The LUCID Framework

(Logical User Centered Interaction Design)

developed by
Cognetics Corporation

51 Everett Drive • PO Box 386 • Princeton Junction, NJ 08550
Phone: 609-799-5005 • Web: www.cognetics.com • Email: info@cognetics.com

The LUCID Design Framework
Pre-Release Version 0.4 – January 1999

Acknowledgments

The LUCID Design Framework was developed at Cognetics Corporation under the leadership of Dr. Charles Kreitzberg. Many people contributed to its development including Whitney Quesenbery, Scott Gilkeson and Janis Morariu. Inspiration also came from friends of Cognetics, including Dr. Ben Shneiderman, whose book Designing the User Interface (3rd Edition, Addison, Wesley, Longman, 1998) has been a constant companion.

Introduction to the LUCID Framework

LUCID – Logical User Centered Interaction Design – began as a way of describing the approach to interface design at Cognetics Corporation. Over the years, it has evolved into a framework to manage the process of designing an interface in a way which can, if not guarantee, at least encourage software usability.

The role of the LUCID Framework

Over the past 30 years, several techniques for managing software development projects have been developed and documented. While these techniques have helped large software development projects meet time, budget, and quality goals, they do not directly address usability issues. Because most systems being developed today are interactive, software development methodology must be expanded to include the design of the user interface.

The LUCID Framework was developed to fill this need. It is a methodology for designing the interactional components or "front end" of a software product. The LUCID Framework can be integrated with other software engineering methodologies or, for small product development efforts, can be used as a stand-alone methodology.

LUCID is called a "framework" because it does not rigidly require specific techniques. At various points in the LUCID process, the design team may either decide to use a LUCID-recommended technique to accomplish a particular goal, or may select another technique that seems more appropriate for their particular project. For example, the design team may select one of several techniques for studying user needs and workflow during the requirements definition process. As another example, the design team may choose to use empirical usability testing or inspection methods to detect flaws in the user interface.

The LUCID Framework allows creative designers to function efficiently and effectively. By systematically following the recommendations of LUCID, designers will be able to:

- Develop a clear product concept that will assist in communicating the project's goals and provide a means for ensuring the conceptual integrity of the product.
- Identify the business objectives that motivate the product development effort.
- Acquire an understanding of the structure of the user population and the needs of each segment.
- Construct a complete needs and functionality analysis from the user's perspective.
- Develop a navigational structure which implements all required functionality in as simple a manner as possible.

- Create screen control and display objects to efficiently implement the navigational structure.
- Develop a prototype embodying the key screens and navigation.
- Refine and extend the prototype through usability testing.
- Develop a complete set of programming specifications for the "front-end" of the product.
- Support software engineers by creating design solutions for problems which arise in the course of development.
- Facilitate the development of on-line help, documentation, and tutorials.

Description of the LUCID Framework

LUCID is organized into six stages:

Stage 1: **Envision**	Develop UI Roadmap which defines the product concept, rationale, constraints and design objectives.
Stage 2: **Analyze**	Analyze the user needs and develop requirements.
Stage 3: **Design**	Create a design concept and implement a key screen prototype implementing this concept.
Stage 4: **Refine**	Test the prototype for design problems and iteratively refine and expand the design.
Stage 5: **Implement**	Support implementation of the product making late stage design changes where required. Develop user support components.
Stage 6: **Support**	Provide roll-out support as the product is deployed and gather data for next version.

Each of these stages is completed in sequence building the elements of the interface until the design is complete. Many of the tasks within a stage are iterative – repeated in a rapid cycle with review tasks until the result is a satisfactory conclusion. In addition, key documents such as the UI Roadmap and the requirements analysis are reviewed at the end of the design stages to ensure both that any new information is incorporated and that the design work has stayed within the scope outlined in them.

LUCID Framework

In creating any software product, many different areas of concern must interact in a sometimes chaotic series of meetings, decisions and events. The ultimate goal of any methodology is to tame this chaos into a repeatable process in which all concerns can be addressed in an appropriate time and manner. It is important that anyone managing a product development project think carefully about how decisions made in one area affect other aspects of the overall product design. This is of particular concern to the interface designer because to the user, the interface *is* the software. The interface design shapes the experience, and can mask or exacerbate technological glitches or an inelegant technical solution. Most importantly, the interface design shapes the users' mental model of the work the software can perform. If the interface concept is out of sync with the task model, it is inevitably less usable.

Some of the interactions between areas of concern are addressed directly in the tasks of the LUCID Framework.

The Business Case

The business case is considered directly in the UI Roadmap produced in Stage 1. When the UI Roadmap is reviewed during the analysis and design stages, the design team has an opportunity to revisit the business assumptions and ensure that the financial basis for the product has not changed in a way that affects the interface design. In the Support stage, the interface and product are evaluated to see how well these business objectives were met (and suggest remediation when needed.)

Resources

Most interface design happens in a world with explicit or implicit constraints on the scope of a project in the budget, manpower, and expected duration. It can be difficult to properly estimate the timeline or costs for a project before the interface design is complete, and many things conspire to throw project management off track. An extra iteration of a prototype, an interface solution which requires new technology, even a change in the release date of a supporting technology can all derail a schedule. This affects the interface designer in two ways.

First, like any project manager, the leader of the design team must frequently review progress against expected deadlines. LUCID assists in this process by defining a clear sequence of deliverables which must be met. When the fourth version of the visual design has failed testing during Stage 3 Task 5, it can be difficult to tell how long it will take to complete this element of the design. But, it is clear where the design stands in the larger process.

Second, a successful interface design relies on acceptance and implementation by other groups, especially the developers. The design team must be sensitive to ways in which decisions made in the interface can affect other resources. Again, the sequence of deliverables in LUCID assist with this by providing clearly defined review points when other interested groups can review the

design and comment on it. Changes required by their concerns can be integrated into the design early, rather than disrupting the process in its late stages.

Physical Environment

A description of the physical environment is one of the items included in the UI Roadmap as an element which must be factored into the design. Contextual user analysis – one of the techniques which can be used in Stage 2 – concentrates on observing users in the context in which the software will be used.

One of the weaknesses of formal usability testing is that it cannot completely reproduce the full physical environment with all its distractions and less-than-ideal conditions. When the physical environment is a critical factor in the design for any reason, care must be taken to evaluate the design within that environment.

Technical Environment

The technical environment is equally important, especially during periods of rapid change. As in the case of the resources concerns, the interface design team must be careful to review each iteration of the design with the systems engineers. These reviews can ensure that technical problems are solved early, obtain buy-in for new design ideas from the engineers, and even be an opportunity to solicit new technology ideas which might be the foundation for solving a design problem.

Users and Usability

Users are at the heart of LUCID, starting with a consideration of the target user groups and user task analysis in Stages 1 and 2. Feedback from users is an important part of the iterative cycle of the design in Stages 3 and 4, with possible additional usability tests during implementation in Stage 5. Finally, supporting users as they interact with the new software, and surveying them for product satisfaction is the primary goal of Stage 6.

User Assistance: Documentation, Training and Help

Helping users be more successful with the software is a goal of LUCID interface design. Whether user assistance is provided in clear on-screen prompts, wizards and coaches, or in more traditional documentation, training and help the design of that assistance must be created as part of the interface design process. The result of leaving the design of user assistance until the end is the same lack of integration which has given online help its bad name.

LUCID Framework prerequisites

There are several prerequisites for using the LUCID Framework:

- A need has been identified.
- There is corporate support for the project.
- The design team has been assembled.

A need has been identified.

While LUCID focuses on defining and refining individual user needs, it does so within the scope of a project that has already been basically defined. Some idea of the scope of the project is needed to begin the design process, which in LUCID begins with the creation of the high concept statement.

A related issue is an early definition of the size of the project. The size of the project will determine whether the LUCID Framework can be used as a stand-alone methodology, or whether it is used in conjunction with another methodology such as CASE (Computer Aided Software Engineering).

Characteristics of Software Development Projects			
Small projects	1 or 2 programmers	The design team and programming team can function as a single integrated unit	Minimal programming load; projects such as CD-ROMs, self service kiosks, instructional software
Medium projects	As many as 12 programmers	There is a single design team and a singe programming team	Typical of many corporate projects. Requires formal communications between the two teams
Large projects	More than 12 programmers	There are multiple design teams and multiple programming teams	Coordination, communication and scheduling are complex. Design teams must develop and use a common style guide and programming specifications must be carefully coordinated with the systems engineers

LUCID works very well for small projects, where it can serve as the sole design methodology. LUCID will typically be used in conjunction with some other methodology on medium and large projects.

There is corporate support for the project.

While many aspects of LUCID involve communicating goals and assumptions to interested parties outside the design team, it does not directly address obtaining initial corporate support for the project.

The design group has been assembled.

At a minimum, the LUCID design group should include a **usability specialist**, a **visual designer**, a **technical writer**, and a **programmer**. If the project content is technical or esoteric (such as a scientific project), a subject matter specialist should be added to the group.

In large projects the number of design group members will increase. Depending upon the size of the project, one individual may take on several roles. If the usability specialist is also a subject matter specialist, the design process can be greatly facilitated.

> The role of the **usability specialist** is to lead the design process and manage the group. While many usability specialists come from the behavioral sciences, especially cognitive psychology, it is important that the usability specialist also has a good technical background and some experience in (or familiarity with) programming. A technical background allows the usability specialist to communicate effectively with the more technical system designers and evaluate the complexity of design decisions.
>
> The **visual designer** is responsible for implementing screen designs and assisting in screen layout. Because it is difficult to determine exactly how much data will fit on a screen, the visual designer must create designs on-screen. The visual designer should also take the lead in developing the program's overall "look and feel," and assist in creating any special "widgets."
>
> The **technical writer** is responsible for developing the usability engineering specifications that will be incorporated into the overall programming specifications. If a user manual is planned, it should be developed as part of the design process. A user manual is a valuable resource for the programming team.
>
> The **programmer** is responsible for employing rapid prototyping techniques to develop a working model of the user interface. This model will allow the design team and management to see how the software will look and operate. More importantly, the prototype will be used for usability testing to refine and validate the design. If there is a software engineering group distinct from the usability engineering group, the programmer should be a member of both. This will improve communication and simplify problem resolution.

Besides these core members, the design group should have access to representatives of the user community. Participatory design improves the quality of the design and its acceptance. The representatives of the user community provide information and serve as "reality checks," making certain that the design reflects the way the software will actually be used.

In addition, a representative of the business unit in which the software will operate should be involved. If the software is destined for internal use, this representative will probably be a manager from the business unit that commissioned the software. If the software is being developed for an external customer, the representative will be associated with them. If the

software is offered for sale (either stand alone or bundled with a product) the representative will probably be from marketing and sales.

LUCID Design Principles

A **design principle** is a general statement that describes some aspect of user interface design 'best practice.' These principles usually have a basis in research concerning how people learn and work. Good user interfaces conform to good design principles.

The following design principles are applicable to all user interfaces:

1. Maintain consistency at all levels.
2. Minimize demands on human memory.
3. Present informative feedback.
4. Keep the user in control.
5. Use a consistent conceptual model.
6. Design for prevention of and recovery from error.
7. Design for simplicity.
8. Design the software to mirror workflow.

Additional information about these design principles may be found in Appendix A.

While these principles identify appropriate design goals (consistency, simplicity etc), they do not tell the designer how to achieve these goals. The exact application of design principles within a particular software design project is governed by **design guidelines**. Design guidelines take the form of a specific goal for a particular user, environment, technology, etc.

Design guidelines can come from several different sources. For example, design guidelines may be mandated by a corporate style guide, or by a particular operating system. They are also developed during the interface design process.

Unfortunately, even design guidelines are usually not specific enough to help with specific design issues. For example, if several designers are working on the same user interface, each one could implement the same guideline in a different way. In cases like this, there may be an even more detailed set of statements called **local rules**, **design rules**, or **design standards**. Unlike guidelines, local rules describe exactly how a given situation should be handled; they are absolutes for the context in which they are applied.

LUCID Cognitive Theory

The fundamental principle behind LUCID is:

Design the interface to minimize the cognitive load on the user at all times.

Minimizing the cognitive load (which is defined as the amount of mental effort a user must exert to use a product) has several key components:

1. Select cognitive strategies that minimize the level of cognitive processing.

   High level—Problem solving
   Mid level—Recall action to take
   Low level—Recognize action to take

2. Use meaningfulness to reduce cognitive load.

   Make certain the user possesses all prerequisite concepts.
   Make certain that the prerequisite concepts are activated.

3. Avoid short-term memory load.

   Do not expect the user to remember the contents of any screen not currently displayed.
   Do not require the user to remember commands or specialized concepts.

4. Design screens for rapid perceptual processing.

   Use colors and grouping to create easy to comprehend perceptual groups.
   Overstate critical cues by emphasis and repeating them.
   Use appropriate techniques to focus the user's attention on critical screen elements.

5. Avoid inferential leaps.

   Do not ask the user to make assumptions and predictions.
   Do not force the user into unneeded search strategies for functions or data.
   Do not surprise the user with unexpected consequences of actions.

Additional information about cognitive psychology and how it relates to usability engineering can be found in Appendix B