RT-UML

1. Principles of RT Embedded System Development

The current state-of-the-art in software development process relies on a small number of important principles. They are: -

1.1. Iterative Development

Targeting development toward the early reduction of risk by "drilling down" both to expose the risk and to mitigate it. To be effective, the rapid prototypes must be production-quality software; be small, focused pieces of the overall application; and address identified or perceived risks.

1.2. Model-Based Development

Large, complex systems can't be effectively constructed using only source-code-level constructs. Abstract models permit the developers to capture the important characteristics of the application and the ways they interrelate independent from low-level implementation concerns.

1.3. Model-Code Bidirectional Associativity

For model-based systems, it is absolutely crucial that the code and the diagrams are different views of the same underlying model. If the code is allowed to deviate from the design model, the separate maintenance of the code and model becomes burdensome, and the system eventually becomes totally code-based. As a result, system complexity can no longer be effectively managed.

1.4. Executable Models

You can test only things that execute, so build primarily executable things, both early and often. The key to this is model-based translation of designs so that transforming a design into something that executes takes on the order of seconds to minutes, not weeks to months using traditional hand-implementation approaches.

1.5. Debug and Test the Design Level of Abstraction

Because today's applications are extremely complex, we use abstract design models to help us understand and create them. We also need to debug and lest them at the same level. We need to be able to ask, "Should I put the control rod into the reactor core?" rather than merely, "Should I be jumping on C or NZ?"

1.6. Test What You Fly and Fly What You Test

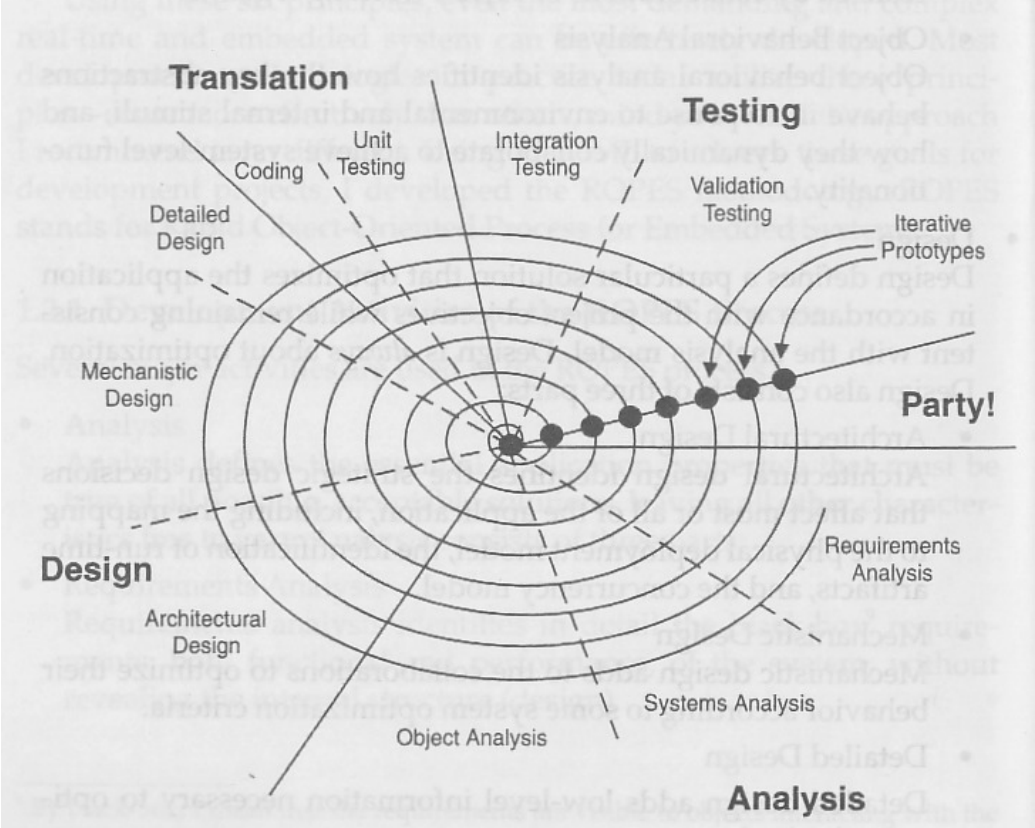
Simulation has its place, but the purpose of building and testing executable models is to quickly develop defect-free applications that meet all their functional and performance requirements. Using appropriate technology, you can get a the benefits of rapid iterative development and deployment, model-level debugging, and executable models using generated production-level software so that the testing needs to be done only once.

Using these six principles, even the most demanding and complex real-time and embedded system can be effectively developed. Most development methodologies in practice today utilize these

principles. Using the UML to achieve these goals for development projects, ROPES methodology has been developed. ROPES stands for Rapid Object-Oriented Process for Embedded Systems.

2. Development Activities of the ROPES Process

Several major activities are used in the ROPES process.



2.1. Analysis

Analysis defines the essential application properties that must be true of all possible, acceptable solutions, leaving all other characteristics free to vary. Analysis consists of three parts:

2.2. Requirements Analysis

Requirements analysis identifies in detail the black-box requirements, both functional and performance, of the system, without revealing the internal structure (design).

By black-box means that the requirements are visible to objects interacting with the system but do not require knowledge of the system's internal structure.

2.3. System Analysis

Systems analysis focuses on three activities: determining the optimal breakdown between hardware and software; creating a high-level system architecture; refinement and characterization of complex control algorithms Systems analysis is usually only performed on large or very complex systems. It is common for small or relatively simple systems to skip this step of analysis.

2.4. Object Analysis

Object analysis consists of two sub phases - object structural analysis and object behavioral analysis:

2.4.1. Object Structural Analysis

Object structural analysis identifies the key abstractions of the application that are required for correctness, as well as the relation that links them together. The black functional pieces are realized by collaborations of objects working together.

2.4.2. Object Behavioral Analysis

Object behavioral analysis identifies how the key abstractions behave in response to environmental and internal stimuli, and how they dynamically collaborate to achieve system-level functionality.

2.5. Design

Design defines a particular solution that optimizes the application in accordance with the project objectives while remaining consistent with the analysis model. Design is always about optimization. Design also consists of three parts:

2.5.1. Architectural Design

Architectural design identifies the strategic design decisions that affect most or all of the application, including the mapping to the physical deployment model, the identification of run-time artifacts, and the concurrency model.

2.5.2. Mechanistic Design

Mechanistic design adds to the collaborations to optimize their behavior according to some system optimization criteria.

2.5.3. Detailed Design

Detailed design adds low-level information necessary to optimize the final system.

2.6. Translation

Translation creates an executable application from a design model. Translation usually includes not only the development of executable code, but also the unit-level (that is, individual object) testing of that translation.

2.7. Testing

Testing applies correctness criteria against the executable application to either identify defects or to show a minimal level of accept ability. Testing includes, at minimum, integration and validation testing.

These activities may be arranged in many ways. Such an arrangement defines the development process used for the project.

The iterative development process model is known as ROPES, for Rapid Object-Oriented Process for Embedded System. Each iteration produces work product known as artifacts. A single iteration pass, along with the generated artifacts. This model is somewhat simplified in that it doesn't show the sub phases of analysis and design, but it does capture the important project artifacts and how they are created and used.

Rhapsody is an advanced model-creation tool with integrated, product-quality code generation and design-level testing capabilities built in. Rhapsody was developed specifically to aid in the development of real-time and embedded systems and integrates smoothly into the ROPES process model. Fully constructive fools, such as Rhapsody from I-Logix, assist by providing support for all the precepts mentioned at the beginning of this section. Although the UML, in general, and the ROPES process, in particular, can be applied using manual means for translation of models into code and debugging and testing that code, the use of such powerful automated tools greatly enhances their effectiveness. Table 1- shows how a fully constructive tool can aid in the generation of development-phase artifacts.

3. Ropes Artifacts

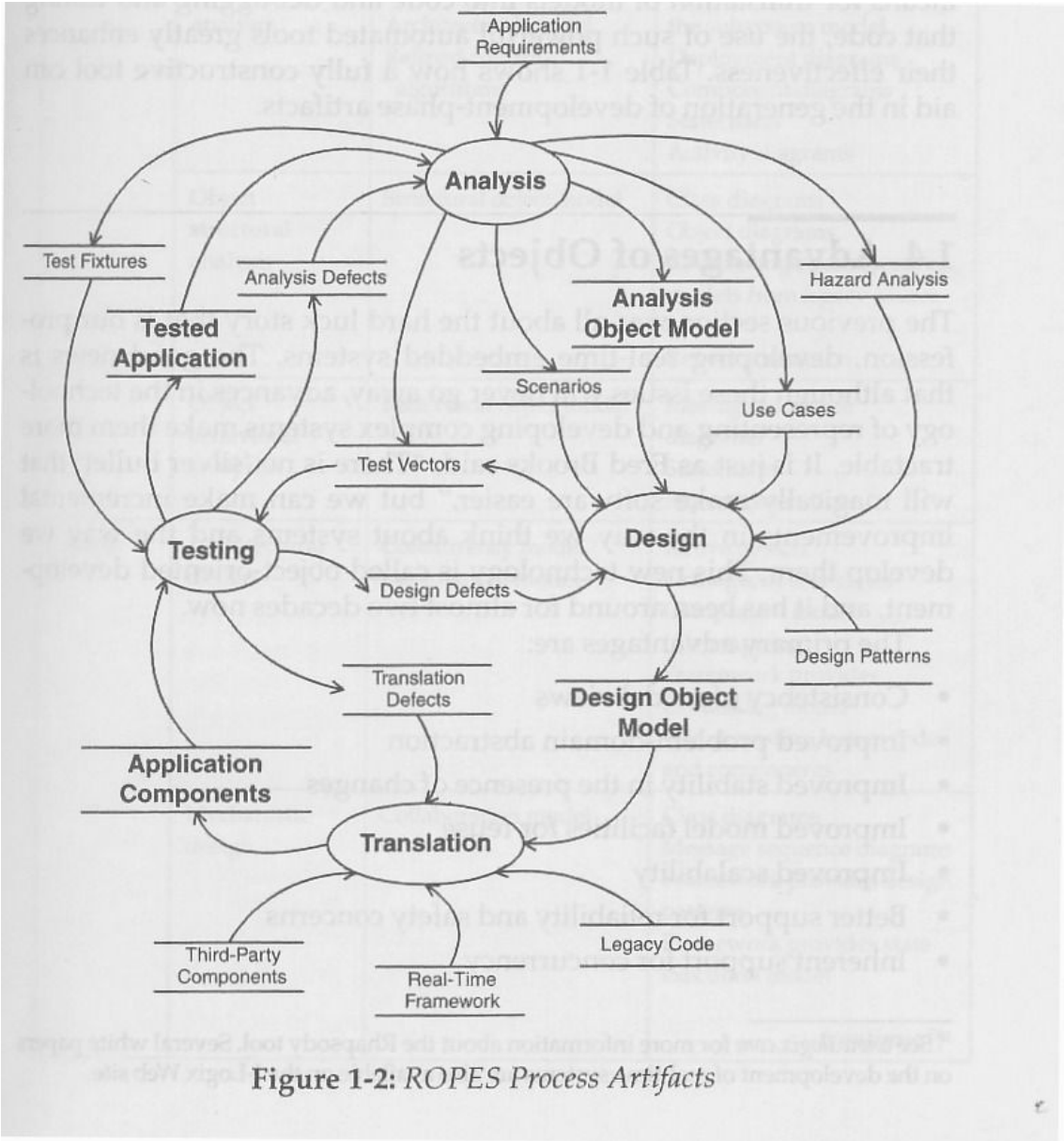


Figure 1-2: *ROPES Process Artifacts*

Phased Artifacts in the ROPES Process

Activity	Process Step	Generated Artifacts	Tool-Generated Artifacts
Analysis	Requirements analysis	? Use case model ? Use case scenarios	? Use case diagrams ? Use case descriptions ? Message sequence diagrams ? Report generation
	Systems analysis	? Initial high-level Architectural model ? Refined control algorithms	? Class diagrams represent the subsystem model ? Deployment diagrams ? Component diagrams Statecharts ? Activity diagrams
	Object structural analysis	? Structural object model	? Class diagrams ? Object diagrams ? Reverse engineering creates models from legacy source code ? Report generation
	Object behavioral analysis	? Behavioral object model	? Message sequence diagrams ? Statecharts ? Report generation
Design	Architectural design	? Concurrency model ? Deployment model	? Active objects ? Orthogonal and-states Component model (file mapping) ? Framework provides OS-tasking model ? Use of existing legacy code and components
	Mechanistic design	? Collaboration model	? Class diagrams ? Message sequence diagrams ? Framework provides design patterns ? Framework provides state execution model
	Detailed design	? Class details	? Browser access to: - Attributes - Operations - User-defined types - Package-wide members

			? Round-trip engineering updates model from modified source code

Translation		? Executable application	? Fully executable code generated from structural and behavioral models, including: - Object and class diagrams - Sequence diagrams - Statecharts
Testing	? Unit testing ? Integration testing ? Validation testing	? Design defects ? Analysis defects	? Design-level debugging and testing on either host or remote target, including: - Animated multithreaded applications - Animated sequence dia grants - Animated statecharts - Animated attributes in brow ser - Breakpoints on: ✍ operation execution ✍ state entry or exit ✍ transition ✍ Event insertion ✍ Execution control scripts ? Simultaneous debugging with other design-level tools (such as Rhapsody from I-Logix) and source-level debuggers.