not clear of artifact context, no process diagram
- want option to open in new window or same window
- colors, fonts good
- want hyperlinks with menu titles, site structure tree

- using time is time consuming, tedious, not impossible
- empty menus frustrating, wonder about quality of other content
- want to open multiple menu levels at once
- can keep pop-up windows open for comparison

- can use title to get some context, but not so much
- very difficult, no clue how to follow iterative process
- don't mind window management, like resize ability
- opening multiple menus would help for comparing across case studies

- hard to search by date, can't search by range
- expect to find something in empty menus
- 2 menu opens would be useful in comparing
- pop-ups create context loss, can manage windows but disturbing
- hard to retain context
- hard to follow iteration, must show what changed
- would rate interaction 2/3

- using dates hard, not organized in order
- had to click menu to learn it was empty
- would be useful to open 2 menus, visual change subtle when opening menu
- used to pop-up windows, don't lose navigation place
- hard to determine context
- difficult to follow iteration
- must manage tabs with truncated titles when multiple pop-ups opened
- like look + feel

- can use dates, but must go back and forth
- expect content in all menus
- annoying that other menus automatically close
- nice you can control all the pop-ups
- titles useful for context
- possible to follow iteration, but must keep track of dates

- no way to compare multiple case studies

- have to go through all menus to get general ordering
- empty menus confusing
- might be helpful to expand multiple windows
- not a fan of pop-ups, hides other info, hard to juggle windows
- generally know where you are
- clean interface, easy to navigate

- no visual sequencing, so hard to use dates
- no difference between placeholder and empty menus
- don't want menus to close unless told to
- pop-up size acceptable, windows don't open precisely over each other
- menu titles aid context
- window management not really bothersome, but do have to move around
- close button at bottom of pop-ups would help

C.2 VaDeR

C.2.1 Comparison task

- began by clicking icons
- 'much better' 'this is nice'
- makes more sense
- requires some learning
- more affordances would help
- 'amazing' 'good'

- filtered out irrelevant icons
- don't see iteration

- knows what came after events

- used active frames, found scenarios with timeline
- context useful
- can turn off irrelevant icons

- correctly guessed at icon meaning
- used icons + frames to compare
- might be helpful to click on filter icons to get menu/submenu

- filtered out irrelevant icons
- used timeline icons + active frame
- same as ucs but easier here

- easily compare scenarios
- hotkey to switch frames would help, visual cues to active frame would help

- able to find claims easy
- would like internal links to open in opposite frame
- thought brighter frame color meant active, at first
- possibly drag links to open in other frames

C.2.2 Iterative process task

- no version control, visual mapping would help
- see there's no iterative design in this case study

- select/unselect all filter would help
- look for changing colors or more RA down the timeline to see if iteration exists
- some iteration occurs because of prototypes

- proximity on timeline implies connection
- another visual connection between scenarios + claims would help
- could use dates to see changes

- tooltips for icons would help
- ordering good for task
- should select active frame by clicking top

- compared test to design, saw a test occurred before some design
- colors don't change, so can tell there is no iteration
- would follow iteration top to bottom with zig-zag left-right

- used timeline
- more helpful, easy to navigate
- looked for colored icons to follow iteration

- easily see process was sequential
- expect to see mixed colors for design changes

C.2.3 Usability

- timeline helpful for using dates
- flat menus helpful
- very good for using 2 menus
- want to customize number of frames
- timeline gives context
- 'like this a lot'
- needs better affordances
- can learn icons, legend helps
- design phase colors helpful

- colors useful with frames
- another marker helpful for empty menu
- filters helpful for big case studies

- using time straightforward, intuitive
- could be too much info in menus, but less interaction
- must use mouse to control active frame, hotkey would help, more visual cues to indicate active
- automatic 50/50 split for frames would help, more frames useful as option
- easy to know context, icons reinforce it
- active frame interaction requires mental tracking
- learned icons by clicking around
- empty menus easily ignored
- filters helpful, maybe click on icons too

- much easier to use time in search
- useful to have all info in flat menus
- helpful to open 2 menus, can see changes
- once figured out, window management more efficient, intuitive
- much better for maintaining context
- much easier to follow iterative process
- would rate interaction 3/3
- icons overwhelming at first, but able to quickly figure out
- design colors useful, active frame colors useful
- filter help for following iterative design

- really easy to see order
- easier to manage menus
- helpful to open 2 menus

- window management easier, wouldn't want > 2 frames
- icon highlighting helps context
- good overall design
- icon design appropriate, don't use white for multiple categories
- design colors helpful
- easy to tell color mapping with active frame
- don't have to search just to run into empty menus
- select/deselect all filters would help
- 'seems pretty good'

- lot easier to use time in search, don't have to look at dates
- flat menus helpful, but can't look at just one submenu
- opening 2 menus, can see more project at once
- could use more windows with UCS, but hassle
- can't lose details by clicking other windows
- can't get lost with overview + menu + artifact open at once, left to right
- seeing all at once helps follow iteration, follow colors
- design colors helpful, frame colors too
- filter helpful to find specific content

- easy to use time, visual relation
- easier to go through menus
- frames better than pop-ups
- can go down timeline to follow iteration
- after awhile, easy to understand icons
- must get used to design colors
- no wasted effort in discovering empty menus
- filters useful when too many icons

- lot of navigation possible, but more powerful

- can see sequence of events by clicking
- expect icons to link to submenus
- good frames, but want option to make more
- rather use these frames than pop-ups
- know context, but can't differentiate versions
- icons make it easier to follow iteration
- more visual cues needed to connect components
- more help links
- click on slits to change active frame
- learn icons eventually, not so hard
- filters for versioning would help

C.3 Comparing UCS to VaDeR

- vader more efficient, better, easy once learned
- 'winner' - vader

- vader provides direct access
- UCS paths help in keeping context
- don't mind resizing windows/frames, but controls to snap size would help
- more frames as option in vader
- timeline allows for following iterative design

- get a lot by looking in vader
- for navigation, vader beats ucs hands down
- frames help for reading details
- vader better for comparison, following iterative design

- vader minimizes clicking, have to go through many menus in ucs
- ucs not horrible, but vader better
- vader better for comparing, follow iteration
- ‘really well designed’

- ucs gives identical overview regardless of content
- vader gives much more info but doesn’t use more space
- vader much better for using time, don’t have to look at dates
- navigation about the same
- vader allows 3 different levels at once, doesn’t automatically collapse menus
- option to have more frames in vader would help
- vader much better for iteration
- ‘less reading, but more insight’, ‘much cooler’ - vader

- ucs simple, not sure where to start in vader
- vader slightly easier navigation
- context easier in vader
- vader better for details because of no pop-ups
- easy to follow iteration in vader

- see more text in ucs, really like using depth navigation to find info
- for finding version, vader is better
- reading details is better without pop-ups
- rather use vader to compare artifacts, follow iteration

Vita

Brandon Berry was born in Virginia Beach, Virginia sometime before this document was written. After many years of some stuff, he found himself attending Virginia Tech in the fall of 1999 and it was inside many classes he was sitting. Then he somehow, unexplainably, managed to graduate with a BS in Honors in Computer Science and a BS in Mathematics in the spring of 2003. Deciding to gain bragging rights and surpass his father's academic achievements, he stuck around to pursue his Computer Science MS degree. General Dynamics in Fairfax, Virginia decided to give him money to do things and therefore he will begin his inevitable illustrious career as a systems engineer. No matter what, he will persevere. He does not know why this vita rhymes. But he is certain he has committed no crimes.