CHAPTER 6: CONCLUSION

6.1 Future of Systems Analysis, Design and Methodologies

There are an unlimited number of ways in which the methodologies discussed in the preceding six chapters might be compared and analyzed. In addition, significant research is proceeding on individual methods as well as on integrating different methods. To confuse matters, new technologies introduced daily profoundly impact our ability to develop applications and will require equally profound changes in methodologies to be used efficiently and effectively.

6.1.1 Research Relating to Analysis, Design and Methodologies

There are two growing bodies of research relating to methodologies and the application development process. The first research is attempting to reconcile the differences in methodologies to develop an improved hybrid. The second type of research studies the decision processed that occur in analysis and design activities.

Currently research in building hybrid methodologies is primarily applied. All authors, so far, are seeking to integrate OO notions and notations with some other methodology, including structured analysis, Jackson systems design, information engineering, and others. This research is purely prescriptive, of the form: "If I were going to put OO together with structured analysis, here's what I would do." While this research is promising, the lack of researcher attention to the differences in reasoning and thinking process of the methods needs to be resolved.
Also, some comments about easy and hard features of methodologies can be developed. The easy features of OO are those that automatically lead to information hiding, minimal coupling and maximal cohesion, the traceability of information throughout the process, and the essential continuity of the method (i.e., building tables and progressively adding details to the information). The hard OO features are the extensive experience in operating systems required to determine service object requirements and the significant coupling between the implementation language and the application design.

The easy features of Information Engineering (IE) are entity analysis, full-life cycle approach including enterprise through maintenance phases, the methods for deciding distribution, and the balanced thinking given to both data and processes. The hard IE features are the mental shift required to move from design to program specification and from an action diagram to its components. The decisions about the size and content of components are left to the Systems Engineer (SE).

The easy feature of Structured Analysis (SA) is the simplicity of the thought process which is easily grasped by most people. The hard SA features are the disjoint phase relationships moving from DFD to structure diagram and decomposing the structure diagram into modules. These actions, like similar one of IE, are left to SE skills and have few guidelines.

Two methods of analyzing methodology classes were used in this project. The first, the information systems methodology framework, was extended to include the characteristics of applications from Chapter 1 and the desirable characteristics of applications. From the analysis we know that both information engineering (IE) and object orientation (OO) are more complete in describing applications than structured analysis (SA), but each addresses different phases of the life cycle. IE is more complete in coverage of organization level information systems planning and analysis, both of which precede design and implementation. OO is more detail and programming-oriented, resulting in a deeper level
of design by the end of the design phase define the integration of object orientation, database, and storage technologies in one application environment.

6.2 Contributions

HYCONAN was developed to cope with the limitations found in the structured methods and OO methods.

There are some overlaps between the features of the compared methods. A better method can be developed by keeping the common features, adding new features and combining useful inherent features. This was the purpose of creating the prototype method, whose elements have been adopted as the building blocks of HYCONAN.

In HYCONAN, domain analysis is the means by which artifacts (i.e., classes, systems and processes) of the problem domain can be reused. Potentially reusable classes, systems, and processes are added to the library of reusable components, maintained during the software development process for future use in other projects within the same problem domain. MetaEdit's generic features were efficiently used to develop the HYCONAN case tool which encompasses the draw windows of HYCONAN's diagramming techniques, their reports and their help files.
6.3 Limitations and Future Research

This thesis has attempted to build a new hybrid SA/OO method by combining and refining the best features of the two methods. Moreover, new features have been added to cope with the limitations of these methods. However, as in any research effort, the contributions will never be perfect. The limitations of this research and future research directions are listed in the following:

- HYCONAN’s development process has addressed analysis, which has the most important impact. A lot of work can be added to address other activities of the software development process, such as design, implementation, maintenance and testing. Software engineering issues, such as project management, risk management, software quality assurance, software metrics, software measurement and OO frameworks can also be addressed.

- Developers may want to avoid rigorous mathematical precision as one extreme, or may want to provide correctness proof of the target system against the requirements as another extreme. Thus, another direction in future research is to integrate more formalism to HYCONAN’s software development process.

The details of the mapping of HYCONAN’s design deliverables to the target implementation system has not been addressed yet. Therefore, some research in this aspect can be done, for example, to map the design deliverables into OODBMS following the Object Database Management Group (ODMG) requirements.
6.4 Final Analysis

To summarize the application development in this project, we know that skills needed seem to vary by activity both across and within phases of a system development life cycle, that task domain facilitates the process of building a mental model of the problem solution, and that different types of domain knowledge exist, including methodology and task domains.

For Systems Engineers (SE), this research has several implications. First, the entire field of methodology research is in its infancy. As it matures, both the methods and the way we use them should be expected to change. Second, beyond hybrid methodology that attempts to integrate methodologies requiring different mental models of a problem, for instance, structured analysis and OO, are unlikely to be very productive. Rather, we need to identify which methodological orientation best fits different problem domains, concentrating on methodology improvement and use in the approximate domains.

Last, since methodologies do not provide complete analysis of all aspects of problem domains, by definition, CASE tools based on the methodologies will also provide partial task coverage. The more complete the methodology, the more complete the CASE tool. Some vendors add completing tasks to support, for example, code generation; these CASE tools are even more complete than those that are only methodology-based.

Applying Humphrey’s framework to research in Information Systems (IS), methodologies are in either initial stage or the defined stage. CASE tools help methodologies attain the defined stage, but sometimes impose such rigidity in doing so that usage is constrained and might not fit either the way SEs work or the work itself.