Proceedings of First World Conference on Computational Medicine and Public Health, Austin, TX. World Scientific Publishing Co., New Jersey. 1994.

How to Design a Genome Database

Mark Graves, Ellen R. Bergeman, Charles B. Lawrence

Departments of Cell Biology & Human and Molecular Genetics,
Baylor College of Medicine

Correspondence:

Mark Graves
Department of Cell Biology
Baylor College of Medicine
One Baylor Plaza
Houston, TX 77030

mgraves@bcm.tmc.edu (e-mail)
713-798-8271 (voice)
713-798-3759 (fax)

Ellen R. Bergeman
Department of Human and Molecular Genetics
Baylor College of Medicine
erb@bcm.tmc.edu (e-mail)
713-798-8105 (voice)

Charles B. Lawrence
Department of Cell Biology
Baylor College of Medicine
chas@bcm.tmc.edu (e-mail)
713-798-6226 (voice)

Abstract

We have discovered that designing a database in stages and from three perspectives simplifies the development of genome databases.

The stages of design include analysis, prototyping, and completion of the full system. Analysis involves discovering the needs of the user and initially describing the solution. Prototyping includes development of specific parts of the application which allows the developer to evaluate the current design, making changes as necessary. Development of the full system includes checking for accuracy and consistency within the subsystems and completing any detail design decisions which were postponed until the end.

The three perspectives we use in designing the database are from the viewpoint of the data, the user, and the interaction of the user with the database. From each perspective, we generate a document to describe the concepts and relationships of data in the database; how data will be displayed for the user; and the process in which the user interacts with the database to complete a task.

By looking at each stage of design from three different perspectives, we found that we were able to discover important information not included in the initial requirements specification, resolve potential problems before they became too large, and improve communication between the user and developer.

Keywords: database development, software engineering, genome informatics, conceptual modelling, user interface design, graph-based representation, Human Genome Project.

1. Introduction

In the process of developing a database to support genome data, we have discovered that designing a database in stages and viewing the development from three perspectives simplifies development and provides a framework for producing a product which is more likely to meet user needs.

The stages of design include analysis, prototyping, and completion of the full system. Analysis involves discovering the needs of the user and initially describing the solution. Prototyping includes development of specific parts of the application while ignoring details necessary for the complete system. Prototyping allows the developer to evaluate the current design, making changes as necessary without affecting the design of the complete system. Development of the full system includes checking for accuracy and consistency within the subsystems and completing any detail design decisions which were postponed until the end.

The three perspectives we use in designing the database include the conceptual schema, user interface mock-ups and dialogues. A conceptual schema describes concepts and relationships to be stored in the database. User interface mock-ups describe how data will be displayed for the user. Dialogues describe the process in which the user interacts with the database to complete a task.

By looking at each stage of design from three different perspectives, we found that we were able to discover important information not included in the initial requirements specification, resolve potential problems before they became too large, and improve communication between the user and developer.

2. Three Perspectives

Rather than develop one document which captures the entire system, we have found it useful to describe database access applications from three perspectives: the data, the user, and the dynamic between the data and the user. These three perspectives describe: what the data is, what the data looks like to the user and what the user does to the data.

2.1 Conceptual Schema

A conceptual schema describes the concepts and relationships in a domain. We use a graph data model for capturing the concepts and relationships of genetics. Conceptual modeling is the process of describing a database domain [1]. It is necessary to capture the essential concepts in a model without making decisions about the importance of each concept. We use a graph data model [2] because we have found that graphs capture the structure of genome data best.

A conceptual model is more than a static description of the data. It should also describe the operations which are to be performed on the database and the constraints which must hold on the data. We have found it useful to augment a conceptual schema with constraints and operations later in design. Constraints include cardinality and type, and operations describe how data is stored and accessed. It is also useful to describe queries which are likely to be asked. There is not a general way of describing operations in a manner independent of the database management system used, but we have found it useful to list the operations which may be performed on the data. Constraints, such as the restriction that some characteristic of two concepts must refer to the same data item, may be described as text or represented on the conceptual graph.

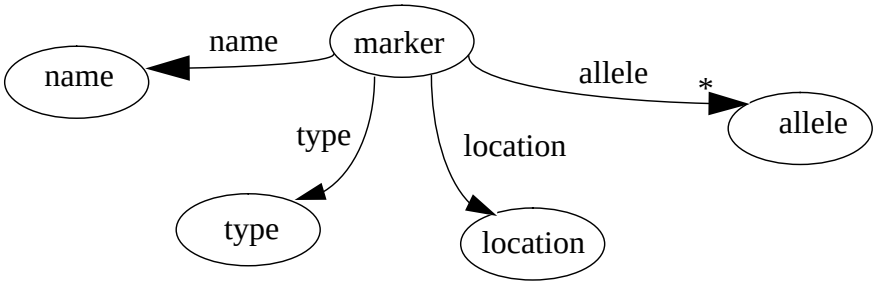
There are many different conceptual models which could be used to define a conceptual schema. A natural language, such as English, meets most of the requirements for a conceptual model. Natural languages are expressive, natural and easy to use. Unfortunately, natural languages are not easily modeled on a computer. Another data model which could be used for conceptual modeling is an object-oriented language. Object-oriented languages appear to be too behavior-oriented for conceptual design. We have found graphs to be useful for developing genome databases.

A graph conceptual model is a language for describing the concepts and relationships of a domain as a labeled, directed graph. Nodes represent concepts and arcs represent the links that connect concepts to each other. Using only nodes and arcs gives the developer the freedom to capture all concepts without getting caught making decisions about relative importance of the concepts. Entity-relation diagrams are too tied to the relational model and require too many decisions be made early in the design to be a good conceptual model for genome databases [3]. However, they have been used successfully in the design of many databases [4,5]. Our graph conceptual model is based on feature structures [6], ψ-types [7], conceptual graphs [8], and terminological description logics [9] which are representation formalisms used in computational linguistics. Other graph data models include GOOD [10], G+/GraphLog [11], and Hyperlog [12]. We have geared our conceptual model toward representing genome data and tried to minimize the number of design decisions which must be made in the initial description of data.

A conceptual schema may be broken up into several conceptual diagrams. A conceptual diagram is a graph of concepts and links that connect them. A conceptual diagram consists of a collection of concepts, a collection of links (link names), and a collection of link-labeled edges between two concepts, *D=(C,L,E)* where $E \subseteq L \times C \times C$. Each edge is annotated with the cardinality of the concepts it relates, e.g., one-to-one, one-to-many. The cardinality is a pair of either any positive integer or “many”.

A conceptual schema is a collection of conceptual diagrams that define views of the domain. Concepts and links may be shared between diagrams. A conceptual schema should include enough diagrams to show all the links that a concept may have and the various configurations in which the links may occur, but it is not necessary to show each state in the concept’s life cycle. When the database is developed, there may be a view of the data which corresponds exactly to a single (or a few) conceptual diagrams, but it is not important to capture all of the views initially. For example, a marker allele may have a name and a frequency in a certain population, or it may have a size and a frequency in a certain population, but alleles don’t typically have both names and sizes.

An example diagram for a genetic marker should include the characteristics: name, marker type, location, and alleles. This is denoted by the graph:



A marker has one name, one type, one location, and many alleles. The “many” cardinality is denoted by an “*”, and the “one” cardinalities are omitted. Name, marker type, location and allele are other concepts which would be described in different diagrams.

A conceptual diagram should be annotated with constraints which must hold between the concepts and links. Constraints augment the conceptual diagram by providing additional information on the concepts and links. For example, a constraint could state that the length of each cosmid sequence is between 10K and 100K base pairs. Although some work has been done on graphical representation of constraints, we have found it best to describe the constraints as text.

Figure 3 shows a conceptual diagram for a filter hybridization experiment, a concept frequently represented in genome databases.

2.2 UI Mock-up

The user interface is the only part of a database application that a user ever sees. User interface mock-ups document how data may be presented to the user and operated on by the user. It is important to spend time describing how the data will be displayed for the user and how the user will interact with the data. User interface mock-ups are a tool for communication between developer and end-user, providing visual descriptions of potential interfaces. Using this tool, the user and developer can explore ways of providing interfaces which best meet the user’s needs. At the same time, the mock-ups provide documentation about user needs and expectations for the application.

It is important to capture the interface between the user and computer early in design, because of its importance to the user and the effect it can have on the system design. It is also a good method of testing the developer’s understanding of the user’s requirements. It is sometimes difficult to get user feedback on data abstractions, but most users will be willing to critique a bad user interface. The developer should not spend too much time on the mock-ups, since the idea is to present rough outlines of what is possible. If too much time is spent on the design, it detracts from the rest of development, the designs starts to look forced, details are worked out which will not be needed later, and the end user may assume this is the final design and resist when changes must later be made. However, the mock-ups do require thought. Most end users do not have the time to critique a long series of bad designs. Showing a few well thought out designs over the course of development should meet the needs of the developer and user.

We have found it useful to develop pencil and paper designs at the early stages until some part of the design solidifies, then document that, and not to use UI prototyping tools until late in the design process. By beginning with a fresh sheet of paper at each iteration, the good ideas of previous versions are often reused, but the insignificant details are frequently changed. After a few iterations, it will be difficult to think of a better alternative to some part of the design. When that occurs, it is useful to document the part of the design which is solid. Another part of design can then be worked on. The process of iterating over the design is continued until a complete design is worked out.

User feedback should be obtained when a design is completed or when the developer gets stuck. End users should be used to obtain feedback when the developer believes the user’s needs have been understood and met. This corrects misunderstandings as quickly as possible and reassures the users that their needs are being considered. When the developer is stuck, consulting with the user may clear the block. For example, the user may have ideas about how they think the user interface should look, or they may be able to clarify for the developer that the developer has overemphasized the importance of a part of the design.

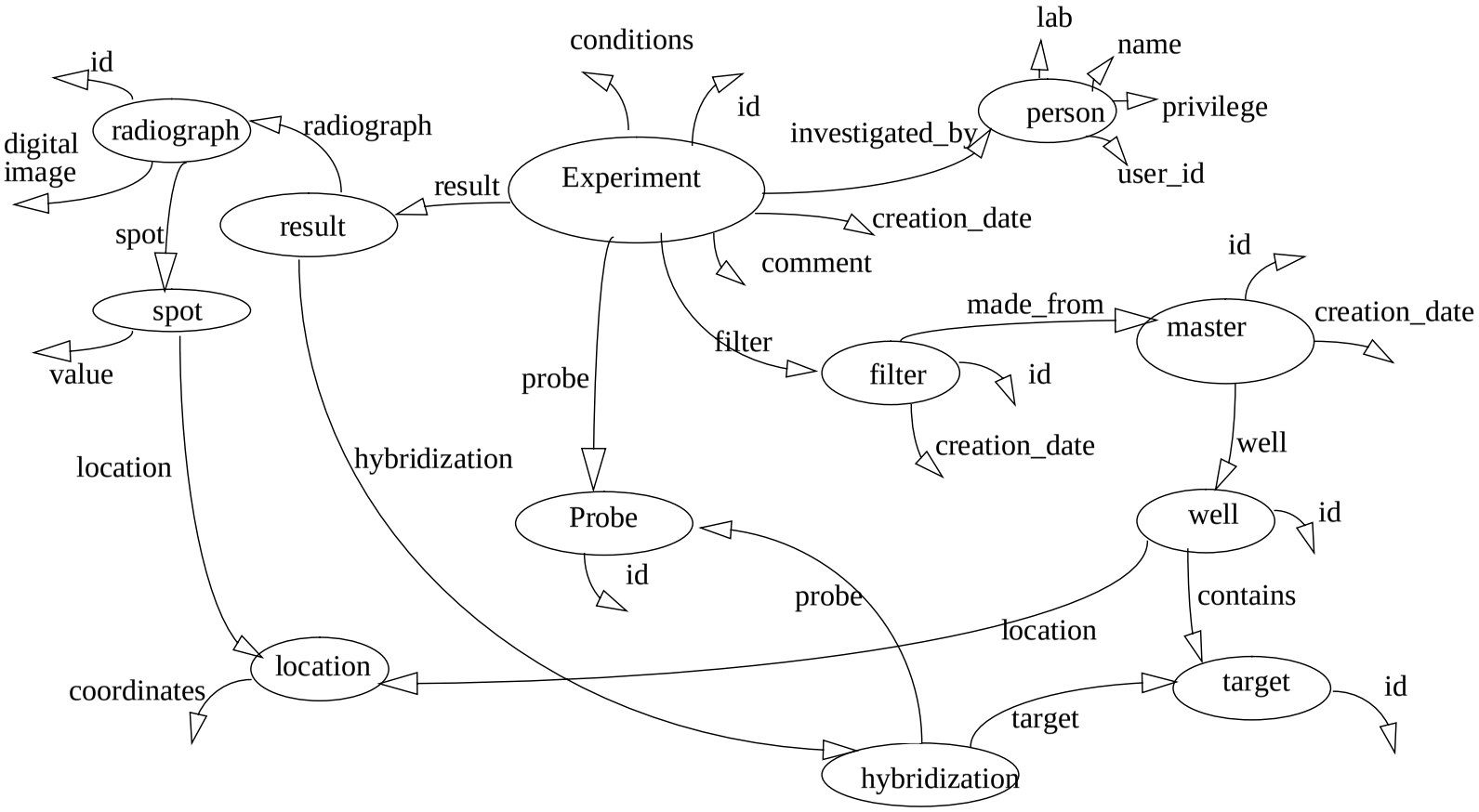
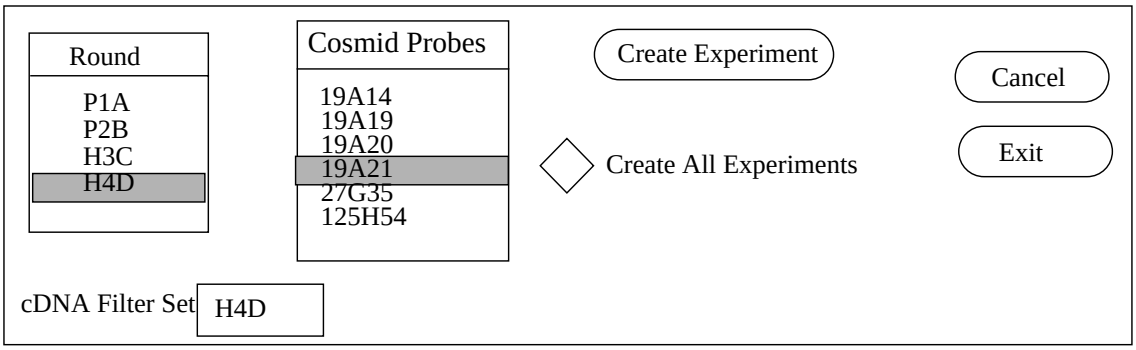


Figure 3: Conceptual Model for Abstract Experiment

Working together to clarify misunderstandings builds a relationship between user and developer which leads to production of a much better product. Time spent resolving misunderstandings should be considered a valuable part of the design process and should not be taken negatively.

One useful strategy is to develop a series of many throw-away mock-ups over the course of design. Even when a reasonable user interface is developed, it may not meet all the needs of the user. At that point, it is important to balance the inconvenience to the user with the extra development time required to meet them.

When documenting a mock-up, it is useful to give a narrative description of the complete process which uses that interface. The narrative should include descriptions of alternative paths through the system. For one system being used at Baylor College of Medicine, the user needs to create hybridization experiments between a cosmid and a filter containing cDNAs. The system has a Create Cosmid->cDNA Experiment window which is displayed for the user after the appropriate option is selected. An early mock-up of that design is:.



Each item in the window is described in the documentation:

- **Round** is a selectable display of rounds which have cosmids which have been marked as probes.
- **Cosmid Probes** is a list of cosmids which have been marked as probes for the selected round.
- **cDNA Filters** is a read-only display of the filter set which can be used for the experiment. This display changes as each round is selected to display the name of the set.
- **Create Experiment** is a button for storing the probe and filter information and creating an experiment based on this information
- **Create All Experiments** associates the target filter will all probes as individual experiments.
- **Cancel** kills the window without saving any current changes, **Exit** saves the current changes and then kills the window.

The user interaction with the interface is also documented:

1. The user selects a round, cosmid probe and clicks on create experiment.
2. By selecting Create All Experiments, the user creates a series of experiments based on the same target filters and all probes in the probe list.
3. Creation of experiments (step 1 or 2) continues until exit or cancel ends the use-case.

2.3 Dialogues

Dialogues describe the purposeful communication between the end user and database. They describe the tasks that an application must support and form the basis for implementing the code to perform those tasks. They also document the tasks to be performed and their subtasks.

Dialogues are descriptions of the interaction between the database application and user. Each dialogue describes a task specified in the requirements or discovered during design. The collection of dialogues completely describes the functionality of the application.

Dialogues consist of step-wise descriptions of the interaction between user and database application. The descriptions should be complete in that they describe the necessary interactions such as data to be passed, errors for which to check, and expected responses. Dialogues should not suggest a means of communication such as mouse selection from a list, but should describe interactions in terms of notifying, informing, or passing information between the two members involved in the dialogue.

A dialogue should completely describe a single task. Within a dialogue, complex subtasks should be broken out and described in their own dialogues. The decision to create a subtask dialogue should be made based on the amount of information needed to implement the subtask. For example, sorting elements in a list may not necessary need to be a subtask but if the type of sort or sort criteria is complex and important to the application, a separate dialogue to describe this would be appropriate.

The use of dialogues in the design process is not new. It has been used in computational linguistics, database design and software engineering. In computational linguistics, they describe the interaction between individuals and are sometimes called scripts, MOPs, dialogue schemata or protocols. In database design, they occur at a detailed level and describe the transactions which occur between a user and a database system.

In object-oriented software engineering, objects are discovered from use cases, which are stepwise descriptions of system interactions. For example, a typical database application must have protections built into the system to prevent accidental loss of data before it is committed. Use cases can be used to decide if the user should be expected to “accept” the data to commit or the application should commit the data as it is entered.

In the Fusion software engineering methodology, Coleman [13] uses scenarios to describe the interaction between the system and a user or other systems. Scenarios are descriptions of sequences of events which occur between agents and the system for a purpose. Agents are not just users in this format, but may be other subsystems or applications. While this is a good general process of describing the system, they are not as descriptive as Jacobsen’s use cases.

In Objectory, Jacobsen [14] introduces use cases, extensive descriptions of each possible interaction between user and system. Each use case describes the complete process that a user would follow from entry into the application until an exit occurs. Each subsystem of the application is also described using an individual set of use cases. Jacobsen introduces a complex notation which we have not found necessary to use. In addition, we have found it useful to document the tasks in a hierarchy to show the shared subtasks.The hierarchy of dialogues consists of tasks initiated by the user, performed by the database and responded to by the user.

When all dialogues are combined, a complete description of the application exists. It is important to confirm that the application meets the requirements in the specification and the user’s general expectations. Dialogues can be given to the user as a tool for communicating the tasks to be performed by the system. At this point in the development process, comments from the user can

correct misunderstandings which could be more costly to correct later in the development process. Any changes should be made before the next stage of design.

An example dialogue is recording a hybridization between a clone on a filter and a clone as probe:

1. The user tells the application which filter and probe should be used.
2. The application checks for validity of the filter and probe. If the filter and probe are not valid, the application informs the user and the user can request another probe or filter.
3. The application informs the user of the clone contents of the filter.
4. The user notifies the application of the desired clone.
5. The user tells the applications what to record as the hybridization result for the two clones. This is restricted to being positive, negative or questionable.
6. The application sends the hybridization information to the database to be stored.
7. The application informs the user of the fact that the data has been stored.

3. Three Stages of Design

Design progresses in three stages: analysis (discover what needs to be done), prototyping (do a little of it) and final design (do the rest of it). It may be necessary to backtrack.

Our goal for design is to create design documents which describes the system well enough to separate programming decisions from design decisions. The design documents includes: a conceptual schema of the data to be stored in the database, a set of user interface mock-ups and a set of dialogues which describe the interaction between data and user. Conceptual schemas serve as a medium for communication between the developer and domain expert. UI Mock-ups make concrete the discussion between the developer and end user. Dialogues document the user's and developer's expectations for the system.

3.1 Analysis

Analysis is the stage of design in which a developer tries to answer the question "What does the user really want from this database and what should be built to meet those needs?" Understanding the data requirements, the user interface requirements, and the interaction between the user and the data provides a means for answering that question. These are captured in the conceptual schema, UI mock-up and dialogue documents.

We have found that a non-linear approach focused on the design documents strikes a useful balance between capturing the needs of the user and meeting the needs of the developer. A nonlinear approach avoids developing a forced design which though logically correct, is overly complicated. Our goal in nonlinear design is to develop a system which is simple to understand and use, not simple to design. By focusing on creating a design document, this limits the scope of our meanderings and brings the result of each iteration to bear on the design.

We incorporate the three design documents in our design process. We select an area of the project to work on and design as much of it as we can focusing on one of the three perspectives, based on the information we currently possess. When we come to a question which we can not answer or which has multiple reasonable solutions, we move on to a different area and perspective.

This approach to database design is different from one in which the documents are worked on together and different from one in which they are worked on in isolation. When they are worked

on in isolation, one perspective does not influence the other perspectives enough. When they are worked on together, they influence each other to much and do not lead to a balanced design.

During analysis, we develop a conceptual schema which is used as a tool for communication between developer and domain expert. We start from the initial user requirements and create a schema which captures the concepts and relationships specified. Using this schema as a baseline for discussion, the domain expert can then modify the schema to more accurately represent the data to be stored in the database. Often during discussion, information about how the data will be used and how the user wants to interact with the data comes out. This information is used to develop dialogues and user interface mock-ups as well as improve the schema.

User interface mock-ups are used as a communications tool between the developer and end user. We initially drew up simple diagrams on paper to show to the users and get their reactions. By providing a concrete example of how the user-interface would act, users were then able to see possible problems, advantages, and disadvantages of our design. This provided good information for creating a useful user-interface while adding to our knowledge of how the user perceived the database.

Dialogues describe the interaction between the user and the database and provide a written description of what the user and developer expect from the database. They also provide a means of discovering parts of the design which were not initially defined in the requirements. We created dialogues based on the requirements specification and used them during discussion with the domain expert about the conceptual schema. We also used dialogues to provide a framework for explaining how the database would interact with a user. Dialogues also provide an initial set of test cases for the implemented database.

During analysis, concentrating on information obtained from only one source, either data, user, or interaction, yields a database which will not meet the real requirements of the project. Finding a means of incorporating all perspectives during analysis yields a much more complete product which is less likely to need major change in future development cycles.

3.2 Prototype

A prototype is a functional system which performs some, but not all, of the tasks required for the final system [15]. After completion of the analysis stage of each phase of development, a prototype should be built which implements the functionality defined during analysis. Prototyping does not implement everything in the system but provides minimum functionality for testing and evaluating the current design. Prototyping a system involves selecting a subset of the system defined during analysis and completing design and implementation. This produces a functional system which performs some of the major tasks of the final product but ignores details. An advantage to building prototypes within the development cycle is that design decisions can be evaluated rapidly and major changes can be kept to a minimum. Prototyping makes the important parts of the system concrete while minimizing the attention necessary for details.

Prototype design of conceptual schemas involves adding constraints and operations to the concepts and relationships represented in the schema. The part of the schema to be prototyped in the current cycle is designed in detail. This includes documenting the specific operations which are appropriate for each object in the schema and defining any constraints to be placed on the objects.

Dialogues which describe the subset of tasks which were selected for the current design cycle are expanded during prototyping. Each task should be described completely, giving a step-wise description of how the database and user will interact to complete the task.

Prototyping the user-interface includes creating detailed mock-ups which completely describe how each task would be completed by the user. From the mock-ups, a user-interface design should be completed in detail to support all tasks from this phase of the development cycle.

Prototyping provides a means of evaluating design decisions rapidly, minimizing the need to throw away large parts of a system, discarding many person-hours of work. The difficulty with prototyping is remembering to stop design and implementation at the appropriate time and evaluate the current product. It is difficult for a developer to remember that the product does not need to meet every requirement perfectly but should capture the essence of the final product instead.

After a prototype has been developed, a decision should be made whether to extend it to a complete system or throw it away. Appropriate design time and other resources should be allocated to allow for either possibility. The decision to use the prototype as part of the full system should only come after an evaluation of the design and implementation.

3.3 Full System

After completion of analysis and prototyping of all subsystems, developing the full system design consists of adding final details to the design and confirming the consistency of all three documents created during design.

After final details are added to the conceptual schemas, they should be checked against the initial requirements documentation for completeness and by the domain expert for accuracy. They should also be checked by the designer for consistency.

The collection of User Interface mock-ups created during design should be made consistent and complete based on the dialogues and documented completely. They should then be shown to the user and the feedback obtained should be used to correct any problems.

Completing the design of dialogues consists of confirming that all tasks necessary within the system are represented and that all subtasks are defined completely. Any redundancies should be eliminated and inconsistencies resolved. The dialogues should be evaluated against the requirements specification, comments made by the domain expert, and needs voiced by the end-user during analysis.

4. The Next Stage -- Building a System

After design is completed, the three design documents drive the implementation. The conceptual schema is turned into a logical schema for the database and into an domain object schema for the application. The constraints and operations describe the procedures which must be implemented on the domain objects. UI mock-ups should be implemented with as little change as necessary, possibly using a GUI builder. Dialogues describe the transactions which must be supported by the application and describe the steps which are necessary for each kind of user interaction.

Implementation of the system is based on the design documents. The documents should completely describe all functionality which is to be implemented. No design decisions should be made during the implementation process. If problems are found during implementation, it is necessary to stop implementing and return to the design phase. A complete redesign is seldom necessary, but this restriction prevents tweaking of the code which soon leads to an unmaintainable system. The restriction is compatible with the spiral design approach of software engineering [16].

The conceptual schema is used to create a logical schema for the database based on the database management system to be used. It is also used to develop the domain object schema for the

application. Operations and constraints defined in the document are used to develop the procedures on each domain object.

The user-interface mock-ups should be implemented with as little change as possible. Using a graphical user interface builder may aid in the implementation. It is important to not allow the user interface development to become a distraction. Often, developers spend a disproportionate amount of time working on the user interface. By implementing only the functionality and look of the mock-up, this problem can be controlled.

The dialogues are used to implement the interaction between the database and user-interface along with any computational functionality which is necessary for completion of a task. Well written dialogues provide a straightforward means of implementing interactions and provide a way to collect code into procedures, methods, or inheritance hierarchies as appropriate for the implementation language.

5. Conclusion

We have investigated methodologies for designing genome databases and found that:

- Building a database in stages reduces the amount of work to be performed at each stage and restricts it to the work which is appropriate.
- Documenting the database from three perspectives focuses attention separately to the vital aspects of the system.
- Designing a database in stages from multiple perspectives simplifies the overall design by decomposing the problem into tasks that support each other and the overall design.

6. Acknowledgments

The authors thank Andy Arenson, Bob Cottingham, Dan Davison, Wayne Parrott, Joanna Power, and Randy Smith for frequent discussions of the ideas in this paper. This work was supported by the W.M. Keck Center for Computational Biology, the Baylor Human Genome Center funded by the NIH National Center for Human Genome Research, a grant to C.B.L. from the Department of Energy, and fellowships to M.G. from the National Library of Medicine and from the Department of Energy.

7. References

1. Brodie, Michael, John Mylopoulos, and Joachim Schmidt, editors. *On conceptual modelling: Perspectives from artificial intelligence, databases, and programming languages* Springer-Verlag, New York, 1984.
2. Graves, M., Bergeman, E.R. and Lawrence, C.B. A Graph-Theoretic Data Model for Genome Mapping Databases. In the Proceedings of the Hawaii International Conference on System Sciences-28, Biotechnology Computing Track. January, 1995.
3. Chen, P P. “The entity-relationship model: toward a unified view of data,” *ACM Transactions on Database Systems* 1:1, pp. 9-36, 1976.
4. Ochs, R. A relational database model for metabolic information. In H. Lim, ed., *Proceedings of The Second International Conference on Bioinformatics and Genome Research.* World Scientific Publishing. June, 1994.

5. Chen, I-Min and Markowitz, Victor. An Overview of the Object Protocol Model (OPM) and the OPM Data Management Tools. Tech report LBL-PUB-33706, Lawrence Berkeley Labs, 1994.
6. Kasper, R.T. and Rounds, W.C. A logical semantics for feature structures. In *Proceedings of the 24th Annual Conference of the Association for Computational Linguistics*, pages 235-242, 1986.
7. Ait-Kaci, Hassan. *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially-Ordered Type Structures*. Ph.D. Thesis, Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1984.
8. Sowa, John. *Conceptual Graphs.* Addison-Wesley, 1984.
9. Nebel, Bernhard and Gert Smolka. Representation and reasoning with attributive descriptions. In K.H. Blasius, U. Hedstuck and C.R. Rollinger, editors *Sorts and Types in Artificial Intelligence*, volume 418 of LNAI, pages 112-139. Springer-Verlag, 1990
10. Gyssens, Marc, Jan Paradaens, and Dirk Van Gucht. *A graph-oriented object database-model.* In Proceedings of ACM Conference on Principles of Database Systems, 1990.
11. Consens, Mariano and Alberto Mendelzon. *The G+/GraphLog visual query system*. In Hector Garcia-Molina and H.V. Jagadish, editors, Proc of the 1990 ACM SIGMOD International Conference on Management of Data, page 388, May 1990.
12. Levene, Marc and Alexandra Poulovassilis. *The hypernode model and its associated query language.* In Proc Fifth Jerusalem Conference on Information Technology,pages 520--530, 1990.
13. Coleman, Derek, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes and Paul Jeremaes. *Object-Oriented Development the Fusion Method.* Prentice Hall, 1994.
14. Jacobsen, Ivar, Magnus Christerson, Patrik Jonsson and Gunnar Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach.* ACM Press, 1992.
15. Connell, John L., and Shafer, Linda Brice. *Structured Rapid Prototyping*. Yourdon Press, 1989.
16. Boehm, Barry. A spiral model of software development and enhancement. *IEEE Computer*, May 1988.