Developing and Evaluating the (LUCID/Star)*Usability Engineering Process Model

By
James William Helms

Thesis submitted to the Faculty of

the Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTERS OF SCIENCE

in

Computer Science and Applications

Approval Names:

_____	_____
Dr. H. Rex Hartson, Committee Chair	Dr. Deborah Hix

_____
Dr. James D. Arthur

May 3, 2001
Blacksburg, Virginia

Keywords: Usability, HCI, Process Model, Usability Engineering, Software Engineering

Developing and Evaluating the (LUCID/Star)*Usability Engineering Process Model

James William Helms

(ABSTRACT)

In recent years, interactive systems developers have increasingly included usability engineering and interaction design as an integral part of software development. With recognition of the importance of usability come attempts to structure this new aspect of system design, leading to a variety of processes and methodologies. Unfortunately, these processes have often lacked flexibility, completeness and breadth of coverage, customizability, and tool support. This thesis shows the development of a process model, that we call LUCID/Star*, which addresses and overcomes the characteristics lacking in existing methodologies and an evaluation of its application in a real-world development environment. To demonstrate the goal of this thesis, we have used a combination of empirical and analytical evidence.

The (LUCID/Star)* process model for usability engineering grew out of the examination, adaptation, and extension of several existing usability and software methodologies. The methods that most greatly impacted the creation of (LUCID/Star)*were the LUCID Framework of interaction design, the Star Life Cycle of usability engineering, and the Waterfall and Spiral models of Software engineering. Unlike most of these, we have found that a sequence of cycles (each of which produces a product evolution) is a more effective analogy for the interaction development process. A sequence of cycles is more modular and makes it easier to focus on each cycle separately. Working with Optim Systems, Inc. in Falls Church, VA we instantiated the process model and introduced it as a process to develop a web-based device management system. (LUCID/Star)* performed remarkably in the Optim case, overcoming the tight constraints of budget and schedule cuts to produce an excellent prototype of the system.

Table of Contents

Figures

Tables

1. Problem Statement

In recent years, interactive systems developers have increasingly included usability engineering and interaction design as an integral part of software development. Efforts in both the corporate and academic arenas have greatly increased the receptiveness toward user-centered design, and in many software development projects usability is becoming a core objective. Such efforts have established usability as an essential part of interactive system design.

With recognition of the importance of usability come attempts to structure this new aspect of system design, leading to a variety of processes and methodologies. Unfortunately, these processes have often lacked flexibility, completeness and breadth of coverage, customizability, and tool support. Existing processes often demonstrate one or more of the following undesirable properties:

- Structure the development cycle as a rigid, linear, stage-by-stage progression
- Force the adoption of new usability methods without accommodating existing methods and techniques for development activities
- Cannot be tailored to meet the special needs of a given project environment
- Restrict usage too specifically to certain interaction styles and applications types
- Do not support integral generation of documentation work products
- Fail to establish realistic connections to the software engineering process for developing the functional core
- Are not tool supported

2. Goal of research

The foremost goal of the research is to: develop a process model, that we call LUCID/Star*, which addresses the issues discussed above and evaluate its application in a real-world development environment. The second part of this goal, the evaluation, can be realized by demonstrating the success or failure of (LUCID/Star)* in a real-world development environment, in terms of these criteria:

- Flexibility of (LUCID/Star)*as a non-linear, iterative progression through the user interaction development cycle
- Ability to accommodate existing methods and techniques for development activities
- Ability to customize the (LUCID/Star)*process model to meet the special needs of a given project environment
- Applicability of the (LUCID/Star)*process model to a variety of interaction styles and applications types
- Ability to include generation of documentation work products as an integral part of the process
- Feasibility of supporting realistic connections to the software engineering process for developing the functional core
- Compatibility with the creation of effective usability engineering support tools

3. Approach

To achieve the goal of this research, as stated in the previous section, we have used a combination of empirical and analytical evidence. Working with Optim Systems, Inc. in Falls Church, VA we instantiated the process model and introduced it as a process to develop a web-based device management system. We have assessed this use of a process model instance by:

- qualitative empirical data observed within the case study, augmented by
- analytical argumentation, where empirical evidence was unavailable.

Answers to the questions posed in the following sub-sections, taken together, satisfy the goal of this research. Note that the application and observation of the process model to a single project can not directly support all of the points in the following subsections. However, support for the points has been be gathered from drawing on the project and its extended environment, including other work at Optim Systems, Inc. and other related work at the Usability Methods Research Laboratory at Virginia Tech.

*3.1. Flexibility of (LUCID/Star)*as a non-linear, iterative progression through the user interaction development cycle*

Existing methodologies and processes for designing user interaction often enforce a rigid, inflexible progression of stages through the design and development cycle. These methodologies leave no room for formative redesign (backtracking) or midstream iteration. What is needed is a structure that can be flexible, non-linear, and iterative without sacrificing design quality. Realizing the shortcomings of existing usability process models and methodologies, we set out to develop a new process model that exhibited all three desired traits. Our model, called LUCID/Star*, grew out of the examination and modification of earlier research in both the disciplines of usability and software engineering. Examining models from software and usability engineering allowed us to take the "best of both worlds" so that our model can realistically fit into and be compatible with existing real-world overall software development environments, while still devoting the special attention to interaction design essential to achieving high usability. From user interface development methodologies, such as Cognetics' LUCID framework [Cognetics, 2000] and Hix and Hartson's Star Life Cycle [1993], we developed four categories of interaction design activities (Analyze, Design, Implement, and Evaluate) and focused the completion of each cycle of activities on evaluation. We gleaned our model's iterative and evolutionary nature from exploring Boehm's Spiral Model [1988], and the Waterfall Model [Royce, 1987] of software design made us aware of some of the constraints and structure we would need to accommodate in real-world usage (see section 4). We demonstrate a new twofold approach to iteration embodied in LUCID/Star*: iteration within activities of a cycle and iteration among cycles. Also, we show the non-linearity of the model by illustrating how implementers of the model can move freely among cycles.

3.2. Ability to accommodate existing methods and techniques for development activities

Existing usability methods are often so differently structured, specialized, and inflexible as to be incompatible with one another. Practitioners often find that they must use Method A or Method B, but they cannot find a way to combine them so that they may draw on the techniques of each method that are appropriate to a given project. This raises the threshold of difficulty and cost when a development team wishes to consider new methods, since they must essentially learn each new method from scratch. Also, developers may have to give up effective existing techniques, because new usability processes often come packaged with a predefined set of techniques that leave no room for incorporating existing techniques. Usability practitioners need an overall *process model* that can be adopted without requiring them to learn a new, rigid structure every time they need to change methods. In (LUCID/Star)*we endeavored to create a model that accommodates a variety of existing usability methods and techniques, and expects to accommodate new methods not yet developed. We felt that since the (LUCID/Star)*model does not require specific techniques or methods to fulfill any activity in the structure (although it does suggest possible choices from among currently available techniques), the implementers would be free to choose from existing techniques or adapt new techniques to fulfill the model. In this way, (LUCID/Star)*would allow practitioners to merge existing methods and techniques with new ones via selective (plug and play) substitution. In essence, we believe the activity types for cycles (Analyze, Design, Implement, and Evaluate) are so basic that the corresponding steps of any existing methodology can be used to instantiate them. We demonstrate this compatibility with existing methods by showing how existing user interaction design methodologies can be realized as instances of the (LUCID/Star)*process model and how new methods and techniques can be merged with existing ones by selective substitution.

3.3. Ability to customize the (LUCID/Star)*process model to meet the special needs of a given project environment

Since different usability methods favor different project environments and are more effective under specific sets of constraints, no single method exists which can easily accommodate every project setting. Unfortunately, each real-world project can have its own unique development environment, budget, schedule, and project management style. So a methodology that a development team invested in learning and that was effective in one project may not remain effective in another project setting. For example, a reduction in time or budget invested in a project may demands less extensive (or entirely eliminated) lab-based usability testing or less iteration of early design. Conversely, an increase in project schedule or budget may result in more extensive usability testing and more iteration of early design. Practitioners need a process that can be adapted to the needs of a given project and can be customized so that a change in constraints between projects has minimal impact on the quality of the design produced by the process. Since we developed a process model and not a methodology, any part of the process is an instance of what is possible, and thus is not a required step in the design evolution. For each project, developers create a customized process instance, including or excluding cycles, activities, or entire iterations as the constraints allow. The process can be

different each time and can even be modified as the project progresses. We show the feasibility of this by analyzing the project constraints and how the process model adapted, as it was instantiated at Optim. We demonstrate how (LUCID/Star)* adjusted to meet the specific needs of the development team there, and how it was structured to continue working for them in the future.

*3.4. Applicability of the (LUCID/Star)*process model to a variety of interaction styles and applications types*

As technology changes, usability engineers are no longer just faced with the charge of designing just desktop GUI's, but are also facing the challenge of designing for the web, virtual environments, peripheral displays, gestural interfaces, and voice interaction. These new technologies present a new set of usage situations and usability requirements and require new techniques to accommodate them. The fast-changing application technology calls for a process model that does not force practitioners to perform predetermined techniques to fulfill specific pieces of the model. The model should accommodate new techniques, which apply more effectively to the differing interaction styles. This is a key consideration when developing the process model, because the varying nature of the interaction styles designed with this model may restrict the available techniques. For example, peripheral displays are displays and devices resident in the periphery of a user's work environment that provide the user with information regarding events outside of the user's current work focus. Examples of peripheral displays include stock or news tickers, Windows taskbar status alerts (e.g. Eudora's 'new mail' icon), and even a car's odometer. Traditional lab-based usability testing cannot be used to evaluate peripheral displays, since these displays are often intended for long-term usage across many other tasks and reside in the background of a user's normal actions. A modified form of testing must be developed to accurately assess the usability of such devices that more realistically accounts for the special properties of these devices. For instance, one possible technique involves using the peripheral display while performing another more involved task [McCrickard, Stasko, and Catrambone, 2000]. For peripheral displays, this new technique fulfills the same basic part of the interaction design cycle (namely evaluation of hi-fi prototype) as lab-based testing would for a desktop application. Since the (LUCID/Star)*process model is designed to allow implementers to choose appropriate techniques to fit their application type, it provides a viable structure that can be customized in such a way that makes designing for traditional and alternative interaction easy. Optim Systems, Inc. provided an excellent testbed for the model in this respect, as opportunity existed to instantiate the process model on some very different interfaces and devices. We were able to see the process model at work on a more traditional remote device management application, a web-based application for communicating with non-PC devices, and wireless interface for PDA's and cellular phones, the designs of which were much more highly constrained. By analyzing the collected work products, we determined the usefulness of the process model in designing the interaction for each application. This allows us to show whether or not the structure is applicable to any application type.

3.5. Ability to include generation of documentation work products as an integral part of the process

Most existing usability methodologies and structures do not even mention documentation, but we know that in real world software development, thorough documentation is essential to a successful design. One of the cornerstones of the (LUCID/Star)*process model is the integration of work product documentation into the overall structure. Documentation work products are generated and evolve through every step of the process model. This documentation is held in a common database that can be accessed and used by any part of the process. This means that no work is lost, and nearly every piece of documentation can feed other design activities. In order to demonstrate the integration of documentation into the process model, we show how documentation was generated, iterated, and refined through out the instance of the process model implemented at Optim Systems, Inc.

3.6. Feasibility of supporting realistic connections to the software engineering process for developing the functional core

Another aspect of the development process, which is markedly lacking from existing usability engineering methodologies, is the establishment of connections to software engineering. Interaction design and software engineering share many similarities in process, needs, and in origin. With the development of a general structure for interaction design, these similarities should become apparent again. The loss of cohesion between the two disciplines lies in the details of designing backend, functional core software versus designing user interaction. Usability engineering focuses on the user's satisfaction and ease of use, while software engineering zeroes in on functionality and reliability. Despite the disparate sub-goals, both have the end user in mind; the disciplines simply satisfy that user in different ways. Herein lies the key to discovering connections between the two fields. One must analyze the purpose and goals of the processes to discover where they are the same and where they are necessarily separate. The (LUCID/Star)* process model can be visualized as a life cycle in much the same way that current software engineering methodologies are, and thus lends itself to integration with these models. In effect, the (LUCID/Star)* instance becomes a life cycle within the larger software development and overall project life cycles and does not have to impact or change most of what's done to develop the functional core (except for interaction with the user interface software development). Also, our process model is designed to be the vehicle of communication between the interface designer and the back-end software developers. We observed how such communication fed the design of the Optim product by narrowing the connections to the backend database and better defining communication in general. From this analysis, we draw conclusions on how to better facilitate defining such connections and improving communication between functional core and interaction design.

3.7. Compatibility with the creation of effective usability engineering support tools

Existing usability methods often provide the 'what' without the 'how', meaning that they often tell practitioners what general step to do next without providing useful instructions or tools on how to do it. At best, such methodologies provide checklists or forms to follow. This often leaves practitioners no clear guidance in implementing the usability techniques prescribed. Usability engineers need tools to help them manage the interaction design process, and to provide direction in performing specific usability techniques. (LUCID/Star)*lends itself to integration with several types of tools. Process management/customization and instructional tools are two classes of tools that have been under development at Virginia Tech for integration into the (LUCID/Star)*model. Process management/customization tools help practitioners define the structure of their instance of the process model, allowing them to customize and visualize the structure to fit their software projects. Thus, for example, in the cases of wireless versus virtual environment interface development, the tool could offer different but equivalent techniques to fulfill the same general activity of testing usability specifications. Along with management tools, instructional tools help usability engineers implement specific techniques to instantiate the activities in their customized process. To show how (LUCID/Star)*can be fitted with a suite of usability tools, we examine initial designs for such tools done by others in the Usability Methods Research Laboratory at Virginia Tech.

3.8. Summary

Table 1 below summarizes the points described in the sections above, showing the section number, the goal described there, and how we intend to show that (LUCID/Star)*can fulfill that goal.

Table 1: Summary of research goals

Section #	*Research Goal*	*We show…*
3.1	**Flexibility of (LUCID/Star)*as a non-linear, iterative progression through the development cycle**	– Iteration, both within activities of a cycle and among cycles. – That implementers can move freely between cycles.

3.2	**Ability to accommodate existing methods and techniques for development activities**	– How existing methods and techniques can be "plugged in" to instantiate almost any development activity in the structure. – How the activity types of a cycle (Analyze, Design, Implement, and Evaluate) are so basic that they can fit the steps of any existing methodology.
3.3	**Ability to customize the (LUCID/Star)*process model to meet the special needs of a given project environment**	– That process cycles and activities can be removed or replaced to meet the constraining parameters of a project environment (e.g. budget, schedule, management style, etc.) – That activities or cycles may be added to include any specialized techniques, methods, or custom requirements of the application domain.
3.4	**Applicability of (LUCID/Star)*process model to a variety of interaction styles and applications types**	– The generality of the structure as demonstrated by the outcome of the design process on multiple application types. – How the structure is application independent, while techniques are application specific. – How implementers may substitute, for various development activities, methods and techniques that are highly specialized for a given interaction style or application type.
3.5	**Ability to include generation of documentation work products as an integral part of the process**	– How documentation and its production are built-in (potentially) to all activities of the process model. – How documentation was created and evolved throughout the project.

3.6	**Feasibility of supporting realistic connections to the software engineering process for developing the functional core**	– How to exploit similarities of the process model to existing software development processes – Techniques for communicating between interface development and functional core (backend) development, – How the process model defines a life cycle within the larger software and overall project life cycle
3.7	**Compatibility with the creation of effective usability engineering support tools**	– The already established feasibility of usability engineering support tools for instantiating and managing the (LUCID/Star)*process and for performing the activities within the model (especially analyzing, classifying, reporting and managing data for usability problems found in usability testing).

4. Background and Related Work

The (LUCID/Star)*process model for usability engineering grew out of the examination, adaptation, and extension of several existing usability and software methodologies. The methods that most greatly impacted the creation of (LUCID/Star)*were the LUCID Framework of interaction design, the Star Life Cycle of usability engineering, and the Waterfall and Spiral models of Software engineering, as shown in the following subsections.

4.1. LUCID

LUCID – Logical User Centered Interaction Design – began as a way of describing the approach to user interface design at Cognetics Corporation of Princeton Junction, New Jersey [Cognetics, 2000]. LUCID is based on a sequence of six stages, which carry the evolution of a product from its initial conception to the final deployment as a product release:

- **Envision**
- **User and Task Analysis**
- **Design and Prototype**
- **Evaluate and Refine**
- **Complete Detailed Design and Production**

- **Deploy and Follow Up**

LUCID presents a novel framework for designing interaction, organizing the development as a sequence of basic activity types. This leaves developers the freedom to instantiate those activities in various ways. However, each activity type is done once over the entire development cycle. We felt that the activity types described were indicative of what usability engineers do for each evolution of the product. With this in mind, we explored the activity types describe in the stages of LUCID and found four essential activity types that are represented in those stages: Analyze, Design, Implement, and Evaluate. We have built those four basic types of activities into each cycle of (LUCID/Star)*, so that they are applied to each evolving form of the product.

LUCID is called a "framework" because it does not rigidly request specific techniques, a characteristic inherited by LUCID/Star*. However, unlike LUCID, (LUCID/Star)*also provides an organizing framework for ordering, structuring, planning, scheduling, and documenting all the interaction development activities. Users are at the heart of LUCID, starting with a consideration of the target user groups and user task analysis. Feedback from users is an important part of the iterative cycle of the design, with possible additional usability tests during implementation.

4.2. The Star Life Cycle

The Star life cycle concept, derived empirically from extensive observations of real-world developers [Hartson and Hix, 1989], is an evaluation-centered iterative usability engineering process for top-down, bottom-up development, or inside-out development. The primary goal is to support continual evaluation and iteration during user interaction development, including much tighter, smaller loops of iteration than imagined in the spiral methodology (see next section). The Star life cycle minimized the number of ordering constraints among development activities. For example, developers did not have to specify all requirements before working on design. In fact, developers sometimes started by exploring design possibilities — perhaps by using a rapid prototyping tool and doing walk-throughs with clients and users — and, in the process, learned a lot about requirements.

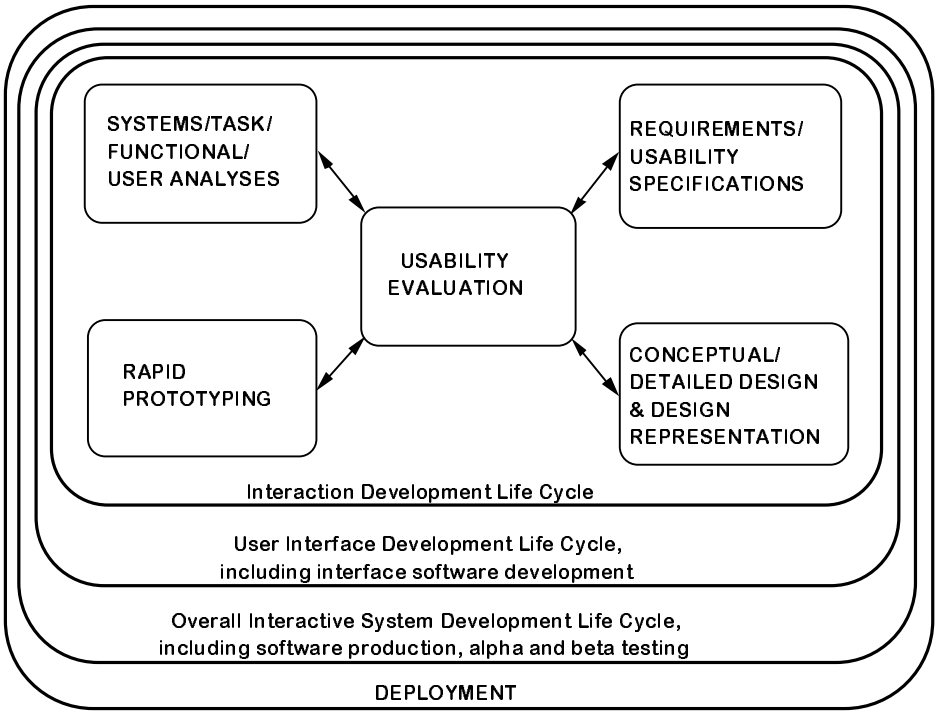

Figure 1: The Star Life Cycle Concept (adapted from Hix and Hartson, 1993)

The resulting evaluation-centered life cycle concept, shown here in Figure 1 [adapted from Hix & Hartson, 1993], was termed the *star life cycle*, because of its shape. The points of the star are not ordered or connected in a sequence. This means that a user interaction developer can theoretically start with almost any development activity and move on to almost any other one. The various activities are highly interconnected, however, through the usability evaluation process in the center; results of each activity are evaluated before going on to the next activity. In general, a different kind of evaluation is required after each different activity in this life cycle. Conventional life cycles lean toward independent performance of each development activity, whereas the star life cycle supports interdependent, but distinct, activities.

Evaluation centeredness is a very important property of the Star Life Cycle, which (LUCID/Star)*inherits. In (LUCID/Star)*each product evolution goes through a cycle of four basic activity types derived from the LUCID framework. Cycle completion is dependent on the results of a final Evaluate activity and satisfaction of the cycle exit criterion. Most of the time these two things will be the same, meaning that the exit criterion is based on attaining some predetermined result from the final Evaluate activity. This implies that each evolution of the product could be evaluated possibly many times before moving into the analysis of the next evolution.

4.3. The Waterfall Model of Software Development

For decades much of the software industry was dominated by a mandate to use a waterfall-like development methodology, such as that shown in Figure 2.

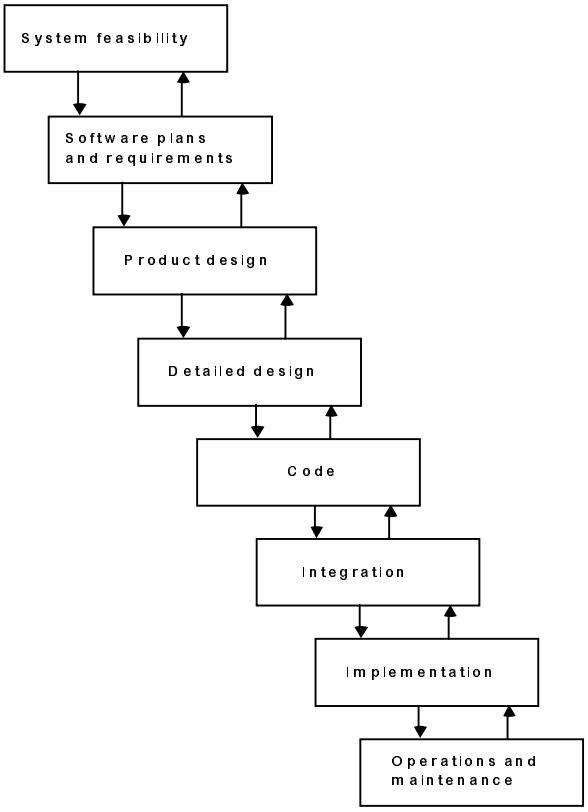

Figure 2: Representative of the Waterfall Model for Software Development

The waterfall model is characterized by a more or less rigid progression of stages, each of which entails a relatively independent development activity. Feedback among stages is limited, restricting the flow of information between non-adjacent stages and allowing major design flaws to filter through the first few stages without detection, only to be discovered too late in the overall process.

Being document- and schedule-driven, rather than driven by goals and accomplishments, the waterfall method is inflexible, but has been perceived as easy to manage because the predetermined schedule made it clear when to move from one phase to the next, regardless of whether goals were all met. Any remaining problems are bundled up in the documentation and "thrown over the wall" to the group in charge of the next phase. The most difficult problems can get passed across several phases until someone realizes that it is too late in the process to solve the problem properly. If the problem is fundamental (e.g., a problem with software architecture) and a solution is essential, the process might have to stop and restart at an earlier phase. At that point schedule slippage and cost overruns are inevitable and the probability of project failure begins to climb.

The waterfall model also had significant effects on organization management, effects that persist to this day in some organizations. Often different teams with different skills and experience, and sometimes with different management, were responsible for different phases of development. As these organizations matured, their management structure

often mirrored the compartmentalized nature of the waterfall phases. The requirements department and its management were separate from the design group, which was separate from the software production group. This rigidly perpendicular organization left a legacy of vertical "stovepipe" management that became an impediment to later interdisciplinary development teams. We have worked with such cross-jurisdiction teams who could not make significant decisions until each team member had passed the question up his or her respective management chain and a decision was passed down.

Although the Waterfall model has endured much criticism in the past, it does recognize one fundamental truth that can be applied to interaction design. That truth is the necessity of “evolutionary drift”. Evolutionary drift is the term we use to indicate that a development process, even an iterative process for a user interaction design or for software, is not *all* iteration. The term evolutionary drift describes an overall positive movement, in addition to iterative cycling, of the product through a series of evolutionary forms toward (and beyond) a final releasable version. Without this evolutionary drift, the development process never reaches its final goal of producing a product.

4.4. The Spiral Model of Software Development

Boehm defined a process model known as the Spiral Model of software development [Boehm, 1988], hoping to create a model that could overcome the shortcomings of a number of pre-existing software models, of which the waterfall model [Royce, 1987] was the best known and most widely used. Boehm's spiral model, shown in Figure 3, was more iterative and more flexible.

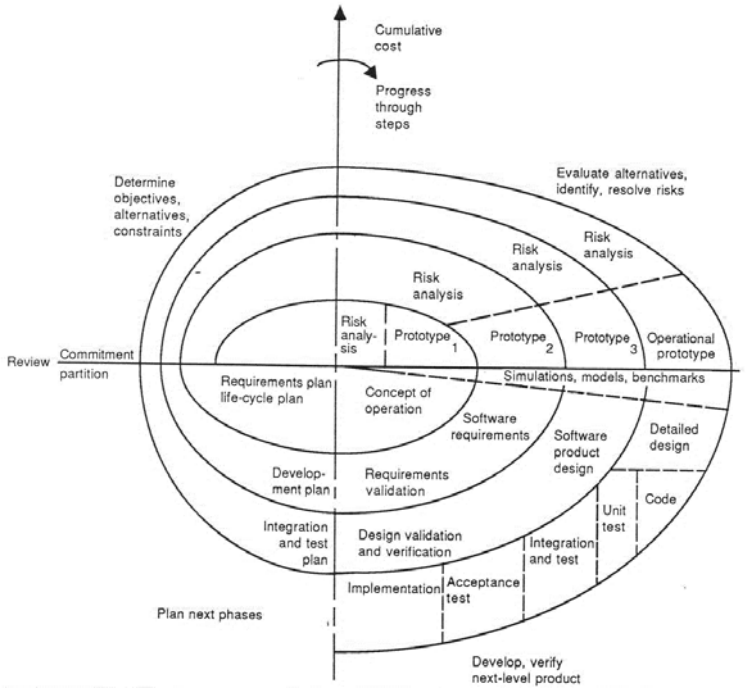

Figure 3: Spiral Model for Software Development [Boehm, 1988]

The model reflects the underlying concept that each iteration involves a progression of similar steps or phases to reach each new level of elaboration. Each time through the

spiral begins with considering the objectives of the current project iteration, determining alternative techniques of development, and recognizing constraints on these alternatives. The structuring of iteration in (LUCID/Star)*is primarily adapted from Boehm's Spiral Model and uses four basic activities to drive the development cycle. These four basic activities are repeated for each pass through the spiral, being applied to an increasingly complex prototype of the system. LUCID/Star*'s activity types are derived from the LUCID framework. Each product evolution revolves through a cycle of the four activities one or more times before becoming detailed design requirements in the first activity of the next cycle.

*4.5. Moving Toward LUCID/Star**

Each of the four methodologies discussed above contributed to the development of (LUCID/Star)* in the following ways:

- The LUCID Framework provided the four basic activity types for interaction design: Analyze, Design, Implement, and Evaluate
- The Star Life Cycle grounded our work in evaluation centeredness, both at the overall process level and at the product evolution level
- The Waterfall model made explicit the evolutionary drift that a process must facilitate in order to progress from concept to product
- The Spiral model offered iteration and repetition of basic design tasks

Based on the contributions of these four methodologies, we began formulating a process model that would become (LUCID/Star)*. We have found that a sequence of cycles is a more effective analogy than the spiral model for the interaction development process. A sequence of cycles is more modular and makes it easier to focus on each cycle separately. In that sense, (LUCID/Star)*is like a "spiral unwound". That analogy can be visualized by imagining the spiral of Figure 4, abstracted from Boehm's spiral model in Figure 3, as being made of stiff wire.

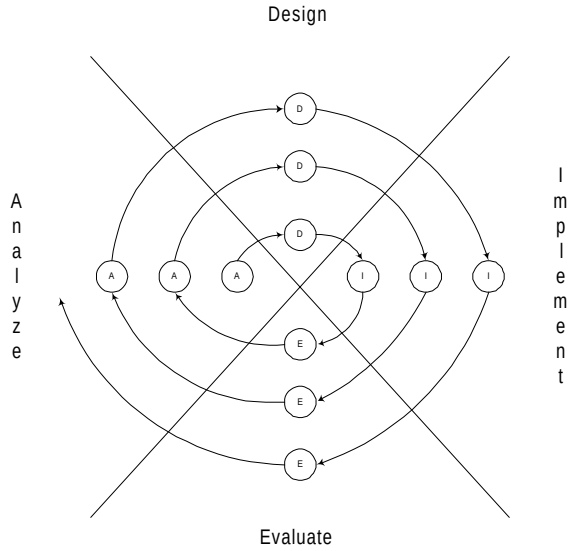

Figure 4: Spiral model abstracted from Figure 3

If one pulls the two ends of the wire apart, the spiral becomes a helix, as in Figure 5.

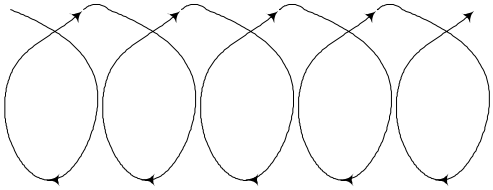

Figure 5: Spiral unwound

If one were then to 'solder' the wire to itself at each place where it crosses over itself in the helix, round the loops a bit, pull it out a bit more, and the result is a connected sequence of circles, as in Figure 6.

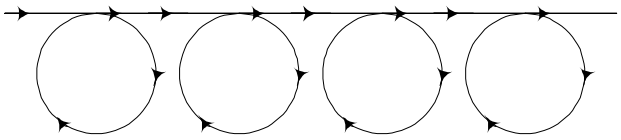

Figure 6: Spiral unwound and stretched out

This representation quickly evolved into,

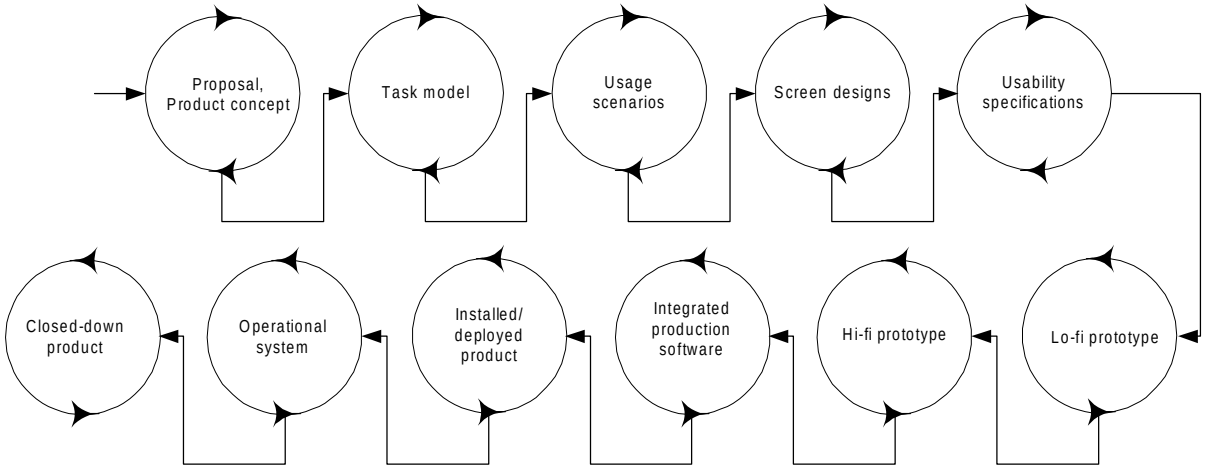

Figure 7: The Waterfall with Whirlpools

Originally we had settled on the representation in Figure 7 to visualize our model, where each circle was a (LUCID/Star)*cycle and the arrows connecting cycles show the evolutionary drift of the product from cycle to cycle. However, this visualization was perceived by some as resembling a "waterfall with whirlpools", which was too linear to truly represent the essence of (LUCID/Star)*. We also realized that we had omitted a critical piece of the model from our diagram (something also missing from LUCID, the waterfall, and the spiral model): an explicitly represented ability to iterate *among* cycles as well as within them, something we originally had in the Star Life Cycle concept. Figure 8 gives a more accurate picture of the (LUCID/Star)* model, explicitly representing iteration among cycles through a central "hub". Any cycle can flow back to any previous cycle as well as progress to the next cycle in the evolution.

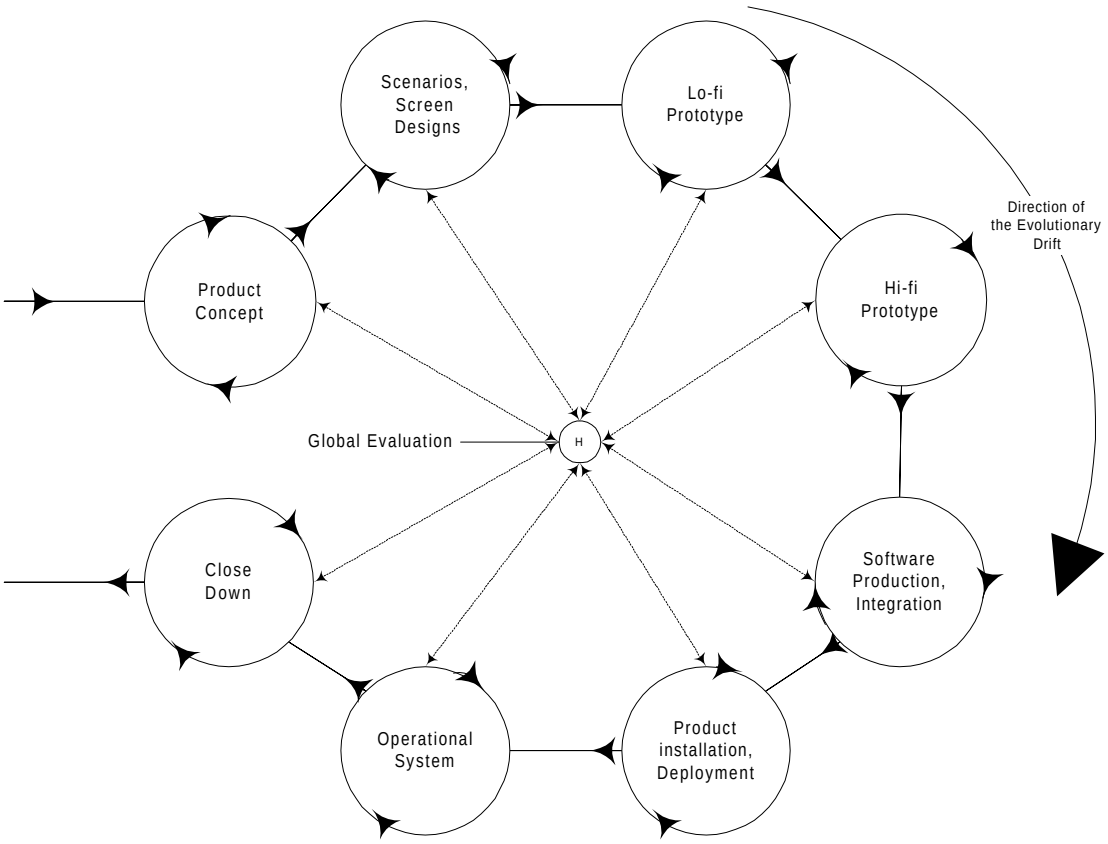

Figure 8: The (LUCID/Star)*structure

The configuration of Figure 8 helps to explicitly show two important aspects of the (LUCID/Star)* process model:

- The ability to iterate back to any previous cycle through the central hub (inherited from the Star Life Cycle.)
- The evolutionary drift toward completeness that the product undergoes (inherited from the Waterfall Model.)

Each circle represents a process cycle containing LUCID "stages" as development activities. The advantages over the spiral model are:

- Each cycle in the process can be distinguished,
- Each cycle can be named for the product it produces,
- Each cycle can be instantiated differently with specific activities and techniques, and
- Each cycle can be iterated independently of the others.

In addition, the new (LUCID/Star)* configuration offers:

- An explicit ability to iterate back to previous cycles using a global evaluation criterion,
- An explicit evolutionary drift to increasingly well-developed product forms

Taken together, the cycles offer a framework for:

- Guiding practitioners and development teams through the process
- Making choices for all activities, techniques, tools and templates, and deliverable work-products, and
- Structuring documentation and communication, plus planning and scheduling of iterations, cycles, and activities throughout the project life cycle.

The modularity of (LUCID/Star)*supports customization; specific methods and techniques already familiar can be "plugged" into activity types. Levels of abstraction make it easy to comprehend; developers can understand each cycle at a high level, then string several together to get a whole process. Deliverables fall out naturally as explicit work-products of the activities.

5. Summary of (LUCID/Star)*

(LUCID/Star)*is a product-oriented process model to guide usability engineering practitioners through activities for developing user interaction designs with measurably high usability, implemented as user interfaces within interactive computer systems. As a process model, it is not a development methodology per se. Many development groups have their own methods and techniques for various parts of the process but need suggestions for other parts and need a way to structure the process cost effectively. Most development groups also need to be able to tailor the process scope and activities to meet the needs and constraints of each individual project. Project groups also need templates and tools to help produce effective documentation of the process and its work-products. But developers cannot afford the necessity to adapt to an entirely new "methodology" in the process, losing the value of their own established tools and techniques and their familiarity and experience with their existing methods. Thus, the (LUCID/Star)* framework can be integrated with various software engineering methodologies or, for small product development efforts, can be used as a stand-alone development method.

We show that (LUCID/Star)* meets these needs as a generic framework that usability practitioners can tailor into a specific process by plugging in methods and techniques most appropriate for situations in their own development environments. (LUCID/Star)*has evolved into a framework to manage the process of designing a user interface in a way that incorporates the best industry practices of user-centered design and usability engineering. The (LUCID/Star)*framework guides the user interaction development process, its planning, its scheduling, and its documentation.

(LUCID/Star)*goals are to:

- Provide UI designers with a framework within which to apply the best practices,

- Allow for seamless integration of design and usability activities with software development methodologies,
- Support a user-centered approach to interface design, and
- Ensure a measurably high level of usability in the finished product.

(LUCID/Star)*has evolved from a combination of several existing usability engineering frameworks and life cycle concepts, indicating the major sources of its development, the Cognetics LUCID (Logical User Centered Interaction Design) development framework [Cognetics, 2000] and the Virginia Tech Star life cycle concept [Hix and Hartson, 1993]. The "*" in the name is a lexical pun[1] on the Kleene iterative closure operation in regular expressions and means, "to iterate as needed". (LUCID/Star)* addresses development activities, cycles and iteration, documentation, documentation templates and support tools, deliverables, and an evolving materialization of the product. The final product evolves as it passes through each cycle of Figure 8, each of which is an iteration of four kinds of LUCID interaction development activities: analyze, design, implement, and evaluate, as shown in Figure 9. The (LUCID/Star)*process model offers a flexible structuring of the development stages, their ordering, and their transition criteria (completion criteria for one stage and entrance criteria for the next) [Boehm, 1988]. (LUCID/Star)*also provides support for planning, scheduling and connections to software development of the non-user interface component of a project.

5.1. Cycles

Figure 8 shows an example (LUCID/Star)*user interaction development process as a sequence of iterative cycles, each cycle producing, and named for, a different form of the same final product. Developers iterate within each cycle until its exit criteria are met. For example, iteration might continue for a high-fidelity prototype until lab-based usability testing produces user performance results that meet or exceed the requirements stated in the usability specifications. Other, similar and parallel, sequences of cycles represent processes for developing concomitant products such as user documentation and training.

Although the "cradle-to-grave" scope shown also embraces software production and use, the focus of (LUCID/Star)*remains on usability across this broad product life cycle.

5.2. Activity Types and Activities

Each cycle of (LUCID/Star)*is an evaluation-centered iteration of four types of development activity, as shown in Figure 9:

- Analysis
- Design
- Implement
- Evaluate

[1] Our thanks to George Casaday.

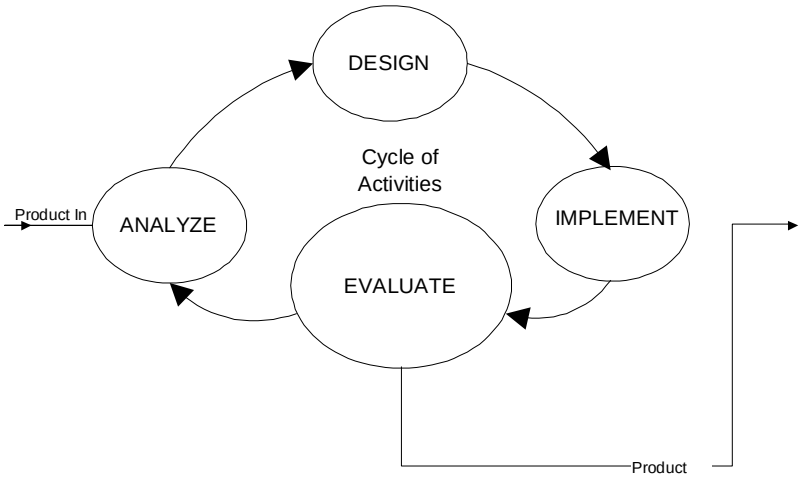

Figure 9: Development activity types within each (LUCID/Star)*cycle

Analyze, Design, Implement, and Evaluate are activity *types* in each (LUCID/Star)*cycle, each of which gets instantiated in each iteration of each cycle by an activity instance and a technique for performing that activity. Thus, each cycle can potentially include several kinds of activities of the same type. For example, Evaluate activities of the Usage Scenarios and Design Sketches cycle can include scenario task coverage, design sketch completeness, and scenario accuracy evaluation – each being done by a different technique.

Table 2: Possible Evaluate Activities and Techniques of the Usage Scenarios and Design Sketches Cycle

Cycle	Activity type	Activity Instance	Technique
Usage Scenarios and Design Sketches	Evaluate	Scenario Task Coverage	Usability Inspection via Comparison to Task Analysis
Usage Scenarios and Design Sketches	Evaluate	Design Sketch Completeness	Design Walkthroughs
Usage Scenarios and Design Sketches	Evaluate	Scenario Accuracy	Examination by Domain Expert

Documentation is associated with a specific activity type and activity (e.g., Analyze, task analysis) but not with a particular cycle. Thus, more than one cycle can contribute to the results of task analysis. As a result, Analyze type activities in the earliest cycles create initial documentation, while Analyze type activities in successive cycles add to and modify these documents as they evolve and mature throughout the whole process. The implication is that no activity is done entirely in one place in the process. For example, task analysis is mostly done in the Analyze activity of the early cycles. But later cycles continue to impact and inform task analysis. The creation of design scenarios, for

example, can reveal missing tasks and can lead to changes in task structure. Figure 10 shows how documentation is tied to an activity and flows between activities via the common documentation database.

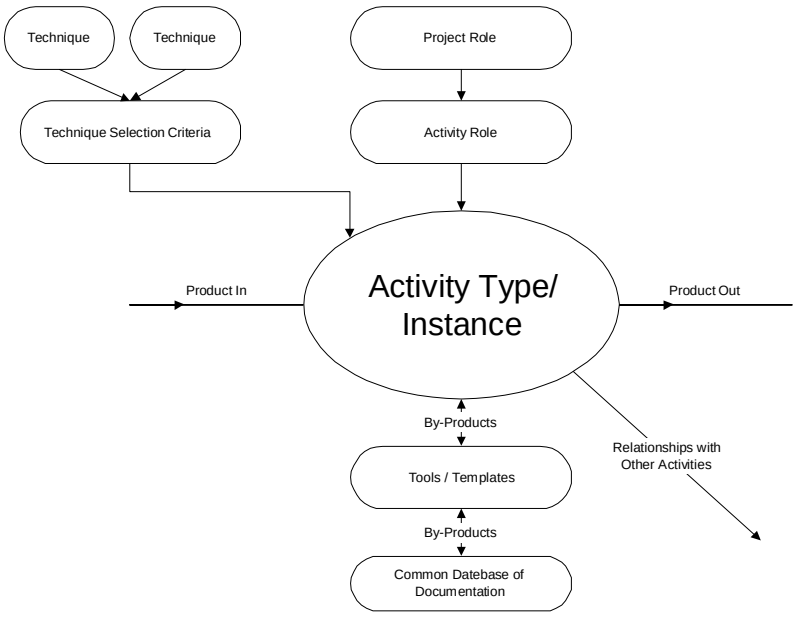

Figure 10: A generic activity

As a process model, (LUCID/Star)* provides an organizing framework for the activities and cycles. The way that the (LUCID/Star)* process model becomes a development process for a particular project lies in the way specific development activities are selected to instantiate each development activity type in each cycle. The choices for activity instantiation, from (LUCID/Star)* methods and from the developers own methods, depend on project size and goals and available usability engineering techniques appropriate to the cycle's stage of development and project team expertise and experience. The same activity type can be instantiated with a different HCI technique in successive iterations of the same cycle.

5.3. Process Instantiation

Processes are created by instantiating the process model, an exercise that that occurs in two phases.

- First, implementers of the process must select which product evolutions (cycles) they wish to develop for this product.
- Second, implementers must populate the activities of each cycle by identifying appropriate techniques for each activity.

Product evolution and technique selection are heavily influenced by constraints such as interaction style/application type, budget, schedule, etc. The amount of product evolutions and the complexity and expense of the techniques can be customized by the implementers of the process model.

5.4. The nature of iteration

The idea of iterative refinement of interaction designs is now commonly accepted. However, many people, including usability practitioners, think of iteration in terms of the process as a single iterative cycle. Although iteration *among* cycles does occurs as part of the process, the most common iteration is a more localized iteration *within* a cycle. For example, developers might go through the Analyze, Design, Implement, and Evaluate activities for the screen design cycle several times before moving on to the low-fidelity prototype cycle.

Each time developers iterate through a given cycle, they might apply a different technique for a given activity type (e.g., design). For example, in the screen design cycle an early iteration might call for the use of scenarios to identify screen objects and user actions in screen design. A later iteration of the same screen cycle might call for participatory design techniques like those discussed in Muller (1991) to refine the screen layout. Although product development can usually never be considered truly complete, neither does the usability engineer seek perfection in any iteration – only a cost-effective target level of usability.

Users of the process model define the exit criteria for each cycle. Usually a cycle exit criterion is based, in turn, on some combination of two types of other criteria: evaluation based criteria and constraint based criteria. Evaluation based criteria are determined as achieving some level of measurable usability on the current product evolution, whereas constraint based criteria are based on outside influences such as schedule, budget, and management style. Combining these allows practitioners to attain a cost-effective level of usability within the constraints of the project and, hopefully, meeting the project schedule, without falling into indefinite iteration.

Once a cycle's exit criteria are met, the product evolution goes through one more 'global evaluation' step (represented by the 'hub' in Figure 8) before moving on to the next cycle. The Evaluate activity within a cycle results in a measure of usability taken locally within the current stage of product evolution. It ensures that we are building the right product. On the other hand, the hub is an evaluation activity that has a more global view and asks whether the state of the product is such that it is ready to evolve to the next cycle (the next product form). If not, the product may be sent back to a previous cycle because of something not working out well or something discovered in the process that indicates the need for repeating a previous cycle or cycles. Here, the hub helps us know if we are building the product right. This property of the model is inherited from the Star Life cycle, since that model had a global evaluation activity as a hub as well.

5.5. Basic Principles

(LUCID/Star)*is based on a few simple principles that govern how it works and that help its users think about how to apply it.

5.5.1. Process is product-oriented

The objective of the whole process is to produce an interaction design with provably high usability upon which the user interface part of an interactive software system is built. Thus, the interaction design is the primary product of the (LUCID/Star)*process model.

However, the full breadth of the (LUCID/Star)*scope includes incarnations of the product (the interaction design) as manifest in the user interface implemented in the production software of a product release and beyond deployment into the field. The objective of each cycle is to produce a different form of that product, each cycle being named for the form of the product that it produces (e.g., low-fidelity prototype). (LUCID/Star)* is intended to be accompanied by other, similar and parallel sequences of cycles for developing concomitant products such as user documentation and training and even product packaging and installation procedures. These other cycles have essentially the same inputs and relationships and development activities such as needs analysis, requirements, user class definitions, prototypes, usability testing, and iteration, but somewhat different kinds of products.

Each activity in each cycle can also produce work-products, usually design documentation and other deliverables such as documentation of task analysis, requirements, and user class definitions.

5.5.2. Products evolve through cycles

As development progresses through the sequence of (LUCID/Star)* cycles, each successive cycle produces a more evolved form of the product (e.g. from high concept statement to scenarios to screen designs to low-fidelity prototype to high-fidelity prototype to product release). Rather than thinking of these different forms as separate entities, we think of them as different embodiments, incarnations, stages of development, or forms of the same product, the user interaction design, as it evolves through the process cycles of LUCID/Star*. *For simplicity we use the term "product" to refer to any of these forms or stages.* Thus, a scenario is the product, as it emerges from that particular cycle. Similarly, a prototype is the product while working in the corresponding cycle for that stage of development. After software production, the interaction design product becomes embodied in the user interface part of the application software and gets integrated with the non-user interface software to form the deployed target application system.

The forward path of evolution is explicit in the process model. Backward paths are implicitly allowed whenever evaluation results of any one cycle indicate that the product can benefit from revisiting the activities of any previous cycle.

5.5.3. Each cycle is iterative

In general, the process will iterate within each cycle. It is possible for any activity to be given a different instantiation by applying different techniques each time through the same cycle. The process moves from the current cycle to the next when the current cycle exit criterion (usually specified in terms of usability evaluation results) is met. For example, the high fidelity prototype can be considered complete when it meets the target levels set for usability specifications.

5.5.4. Each cycle contains same types of activities

Activities are the basic building blocks of a (LUCID/Star)* cycle. Each cycle contains the same activity *types*, inherited from the original LUCID: Analyze (including envisioning),

Design (including redesign), Implement (including prototype), Evaluate (including refine).

Many linear development methodologies such as the waterfall method contain activities for analysis, design, implementation, and evaluation, but they appear at too high a level. We now know that every form of the product and work products (such as requirements, prototypes, documentation, and training) each needs iteration of all the activity types. So we replaced phases with cycles at the highest level of the process model and put the Analyze, Design, Implementation, and Evaluate activity types in each cycle. Thus, unlike the waterfall model, (LUCID/Star)*iterated through these activities in ever cycle.

5.5.5. Each cycle is evaluation-centered

The Evaluate activity is the center and controlling activity of each cycle. The Evaluate activity is the key to process management. These ideas are derived from the Star Life Cycle of interface development. Each (LUCID/Star)* cycle represents an evolution of the product (e.g. product concept, low fidelity prototype, etc.) and often a cycle's exit criteria is based on achieving some high degree of completeness and/or usability for that evolution. The purpose of the Evaluate activity is to measure the completeness and/or usability of the current product evolution. For example, usage scenarios can be evaluated based on the breadth of coverage (as compared to the task analysis) and accuracy of the narrative (as judged by a domain expert). If the product evolution does not stand up to the appropriate measure, then the cycle will iterate again to improve the design. If the product exhibits an appropriate level of completeness and/or usability (as defined for that specific evolution of the product) then the design evolution may continue on to the next cycle.

5.5.6. Work products (documentation) evolve over cycles

Work-products are documentation deliverables that are produced by various activities within a cycle and are held in a common database for use as they evolve in the same and later cycles. Documentation is usually associated with a specific activity type (e.g., Analyze, Design) but not with a particular cycle. Thus, the Analyze activities of any given cycle each contribute to the evolution of the analysis documents. As a result, Analyze type activities in the earliest cycles create the initial version of many essential work products that make up documentation we call the UI roadmap (e.g. high concept statement, business process model, needs analysis, user analysis, task analysis, functional analysis, task / function allocation, usability goals, constraints, and requirements). Analyze activities in successive cycles add to and modify these documents as they evolve and mature throughout the whole process. No work product is developed entirely in one cycle of the process. For example, much task analysis is done in the Analyze activity of the early cycles. But later cycles continue to impact and inform task analysis. The creation of design scenarios, for example, can reveal missing tasks and can lead to changes in task structure.

5.5.7. Process can be viewed at different levels

Once developers have determined how they want to tailor their instance of the process model for their specific project, they can view the entire process at multiple levels and from multiple perspectives. For example, at a given time one can look at the various

cycles without concern for the detailed activities within a cycle, one can focus on inter-cycle relationships, or one can narrow the view to just the documentation for task analysis.

Work is underway on providing a process modeling tool to support developers in creating and editing the process instance for a project in addition to navigating through it and viewing it at various levels of abstraction: process view, cycle view, activity view, or documentation view.

5.5.8. Any part of the process is an instance of what is possible

The (LUCID/Star)*process model is a guiding framework, not a methodology, because, although it offers well-tested and successful techniques for each activity in many cycles, it does not rigidly request specific techniques for any activity type in any cycle. Any specific version of the process, a cycle, or an activity is an *instance* of that part of the process model and is therefore only an example of what that part of the process can be for a given project at a given time. Thus, flexibility and tailorability are keystone features of the process model. A process can be instantiated from the model in two ways: by deciding which cycles and activities to include and by deciding which techniques and methods to use. Developers can and should include or omit whatever activities or cycles their scope, team composition, experience, project management style, budget, and/or time schedule indicate. Developers should include their favorite usability engineering methods and techniques where appropriate. The process can be different every time and still follow the (LUCID/Star)*process model. There is no cut-and-dried formula or recipe for configuring a custom process. This is an engineering activity and, as usual in such cases, experience is the best way to learn and improve. The (LUCID/Star)*process modeling tool will facilitate specification and editing of each custom instance of the process.

Because users are at the heart of LUCID/Star*, user-centered techniques are prominent throughout the interaction development life cycle. For example, Analyze activities in early cycles should include target user class definitions and user task analysis. A design team may select one or more techniques, such as structured client and user interviews, from the techniques available for studying user needs, client bus process, and workflow for the Analyze activity of the product concept cycle. The team might choose design walkthroughs for the Evaluate activity in the scenario and screen design cycles, but lab-based usability testing for Evaluate in the high-fidelity prototype cycle.

Usability specifications are often developed in the Analyze activity of the low-fidelity prototype cycle, but can be developed in the high-fidelity prototype cycle instead, or in addition. Usability inspection techniques or design walkthroughs are appropriate Evaluate activity techniques for scenarios and screen designs or low-fidelity prototypes. Similarly, lab-based usability testing can be applied as an Evaluation activity technique to either low-fidelity or high-fidelity prototypes. A small project might have only a low-fidelity prototype or only a high-fidelity prototype. Benchmark tasks can be used for usability testing either a low- or high- fidelity prototype. Thus, for developing a web-based application, for example, on a very short schedule (Internet time!), the process can be very light on its feet.

In a large or important project where more resources are available and more is at stake in the resulting usability, the team may plan to make both low-fidelity and high-fidelity prototypes and to do lab-based usability testing with both. It would be typical in that case to do less formal usability testing with the low-fidelity prototype, probably taking only qualitative data to eliminate as many usability problems as possible before committing to implementing a high-fidelity prototype. Then more formal usability testing would be done on the high-fidelity prototype, to include quantitative data, to assess whether and when usability specifications are met.

The development team or the project manager would use the (LUCID/Star)*Process Modeling Tool to produce a process specification, which defines the process model instance for a project and which results in meta-documentation, which is one kind of work-product.

5.5.9. A user interface development process must be integrated with the software engineering development process and not as an add-on

(LUCID/Star)*is for development of user interaction designs, but the overall development of an interactive software system involves software development, too – for both the user interface software and the non-user interface software (the functional core of the application).

Although we believe that the user interaction design should be developed separately from consideration of software, we also believe that any user interface development process must be integrated with the software engineering development process and not treated as a separate add-on process.

- A more integrated process will produce a more integrated product.
- Both development worlds have to have the same vision of the overall system concept, goals, and architecture.
- Surely it is more cost-effective for the two development worlds, for user interaction development and for non-UI software development, to share many common activities (e.g., needs analysis, requirements gathering, prototypes for usability and software concept proving) rather than performing these activities separately and redundantly.

Despite the obviousness of the above claims for an integrated overall development process, too often usability engineers developing the interaction design for specific system functionality and the software engineers developing the non-user interface part of that functionality do not communicate adequately. Yet they are working on different aspects of the same thing!

Given that software engineering processes are usually more well established within development organizations, a successful candidate for a usability engineering process must be connectable and integratable with most *existing* software engineering methodologies/frameworks. Taken to its logical end, this course of thinking probably means that a project should not have two parallel processes, but one single, integrated process and one team, even though the two parts might often have different activities, goals, and work products.

5.6. Using the Model for Scheduling and Planning

In brief and simple terms, an instance process of (LUCID/Star)*is a blueprint to guide developers through the user interaction development process. It indicates what activities to do, in what order, and how long to do each activity, and what products and work-products are produced.

In addition to guiding interaction development, (LUCID/Star)*also supports project planning and scheduling, where the cycles and activities become nodes in a PERT or Gantt chart. Projected completion dates can be associated with activities and cycles. The planning chart for interaction development can be combined with that of non-user interface software development activities to help synchronize those two development domains.

In addition to the process modeling tool, (LUCID/Star)*will provide a project planning and scheduling tool for creating, editing, and viewing a visual plan and schedule for process cycles and activities at various levels of abstraction. Initial designs for this tool are underway now at the Usability Methods Research Laboratory at Virginia Tech.

6. The Field Study

James Helms and Dr. H. Rex Hartson of Virginia Tech undertook this research in conjunction with Optim Systems, Inc of Fall's Church, VA. The project was sponsored by the Center for Innovative Technology (CIT). The project consisted of assembling a usability team to help Optim develop a user interface for a new web-based device management application. The opportunity provided us with the chance to instantiate the new (LUCID/Star)*process model on a more traditional interface. However, in addition to the web-interface, Optim also hoped to use the process model to build a wireless web application for cell phones and PDA's. The mixture of traditional and non-traditional interaction design allows the researchers to test the (LUCID/Star)*process model in two very different modes.

In addition to the better known advantages of using such a process to design interfaces, Optim hoped to acquire a process for achieving high usability in their products and to instill usability as a core competency of the company, thereby improving their future efforts in interface development. (LUCID/Star)*is well suited to Optim's diverse design needs, being easily customized for the development of traditional and non-traditional interfaces.

The author, in collaboration with H. Rex Hartson, developed the current, initial version of the (LUCID/Star)*process model described above. After completing the development of the process model, I oversaw the instantiation of the model and its application as fieldwork. Work on instantiating the process model at Optim began in January of the year 2000, when personnel from Virginia Tech and Optim Systems met several times to determine interest in the process and to discuss Optim's expectations and needs. After attending a short seminar on usability and (LUCID/Star)*the Optim management was very interested in obtaining a process for their company.

Upon receiving positive indication of interest from Optim, we assembled a team of three interns to act as liaison between personnel in Falls Church and personnel in Blacksburg. The interns took responsibility for implementing the techniques chosen by the researchers in Blacksburg. Techniques were chosen to display the way in which the (LUCID/Star)*process model could be adapted to the design of each individual interface.

During the instantiation of the model at Optim, I observed (LUCID/Star)*in action in order to address the questions and needs addressed in section 3. The process model was instantiated to design two interfaces for communicating with Internet connected devices: a web-based and handheld interface. The web interface progressed through early user profiles, usage scenarios, task analysis, screen sketches, and high fidelity prototype. The wireless interface was held up by technology constraints, but early analysis, task analysis, and usage scenarios were completed and documented.

The field study ended on October 31, 2000 when I presented a summary of the process instance and a guide for using the process model in the future to the staff at Optim.

7. Results

We divide our results into two sections: the goal-related results about the process model and those observed about other aspects of the usability effort.

7.1. Goal-related Results

7.1.1. Flexibility of (LUCID/Star)*as a non-linear, iterative progression through the development cycle

Optim Systems, Inc. is a company on the edge of their technology. The domain of remote device management is new and quickly changing. Success in the domain requires not only technical know-how but creative license as well. Creativity was especially required in the user interface, which Optim was developing without the aid of precedent. Although Optim personnel had no dedicated usability specialist, the upper management was familiar with usability practices and goals. They were convinced that instilling usability as a core corporate competency would increase their competitive edge and further sales of their product. They wanted a structure that would allow them to develop and refine their design in the most cost-effective manner, while preserving flexibility and allowing them make creative design and re-design decisions. They felt that a traditional, rigid structure would 'pigeon hole' the design too early and not leave room for creative iteration. (LUCID/Star)* fulfilled the need for a flexible, non-linear, and iterative process model while allowing creativity in the design process.

Devising a situation in a study to demonstrate non-linearity in the model is difficult since it implies a, possibly unexpected, decision to jump forward or backward in the design development. As such, it is troublesome for one to plan non-linearity into a process instance or to predict how non-linearity will affect the overall outcome. Fortunately, at Optim, we had the opportunity to observe the need for non-linearity, as well as, the results within the process. We spent a lot of time working through and iterating the early design, so when we had finalized the design sketches and were ready to move on to the

low fidelity prototype, we realized that we had used more of the schedule than intended. To alleviate some of the time pressure we elected to skip the low-fidelity prototype and move straight into development of the high fidelity prototype. In a rigid process structure this move would have been impossible since each stage in such a structure is tied strongly to its predecessor. We encountered no difficulties in adjusting the (LUCID/Star)* process to accommodate the change. Cycles in (LUCID/Star)* are dedicated to developing one particular evolution of the product, which then feeds its completed product to the next cycle in the process. This product evolution serves as detailed design requirements for the next cycle. In this way, all the cycles are connected, but not utterly reliant on the ones before it, allowing for some limited forward jumping. Thus, in this case, instead of the high fidelity prototype cycle receiving a low fidelity prototype, it received design sketches. The low fidelity prototype cycle was simply lifted out of the process.

The ability to revisit and iterate among cycles is also part of our requirement for flexibility. It is often the case that analysis and evaluation of a later evolution of the product will reveal holes in the earlier analysis and require the earlier evolution to be reanalyzed and modified. A prime example of this, which occurred at Optim, is the connection between task analysis, usage scenarios, and design sketches. Although we had originally expected only one or two iterations here, over a period of thirty days we worked with the interns in Fall's Church to iterate these three product evolutions through nine cycles each. The task analysis would lead to the writing of usage scenarios, which would, in turn, lead to the drawing of design sketches. Walkthroughs performed on the design sketches would uncover tasks that had been omitted from the task analysis. Adding these discovered tasks to the task analysis would require the writing of new scenarios and so on. For example, establishing currency of work object (account, device, etc) became an issue for our design team. In the office, this is done by simply picking up the right file or focusing your attention on the appropriate papers. The action is so automatic that it becomes implicit to the task of managing the account, device, etc… For this reason, establishing currency was overlooked as a potential snagging point in the design and in effect caused us to have to revisit the design multiple times. This period of design, which was initially scheduled for approximately two weeks time stretched into a seven-week ordeal. Each time a new task was uncovered, we would revisit the task analysis to determine where that task belonged in the task hierarchy and how it affected the typical usage exemplified in the usage scenarios. This unexpected extra iteration among cycles was accommodated easily by the high-connectivity among cycles through the hub, as shown in Figure 8.

Several features of the process model made the process restructuring and iteration possible, namely the modularity of the model and the two-fold nature of iteration found in (LUCID/Star)*. Modularity refers to the ability to determine how many cycles the process will contain and what activities the cycles will contain. Since each cycle represents the development of one evolution of the product, implementers of the process can choose which evolutions the product will go through.

Iteration in (LUCID/Star)* stems from two sources: the structure of the cycles and the ability to revisit any given cycle based on the overall evaluation of the current product evolution within the scope of the overall project. Thus we have two forms of iteration in

(LUCID/Star)*, iteration within and among cycles. Iteration within cycles is based on cycle exit criteria, thus, if a cycle does not meet its exit criteria after its evaluate activity, then the cycle continues back to the analyze activity. If a cycle meets its exit criteria, but the results of the evaluation reflect a deficiency in a previous evolution (even with respect to changing overall criteria), then the users of the process can elect to return to the corresponding cycle and re-iterate the product evolution represented within. If significant changes are made, this may result in the iteration of several subsequent cycles as well.

7.1.2. Ability to accommodate existing methods and techniques for development activities

As stated earlier, Optim's management was familiar with some usability techniques and practices. Before the process model was instantiated and introduced through this research, they had already begun to piece together an ad hoc usability scheme for interaction design. Their scheme consisted of an eclectic mix of methods and techniques, some of which worked well and others which did not. In order to retain familiarity with, but apply structure to, their usability practices, Optim wanted a usability process into which they could adapt their existing methods into its structure and supply a bit more order to their initial efforts.

Optim had begun working with early design techniques in preparation for the work on their remote device management system. They had experimented with business process modeling, devised some bare-bones usage scenarios, and were investigating some usability testing techniques such as testing with portable usability lab equipment. Their initial efforts proved fruitful in that they were able to identify techniques that worked well and eliminate techniques that did not fit with their level of comfort and resources. They identified several techniques that they had grown used to and wished to incorporate into their new process. Among these techniques were business process modeling, writing usage scenarios, and internal design walkthroughs. Each of these techniques became part of the process instance created from the (LUCID/Star)* process model. Business process modeling became an Analyze activity in the product concept cycle. Usage scenarios became an evolution of the product, with an entire cycle devoted to their production. Internal design walkthroughs served as the Evaluate technique for both usage scenarios and design sketches.

Optim's usability methods were ad hoc and undeveloped, but (LUCID/Star)* proved itself capable of supporting almost any usability technique. The practitioner needs only to identify to what cycle (evolution of the product) the technique applies and what activity type the technique fulfills. For example, scenario analysis is an Analyze technique that can be applied in the Design Sketches cycle to glean UI elements from prewritten scenarios. Another example would be traditional lab-based usability testing, which can be applied to several Evaluate activities, including the Evaluate activity in the Low Fidelity Prototype, High Fidelity Prototype, or Operational Systems cycle. This "plug and play" capability stems from the model's ability to use multiple techniques to carry out the various activities within any cycle. This allows us to accommodate existing techniques by finding the most appropriate places for them in the overall structure.

Entire methodologies can be incorporated into an instance of (LUCID/Star)* in the same way. The methodology only needs to be decomposed into its constituent techniques

and/or design evolutions, and then ordered and integrated into the model. For instance, the LUCID Framework for interaction design [Cognetics, 2000] does not prescribe any particular techniques, but does suggest certain product evolutions that should be generated. A typical product designed with LUCID will result in the following product evolutions: high-concept statement, UI roadmap, user characterization matrix, hierarchical task analysis, usage scenarios, usability specifications, low fidelity prototype, high fidelity prototype, and the final UI. Each of these product evolutions can be given a cycle in an instance of the (LUCID/Star)* process model. Then, just as in LUCID, the practitioner can fill in techniques for the activities in the cycles to develop each of the product evolutions. This flexibility allows existing frameworks to be quickly and easily integrated into the (LUCID/Star)* structure, thus allowing practitioners to continue designing with techniques and methods they are familiar with.

7.1.3. Ability to customize the (LUCID/Star)*process model to meet the special needs of a given project environment

Optim Systems, Inc. is a start-up company striving for competitive edge and market share. As such they are a small company with a full-time roster of only nine employees. Obviously, each employee must be able to perform multiple tasks and duties. At the time of the research, no personnel could be spared to specialize in usability and thus a rigid, linear process that required a great deal of learning and structure could not be used. Since, in a start-up, resources and schedules are quickly changing, Optim required a customizable usability model that could be modified from project to project. Optim wanted this model to be able to adapt to their changing constraints both now and in the future as size and resources grew.

This aspect of the model was tested early at Optim, as we received an unexpected reduction of the research grant schedule and budget. The schedule was cut from 12 months to 6 months, and the budget suffered a similar decrease. This cut occurred after the research had already been planned and we were prepared to begin. We had already created an instance of the process model under the original constraints, but the new constraints forced us to re-structure our process, compelling us, in effect, to create a new instance of the model that fit the new constraints and resulted in an acceptable level of usability and completeness. The new process was necessarily abbreviated in time-scale and needed to operate with reduced resources and preexisting facilities since funds were no longer available to purchase new equipment or compensate user subjects. We were able to create such a process by reducing the number of product evolutions (and, in turn, cycles) and constraining the time for each cycle to fit within the overall project scope. This meant that our cycle exit criteria became based on both the schedule and the results of each cycle's evaluation activity. In the end, the process flexibility allowed project survival and even success through a firestorm of changing constraints.

Being a start-up company, Optim had limited facilities and resources, outside of the CIT grant, to devote to building a usability-engineering infrastructure. As such, they did not have room for a formal usability laboratory even if sufficient funds had been available. In addition to this, Optim had very limited access to their end-users, who were as a group, extremely busy and expensive. These constraints limited the techniques available to the usability effort at Optim and thus the situation called for great flexibility in tailoring the

process to the constraints. (LUCID/Star)* allowed the usability team to "plug-in" techniques that would fit the constraints. For instance, as we were nearing completion of the early design phase of the project, we had developed a rather comprehensive set of design sketches. Although not robust enough to represent a proper low fidelity prototype, these sketches were able to facilitate design walkthroughs. Ideally, these walkthroughs could have been performed by an end-user observed by a usability specialist, but the under-availability of end-users made this impossible. Instead, we used employees of Optim that were most familiar with the domain and with the expectations of the end-user. This activity gave us excellent feedback and resulted in a major iteration of the interaction design. By plugging-in this technique in place of walkthroughs by end-users, we were able to evaluate the design sketches and avoid major usability problems later on.

7.1.4. Applicability of (LUCID/Star)*process model to a variety of interaction styles and applications types

The term "variety of interaction styles and application types" applies more to Optim that most other companies. Optim's overall corporate goal was to create a remote device management product that could communicate with and diagnose a large variety of Internet-connected devices. Optim put no constraints on the device types the application would communicate with, and so the application would potentially have to communicate with a broad range of PC and non-PC devices. Each of these devices, their access ports, and functional characteristics had to be appropriately represented in the interface. In addition to this, Optim wanted to provide global access to their application via Web interfaces and wireless interfaces so that technicians could eliminate the requirement of having to directly connect to the device via a diagnostics port. Optim did not want to have to learn two separate methodologies for developing the wireless and the web-based interface, and thus required the process model to be applicable to both interaction styles. The resultant interface also had to appropriately represent the variety of devices with which the application communicated.

At Optim, we were able to instantiate (LUCID/Star)* for both the web and the wireless interface. Since the web interface was deemed most important to the business plan by the Optim management, we focussed our limited time and resources on developing it from concept to high fidelity prototype. Unfortunately, due to time constraints placed on us by the grant cutback, we were not able to see the wireless interface through to a robust prototype. We were, however, able to move through the early design phases of development, proceeding through scenarios and preliminary design sketches. Our observations of the process model in this case indicate that, given time to implement the process, the process model would have supported development of a highly usable wireless interface.

Our confidence stems from (LUCID/Star)* 's unique ability to be customized for any interaction style/application type at the process and technique level while leaving the local cycle structure intact. Interaction style/application type is a major factor in technique selection and defines product evolutionary stages. Thus, process structure (# of cycles, iterations) and techniques are affected by the interaction style/application type and adjusted by choosing appropriate product evolutions and techniques. On the other hand, local cycle structure is unaffected by interaction style/application type, meaning that the

basic activity types do not change within each cycle. This allows us to maintain the cycle structure and instantiate activities using any appropriate technique for that activity type, interaction style/application type, and product evolution. This combination of customizability and static cycle structure allows implementers to organize their process appropriately and maintain a consistent development scheme across interaction styles/application types.

7.1.5. Ability to include generation of documentation work products as an integral part of the process

Optim, like most companies, relies on a paper (or at least electronic documentation) trail to record and document their development efforts. This documentation supplements the design and records results of development activity. As the product evolves, the documentation must evolve with it, staying current with the design, so that developers can maintain a system of checks and balances, enforcing the results contained in the documentation to be reflected in the design. The end-result is a tangible record of the design activities that has evolved over the development cycle alongside the product. With this idea in mind, the Optim management wanted a usability process model that would provide for the integral generation and evolution of documentation throughout the development cycle. In (LUCID/Star)* we provided the documentation support they desired.

For Optim, documenting the process was especially important, as this instance of (LUCID/Star)* was their first real attempt at using a formal usability process. If successful, Optim wanted to be able to recreate the process in later projects, thus documenting development activities meant preserving the usability process they had used in the current project so they could apply them again in later projects. To this end, we generated 102 pages of process documentation, including project management decisions, early analysis activities, user analysis, task analysis, usage scenarios, error case analysis, usability specifications, scenario analysis, evaluation results, and plans for deployment and post deployment. This volume of documentation was one of the biggest resources for both the product and process developers that came out of the fieldwork. For the product developers, it was essential in ensuring that the same mistakes and false paths were avoided in the continuing design and other projects. For us, the process developers, it offered a wealth of insight into the performance of the process model instance.

In general, documentation is especially important in a usability engineering process since most evolutions of the product become detailed design requirements for the next; each product evolution can feed any later evolution. As such, when cycles are revisited the documentation itself evolves before arriving at a final acceptable state. This evolution drives the refinement of the documentation, which in turn leads to the refinement of the design.

Figure 10 above shows how an activity generates documentation, which then flows between activities via (LUCID/Star)* 's common documentation database.

7.1.6. Feasibility of supporting realistic connections to the software engineering process for developing the functional core

One of Optim's largest concerns with obtaining a usability process for their corporate culture and practice was the issue of compatibility with their software development methodology. Optim's Chief Technical Officer stayed very close to the usability effort to ensure that the (LUCID/Star)* process instance would integrate and communicate well with their methodology for software development. Their concern was that the addition of the usability life cycle would separate the user interface development too far from the core software development, leading to major challenges in trying to combine the two later in the overall project life cycle. For example, it was a requirement that data definitions reside in only one place within the overall system, namely as the data definitions of the backend database. Thus error checking in the user interface must be based on structured communication with the non-user-interface backend software. To be successful at Optim, (LUCID/Star)* had to integrate well with their software practices so that this kind of communication between the user interface and functional backend can be integrated early in the project life cycle and not treated as an add-on.

We were able to establish communicative connections early through the completion of comprehensive early design, including hierarchical task analysis, refined usage scenarios, user analysis, and design sketches. One such connection was through Error Case Analysis, which helped the backend designers realize what could go wrong as well as at what point in the interface these errors could occur. Once this was done, the interaction designers had to structure the UI in such a way that these errors would be minimized. In conjunction with this effort, the backend designers worked to prepare for every possible error. This proved to be a useful and necessary analysis in the design evolution.

Error Case Analysis proved important to the design effort, as one of the backend requirements was that all input formatting and data definitions be contained in one central database. Unlike usual front-end designs, Optim wanted to do no error checking at the interface, instead they chose to send all input to the database for error handling. This made establishing early connections with the backend designers essential, as they had to be aware of the types of user input and the places in the design where such would originate.

One of the factors that made establishing connections with Optim's software development methodology relatively simple was the access we had to the chief technical officer and the small size of the company in general. We realize this type of access and interaction is more the exception than the rule, and so to demonstrate (LUCID/Star)* 's ability to integrate with more rigid software development strategies, we show how an instance of the process model can be structured to work in conjunction with some of the well established methodologies.

Figure 11 shows an abstraction of the traditional waterfall model with the (LUCID/Star)* model overlaid to demonstrate how an instance of (LUCID/Star)* can integrate with this software methodology. Each box in the figure represents a stage in the development of the software system. Circles within those boxes are (LUCID/Star)* cycles integrated into each stage of the development. Thus in parallel with the development of software requirements, developers should create user and task analysis. The overlay shown

describes an instance of (LUCID/Star)* process model for developing a standard GUI, however, implementers of the model are free to define which product evolutions are represented by which cycles to facilitate the development of other systems.

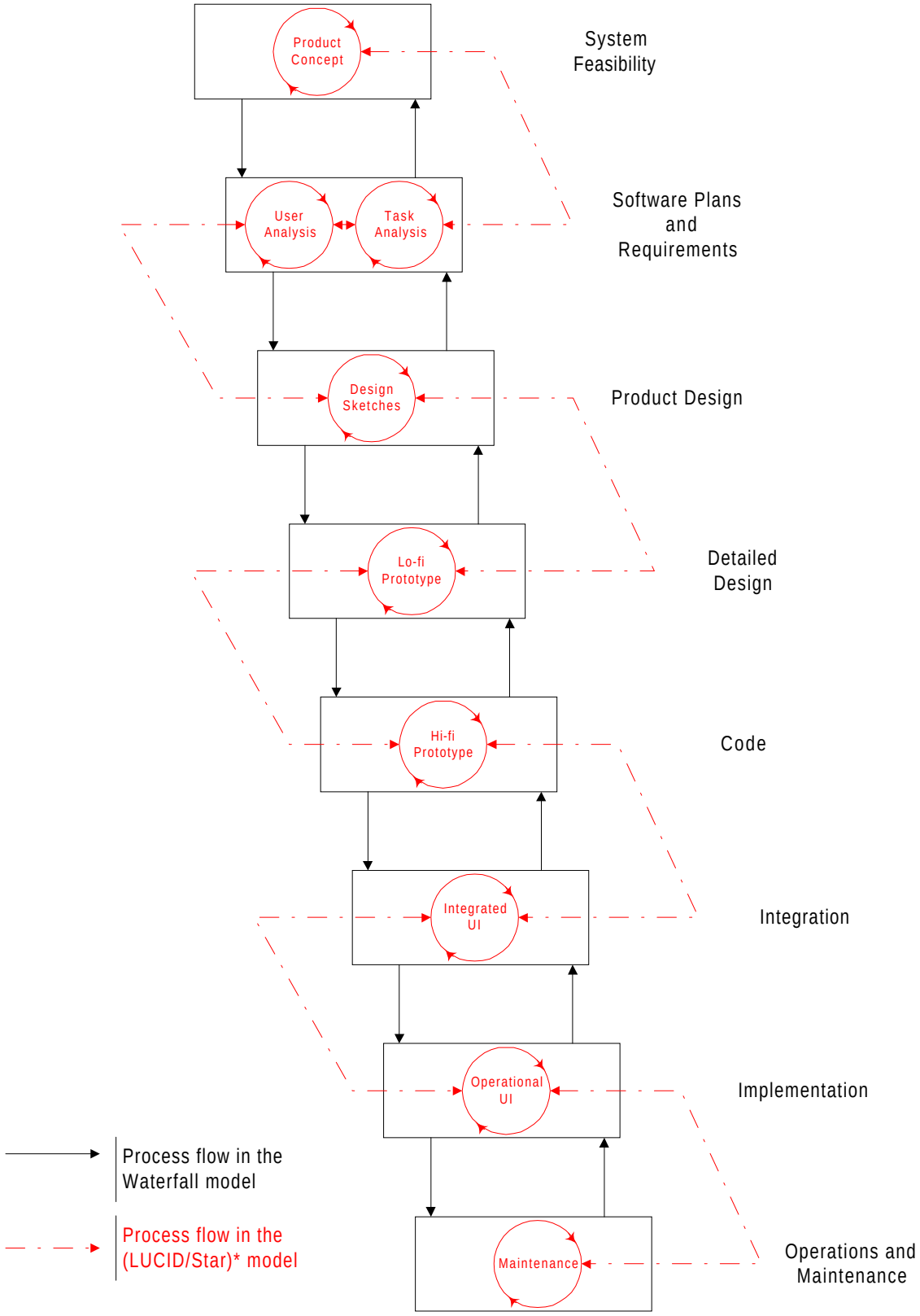

Figure 11: (LUCID/Star)* overlaid on the traditional Waterfall model

Similarly, Figure 12 shows the (LUCID/Star)* process model overlaid on an abstraction of the Spiral Model. The red lines represent progression through the (LUCID/Star)* structure and each quadrant of the spiral represents an (LUCID/Star)* activity type. Where the red and black lines cross indicates that while the software model may be progressing through one iteration of the Spiral, (LUCID/Star)* can iterate more than once. Circles along the primary axes represent commitment points in both the Spiral model and in (LUCID/Star)*. These points reflect the decision to proceed to the next activity type. The numbers labeling the red lines in the Evaluate quadrant represent the evolutionary drift of the interaction design.

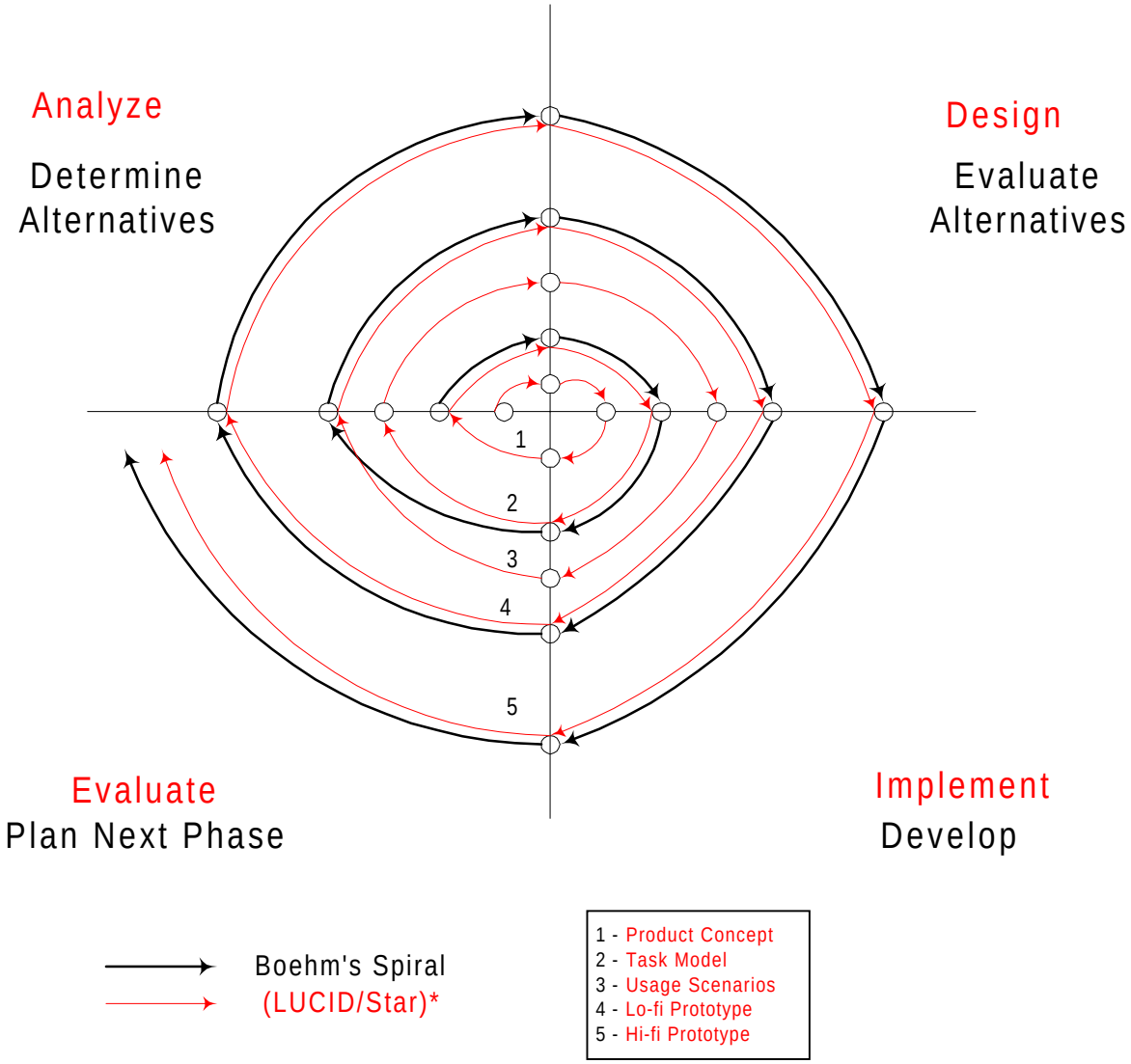

Figure 12: the (LUCID/Star)* process model overlaid on an abstraction of Boehm's Spiral

Finally, Figure 13 represents a more detailed instance of the process model running parallel to the improved Waterfall model [Virginia Tech CS Online SE Module, 2001]. Like (LUCID/Star)*, the improved Waterfall is product driven, with each stage in the product development (box in the figure) representing the creation of a new evolution of

the product. Also similar to (LUCID/Star)* is the improved Waterfall's iterative ability and verification step before leaving each stage. Each box in the figure is separated into two sections, one representing the stage of the overall software methodology and the other showing the (LUCID/Star)* component of that stage. Together, the two halves of the stages represent the integrated life cycle that encompasses both the back end and interface development. Thus, as the Communicated requirements are being gathered to begin design on the backend, interaction design is underway with defining the product concept and user analysis. As the backend design evolves into a requirements specification, interaction design becomes task analysis and usage scenarios. This process continues with each design evolving in tandem, communicating via documentation and the performing of activities that apply to both front and back end design.

As mentioned above, each stage in the software process ends with a Validation and Verification (V & V) step which determines if the product it ready to move into the next stage. "Global Evaluation" is part of this Validation and Verification step, which represents and functions as the 'hub' in Figure 8 above. The arrows radiating from the right side of each box show iteration in this version of the Waterfall. Backend or interaction design evolutions can be iterated as needed. In this diagram, cycles were assigned to stages of the Waterfall based on an "equality of product" ideal. "Equality of product" means that combinations of cycles were applied to stages if the combined usability products represented what usability engineers needed to gain a similar level of completeness to that stage of the software methodology. Once again, the representation shown here would be most appropriate for the development of a standard GUI, but could be easily modified to represent the development of any interaction style.

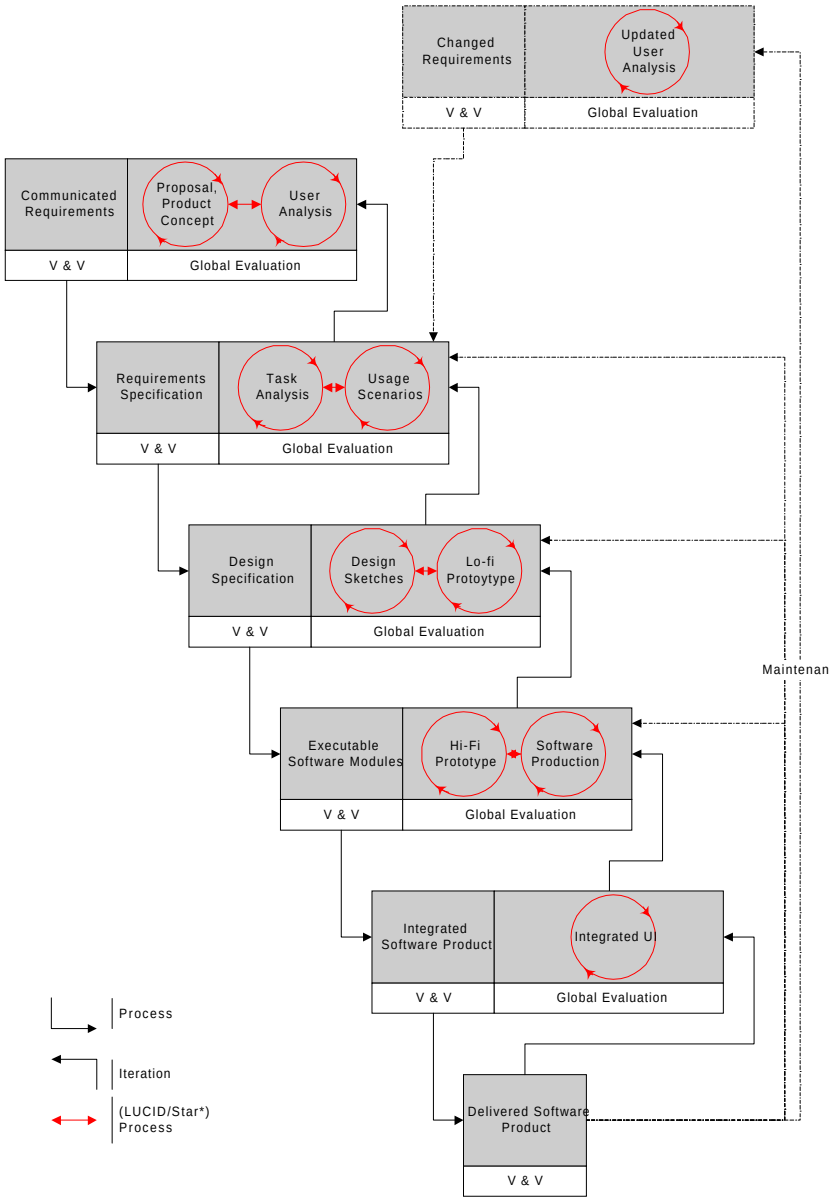

Figure 13: An instance of the (LUCID/Star)* process model running parallel to the improved Waterfall model

7.1.7. Compatibility with the creation of effective usability engineering support tools

In addition to establishing the capabilities of the (LUCID/Star)* usability engineering process model, we also recognized the need to provide tools for process management, technique selection and instruction, and customization. Such tools have been under experimental development at the Usability Methods Research Laboratory of Virginia Tech.

Usability tools are being developed to aid practitioners in instantiating this process model. These tools range in subject, from overall project management to technique selection and tutorials. The model provides an excellent visualization scheme, which

allows toolmakers to incorporate these visualizations and create an accurate rendition of the model's structure. This rendered structure can then be customized to meet the implementer's needs and thus create a specific instance of the process. Figure 14 below shows a preliminary design of a process instantiation/creation tool. Practitioners use this tool to design and customize their process [Sawicki and Whiting, *unpublished*].

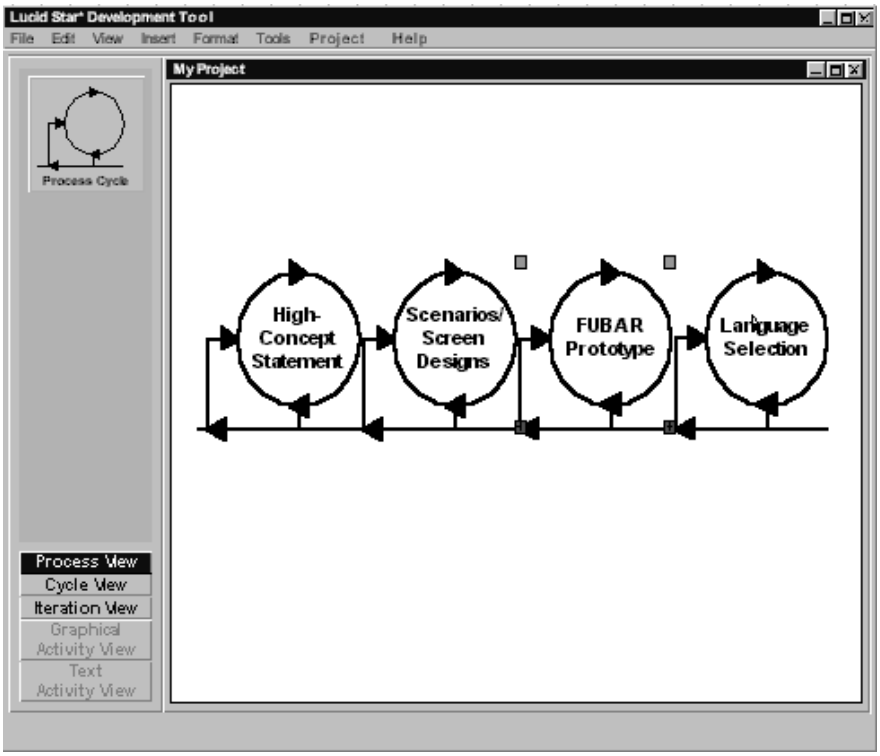

Figure 14: A screen design from the process management tool

Other tools include: tutorial tools to step practitioners through specific techniques; technique selection tools that help implementers choose appropriate techniques based on project constraints, interaction style/application type, and product evolution; and project management tools to aid in scheduling and integration.

7.2. Other Results

In addition to the goal related results above, we also observed and make conclusions about several other aspects of the development cycle. In particular, I made a number of interesting observations about early design, task analysis, usage scenarios, and design walkthroughs.

7.2.1. Overlap of HTA, Scenarios, and User Classes: Much more powerful than we anticipated, or than is generally believed

A combination of three techniques proved very useful in both the traditional web-design and the wireless design. Developing usage scenarios, analyzing task structure, and identifying defining user characteristics, each technique fed the others in a way that

created an information feedback loop, resulting in a very complete conceptual picture of application use. The scenarios utilized the defining user characteristics to create a more realistic narration of task performance. In turn, the scenarios informed the task structure in such a way that mapping the tasks and sub-tasks in a more system-oriented way became simplistic. This observation is consistent with the "80/20" rule of software engineering, stating that while 80 percent of development time is spent in design, 80 percent of design problems are discovered there as well.

7.2.2. Overlap of HTA, Scenarios, and User Classes: Much more iteration and attention required than we anticipated

The connection among these three general-purpose techniques powers the early design stages of the interaction development. As a consequence they require much more attention and iteration than is generally expected. For example, over a period of seven weeks we found ourselves iterating each of these about nine times. Most usability methodologies consider these techniques in isolation, commenting on the usefulness of each in a limited context. However, if one wishes to harness the true power of the techniques then they must be considered together. Each feeds the other and thus we learned that whenever one was iterated the others should be re-considered as well. This period of redesign stretched the limits of our confidence, but in the end we learned to "trust the process" and see it through to the end.

7.2.3. One cannot overestimate the power of design/usability walkthroughs.

Much of our evaluation of the early design consisted of performing usability or design walkthroughs. The usability experts would perform a walkthrough of the screen sketches, using a set of scenarios. From these sessions we discovered a plethora of usability problems and design improvements. These walkthroughs provide quick turnaround of the design and can account for the discovery of scores of usability problems.

However, even the usability expert can have misguided intuition, and can spend a lot of time refining a nonviable screen design. To combat this we learned to disseminate the design early. We encouraged people who were not involved with the usability effort run through some scenarios, soliciting their reactions. Internal evaluation can save the company a great deal of time and can foster greater customer satisfaction later.

In our case, we had developed a navigation scheme based on tabbed panes and windows. The design reflected the deep hierarchical nature of the task analysis. We were very pleased that the highly structured design had fallen so readily from the task analysis. As we pursued the navigation schema it became apparent that we needed nested tabs to represent the break down of a super task into its subtasks. To maintain consistency in the design we left the tabs in the same place, so the effect for the user was that when a user selected a tab, that tab and it's sibling would be replaced by a new set of tabs representing subtasks. Changing the tabs was confusing to our evaluators, and resulted in them getting totally lost in the system. This feedback was essential in redesigning the interaction and helped us avoid allowing major usability problems to be passed on to the user.

One important qualifier to this point is the fact that while internal evaluation can be very effective, it does not replace end-user evaluation. A system's end-users have the unique perspective of being the one group that will use the system repeatedly in order to perform tasks. As the ultimate users, they are the group whose opinions most matter in determining usability of the product. In some cases, internal evaluators are in synch with user expectations, but the only way to discover how in synch they are is to evaluate with end-users.

7.2.4. Early Design makes the implicit, explicit.

One true advantage of interaction design via a process model lies in the fact that in implementing the model one begins to see the implicit made explicit. The structure imposed by the model helps assure that nothing is left unexamined, greatly reducing the chance of surprise problems.

For example, in the process of performing task analysis, sub-tasks would surface from analysis of the application structure but did not appear in any scenarios. One such sub-task was the identifying of a current object of interest (establishing currency). In everyday activity one does not have to explicitly determine which object he / she wishes to manipulate; he / she would simple pick it up and work with it without thinking, "I will select that item and manipulate it." From the systems view, such an action must be explicit to avoid the application assuming too much of the user's intention. Since the step is not intrinsic to everyday life, it was left out of the scenarios, which are designed from a human viewpoint. In this way, task analysis illuminated the revision of the usage scenarios, making them more complete and comprehensive.

7.2.5. Traditional usability focuses on the later stages of usability evaluation. Our work demonstrated the importance of early analysis and design.

Much of the work that has been done in the area of usability focuses on post-design evaluation. Techniques for formative and summative evaluation are emphasized with the intention of improving the design in the "next version." Interaction design begins with the solidifying of a product concept. Usability in the early stages of product design helps ensure a usable product and cut the costs of later redesign. In our case, before any code was devoted to the interface, many of the usability problems had been identified and corrected.

7.2.6. It helped to have a backend software person on the team to keep us realistic

One of the obvious ways to connect the interaction design to the software design is by including a software developer in all the important usability decisions. This not only helps keep the usability team up to date on the technical backend, but also provides the functional core developers with an insight in controlling information flow. A software engineer in the midst of usability experts can help keep platform constraints and restrictions in the forefront while gaining a better understanding of user needs and task structure. A similar observation could be made about including usability personnel in software development decisions.

8. Summary and Conclusions

For Optim, this project was a grand success. They have acquired a usability process that meets the special needs of their business and that can be adapted and re-used use for future projects. Perhaps the biggest measure of success for Optim was that their prototype was presented at several venture capital fairs and has secured them five million dollars in venture capital funding so far.

(LUCID/Star)* performed remarkably in the Optim case, overcoming the tight constraints of budget and schedule cuts to produce an excellent prototype of the system. We were able to conclude either empirically or analytically that (LUCID/Star)*,

- is a flexible, non-linear, iterative progression through the user interaction development cycle
- accommodates existing methods and techniques for development activities
- can be customized to meet the special needs of a given project environment
- applies to a variety of interaction styles and applications types
- includes the generation of documentation work products as an integral part of the process
- supports realistic connections to the software engineering process for developing the functional core
- is compatible with the creation of usability engineering support tools

8.1. Future Work

Presented in this thesis is a qualitative study of the (LUCID/Star)* process model and its instantiation at Optim System, Inc. Although the process resulted in a satisfying version of the product and helped Optim gain funding, more work needs to be done to assess the effectiveness of the model. A quantitative study that compares an (LUCID/Star)* process to existing usability methodologies should be undertaken to uncover whether or not the process model can produce more effective processes.

(LUCID/Star)* has proven able to organize and structure the usability effort in a small real-world development environment, but still needs to be incorporated into a large-scale development effort. Such an effort would place different constraints and pressures on the process model and would provide further insight into the usefulness of (LUCID/Star)*.

We believe that (LUCID/Star)* identifies and utilizes the fundamental structural pieces of any design process. These pieces are represented in (LUCID/Star)* as activities, product evolutions, iterations, techniques, and cycles. Work needs to be done to investigate whether more such fundamental pieces can be found and to begin defining a grammar for process definition. It such a grammar existed, defining processes would be an exercise in organizing the pieces and defining the evolutionary drift.

9. Acknowledgements

I would like to extend a special thanks to Drs. Rex Hartson, Deborah Hix, and James D. Arthur for their guidance and feedback as I was undertaking this research and write-up. Without them none of this work would be possible.

10. References

Boehm, B. W. (1988). "A Spiral Model of Software Development and Enhancement." IEEE Computer, 21(3), 61-72.

Cognetics, http://www.cognetics.com/lucid/index.html

Hix, D. and Hartson, (1993). H.R. Developing User Interfaces: Ensuring Usability through Product and Process. New York: John Wiley & Sons, Inc.

Hartson, H. R., & Hix, D. (1989). "Toward Empirically Derived Methodologies and Tools for Human-Computer Interface Development." Int. J. Man-Machine Studies, 31, 477-494.

McCrickard, D. S., Stasko, J. T., and Catrambone, R. (2000). "Evaluating Animation as a Mechanism for Maintaining Peripheral Awareness." GVU Technical Report GIT-GVU-00-01.

Muller, M. J. (1991). "PICTIVE An Exploration in Participatory Design." *In Proceedings of CHI Conference on Human Factors in Computing Systems.* New York: ACM, 225-231.

Royce, W. W. (1987). "Managing the Development of Large Software Systems: Concepts and Techniques," *In Proceedings of Wescon*, August, 1970. Also Available *in Proceedings of ICSE 9*, Computer Society Press.

Sawicki, D. and Whiting, J, Unpublished Report.

"Virginia Tech CS Online SE Module" *Online CS modules: The Waterfall Model.* http://manta.cs.vt.edu/cs5704/SEmodule/SE/Lessons/Waterfall/index.html

11. Vita

James W. Helms

Education:

MS, Computer Science
Virginia Polytechnic Institute & State University (Virginia Tech), Blacksburg, VA
Graduate QCA: 3.8

BS, Computer Science,
Virginia Polytechnic Institute & State University (Virginia Tech), Blacksburg, VA
Overall QCA: 3.7 In-major QCA: 3.7.
Second Major: Mathematics Math QCA: 3.7

Work Experience:

Graduate Teaching Assistant, Department of Computer Science, Virginia Tech, Blacksburg VA
August 1999 - December 1999; August 2000 - present

- Spring 2001: CS2704 - *Object Oriented Software* (Instructor)
- Fall 2000: CS5204 - *Distributed Operating Systems*
- Fall 1999: CS3724 - *Introduction to Human Computer Interaction*
- Worked closely with faculty and staff to teach and administer class materials.
- Maintained class websites, and graded assignments and projects

Usability Consultant, Optim Systems, Inc, Falls Church, VA (funded by the Center for Innovative Technology)
December 1999 – October 2000

- Designed and evaluated the user interface for a remote device management system.
- Managed a team of design and implementation interns while building the interface.

Research Assistant & Software Developer, Learning in Networked Communities (LiNC), Center for Human Computer Interaction, Department of Computer Science, Virginia Tech, Blacksburg, VA
June 1997 – August 1999.

- Designed and developed networked applications and interfaces for use in high schools
- Performed field studies to determine usability of designed software

Computer Skills:

12. Hardware:	**13. IBM PC (Pentium), X-terminal, Macintosh, and Palm OS**
Operating Systems:	Windows 9x, Unix, Windows 2000, Windows NT, Linux, and MS-DOS
Programming Languages:	Java, C++, TCL/TK, UIML, HTML, BASIC, Pascal, and Perl scripting
Programming Environments:	JDK, and Microsoft Visual Studio;

Publication:

Helms, J., Neale, D. C., and Carroll, J. M. (2000). Data logging: Higher-level capturing and multi-level abstracting of user activities. In *Proceedings of the 40th annual meeting of the Human Factors and Ergonomics Society* (pp. 303-306). Santa Monica, CA: Human Factors and Ergonomics Society.

Honors, Awards, & Scholarships:

Member, Upsilon Pi Epsilon, the National Computer Science Honor Society

- Secretary / Treasurer (Local Chapter), Fall 1998 – Spring 1999.

Member, Association for Computing Machinery
Member, Phi Kappa Phi and Phi Beta Kappa National Honor Societies