THE *CATALYST* SOFTWARE ENGINEERING ENVIRONMENT

H. E. Romanowsky **S. L. Mulholland**

Collins Commercial Avionics
Advanced Technology and Engineering
Rockwell International
400 Collins Road NE
Cedar Rapids, IA. 52498

Abstract

Advanced Technology and Engineering of Rockwell International, Collins Commercial Avionics, has an ongoing effort to develop an integrated software engineering environment, named Catalyst, *which supports both government and commercial projects within the Avionics Divisions headquartered in Cedar Rapids, Iowa. The name* Catalyst, *a word synonymous with the word* stimulus, *was chosen because the use of the environment-related technology within* Catalyst *stimulates a change in the way that software is developed — it provides the motivation for adopting techniques, methods, and processes which result in better engineered software. This paper focuses on some of the challenges in* Catalyst's *development, its capabilities, and benefits of its use.*

Introduction

Integrated software engineering environments (ISEEs) are needed to assist software developers and managers in light of the increasing complexity of avionics systems software, continually increasing software development related costs, rising demand for high quality software, and the necessity for the application of discipline within software development life cycle activities. The use of an application domain supportive ISEE can be a market discriminator with respect to achieving significant quality and productivity gains while minimizing cost and needed resources.

In the past, software engineering tool support has had a tendency to be a collection of tools which functionally relates to a narrow portion of the overall software development life cycle, primarily in the implementation phase. Current trends are to provide a comprehensive solution to the needs of software development teams. Emphasis is now being placed on providing capabilities for the entire software life cycle and for all project personnel. In the past, tools might have been purchased and then not used because of the lack of training on them or lack of defined work context, or process, within which to use them. Industry is currently advocating the provision of tool specific training, as well as training in the procedures of using a tool or set of tools with respect to the process used by a project.

In 1989, Advanced Technology & Engineering, Collins Commercial Avionics, established the Software Engineering Environment (SEE) section whose charter is to work with and transfer to users appropriate environment related technology. A key effort for the SEE section is the definition, implementation, and support of the *Catalyst* software engineering environment.

Background and Goals

Prior to the start of work on *Catalyst*, specific goals and assumptions were made for its development and implementation. Many of them came as a result of foundation work done in the 1985–1989 time frame.[1] It has always been an assumption that tool usage should be coupled with a definition of the process for how to apply those tools to a specific development. Some of the major goals are listed below.

- *Catalyst* is to be platform independent, but targetted to workstation hosts.
- *Catalyst* is to provide a common, environment-controlled user interface.
- The environment is to be extensible so as to accommodate an evolutionary growth path.
- *Catalyst* must be useable by projects for production use at incremental stages of its evolution.

- Use is to be made of Commercial-off-the-Shelf (COTS) products as much as possible and as much as is appropriate.
- There is to be support of a heterogeneous approach which takes advantage of the current resources and eventually supports truly distributed development.
- Tailorability should be provided so as to facilitate support for divisions' software processes from both platform and tool points of view.
- *Catalyst* use should be supported with training, consulting, and follow-up as needed.
- The environment should support multiple software development standards, within both commercial and government business areas.
- *Catalyst* should make use of industry or de facto standards where applicable.

The overall goal of the *Catalyst* environment effort is to provide a tailorable, platform independent, extensible vehicle to be used during the software life cycle activities. Some of the goals can be achieved in the short term and others are used as consistent guidelines in planning for future versions of *Catalyst*. Successful use of the environment is predicated on providing adequate and timely training with respect to methodologies, tools, and their use within the context of *Catalyst*.

Early Decisions and Definition

Development of a software engineering environment that covers the entire software development life cycle is a time consuming and labor intensive undertaking. Therefore, it was decided early in the Catalyst development program, that the largest, initial, positive impact would occur if the often times hardest and least automated aspects of the software life cycle — requirements analysis and design — were addressed first, particularly with emphasis on:

- Providing support via *Catalyst* for project managers to establish a consistent approach for their team members to use in development and documentation activities.
- Automatic generation of DOD-STD-2167A[2] Data Item Deliverables (DIDs) that result from engineering work done for the software requirements analysis and preliminary design activities.
- Providing support that would encourage the reuse of project artifacts.

Considering the tools already in house to assist with coding and debugging, it was felt that projects could continue with the way in which they were accustomed to performing implementation and testing tasks, but still take advantage of new capabilities in the early activities of the software life cycle.

In conjunction with *Catalyst's* development, it was necessary to determine how the current software engineering practices, processes, and tools could be accommodated so as to make the transition to *Catalyst* use as non-disruptive as possible. An approach that was adopted was to structure *Catalyst* so that the whole environment or a subset of its capabilities could be introduced on a project.

Version 1 Series

Work on *Catalyst* began under the premise that its implementors would have between 6 to 12 months to put together its initial capabilities before any of it would be put to production use. In reality, a program within the Collins Avionics and Communications Division started to use initial versions of *Catalyst* about three months into the environment's development.

Unexpected early use of the environment led to an initial version 1.0 capability which exceeded initial expectations, but has caused an accelerated rapid prototyping approach to environment development as more projects have been signing up to use *Catalyst* along the way. One of the biggest challenges to the *Catalyst* team is the provision of support to actual users during the environment planning, development and implementation efforts. This experience has provided an invaluable production test of the process and capabilities associated with the environment.

The version 1 series of *Catalyst* will provide full life cycle coverage. Version 1.0 of *Catalyst* was completed in December 1990. Version 1.1 will follow later this calendar year.

Catalyst Supported Activities

Catalyst support covers four aspects — engineering process, requirements management process, document generation, and tools, both COTS and in-house developed, which enforce the processes and automate document generation. This section briefly identifies the major software development life cycle activity capabilities supported by the version 1 series of *Catalyst*.

Software Requirements Analysis Support.

1. Requirements Engineering.

- Verification: specifically addressing consistency, correctness, completeness, and static performance.
- Automatic Generation of Software Requirements Specification (SRS) and Interface Requirements Specification (IRS).

2. Requirements Management

- Verification: specifically addressing consistency, correctness, and completeness.
- Tracking and Traceability at Computer Software Configuration Item (CSCI) and System levels.
- Automatically documented in the SRS for CSCI and System levels.

3. Requirements Qualification

- Verification: specifically addressing consistency, correctness, and completeness.
- Automatically documented in the SRS at the CSCI level only. The System level is documented in the Software Test Plan.

4. Project Management

- Verification of process usage.
- Automatic generation of SRS and IRS supports incremental, frequent generation of documents which is utilized for project progress tracking.
- Support for Change Impact Analysis. All source and destination occurrence pairs of data elements and interfaces are automatically and dynamically generated and documented in the SRS. All possible context views of requirements are automatically and dynamically generated and documented in the SRS on both a CSCI and System-level basis.

Preliminary Design Activity Support.

1. Preliminary Design Engineering

- Implements process that supports projects of varying sizes and is not development language specific
- Automatically documented in Software Design Document (SDD), version 1, and the Interface Description Document (IDD).

2. Requirements Transition

- Controls requirements transition from software requirements analysis activity to preliminary design activity.

3. Requirements Management

- Verification: specifically concerning consistency, correctness, and completeness.
- Tracking and Traceability of development objects and document objects.
- Automatic documentation in SDD, version 1.

4. Project Management

- Verification of process usage.
- Automatic generation of SDD, version 1, and IDD supports incremental, frequent generation of documents which is utilized for project progress tracking.
- Support for Change Impact Analysis. The controls placed on requirements transition facilitate the identification of the extent of impact upon review of a change request.

Detailed Design Activity Support.

1. Detailed Design Engineering

- Implements a process which supports projects of varying sizes and which are to use either Ada or C for source code implementation.

- Automatically documented in SDD, version 2.

2. Requirements Management
- Verification: specifically for consistency, correctness and completeness.
- Tracking and Traceability of development objects and document objects.
- Automatic documentation in SDD, version 2.

3. Project Management
- Verification of process usage.
- Automatic generation of SDD, version 2, supporting incremental, frequent generation of documents which is utilized for project progress tracking.
- Support for Change Impact Analysis. The requirements management process facilitates the identification of all objects impacted by a specific change request.

Code and Unit Test Activity Support.

1. Uniform code creation.
- Provides common native code generation for Ada or C source.
- Allows each project to insert the specific target code generator required by their project.

2. Requirements Management
- Verification: specifically for consistency, correctness, and completeness.
- Tracking and Traceability for development objects and document objects.

3. Automatic Generation of White Box test cases based on the Computer Software Unit (CSU) and/or Computer Software Component (CSC) definition.
- Facilitates quality review and verification of code prior to moving to target test environment.

4. Project Management
- Verification of process usage.
- Automatic generation of Software Development Files (SDF) and SDF artifacts, facilitating configuration management.
- Support for Change Impact Analysis.

Platforms.

The initial host platform for version 1.0 is the Apollo workstation running BSD 4.3 Unix. Porting was then done to the Sun Sparcstation, again running BSD 4.3 Unix and then to the DEC VAXstation 3100 utilizing VMS. (Plans are underway to port to the DEC DECstation running Ultrix).

Catalyst Development Process Definition

In order to achieve the amount of automation, documentation, and verification that has been accomplished in the first releases of *Catalyst*, a development process had to be established whose use would result in the uniform, consistent development of high quality project artifacts. In developing this process, engineering, testing, and documentation objectives were reviewed to ascertain that the defined process did not ignore requirements for the accomplishment of typical project objectives. From this effort, a preliminary development process was defined.

This process definition was then analyzed from the viewpoints of both commercial and government projects. What was discovered is that the engineering objectives for commercial and government projects are essentially the same. Some differences exist in the testing objectives due to the application of more or less rigorous verification requirements. However, most of the differences existing between commercial and government projects occurred in the area of documentation. Upon closer review, it was found that these differences centered more in the area of document format rather than content. Because the proposed development process identifies and manipulates data based on its object classification, and the process exhibited the qualities of flexibility and tailorability, it was found to be appropriate for use in both commercial and government projects.

Next, the proposed process definition was analyzed to determine its applicability to multiple application domains, project sizes and source language implementations. After ensuring that the process provided support for all these issues, the definition was finalized and adopted as the *Catalyst* software development process. This process is documented in the *Catalyst Environment Reference Manual.* Projects that utilize this development process gain the full benefits of *Catalyst.* Projects that partially utilize it, or are too far into development to begin use of *Catalyst,* still benefit from the many "stand–alone" capabilities supplied by *Catalyst.*

Next Generations

Although the goal for *Catalyst* is that it be fully integrated, the first releases of the environment provide only localized integration with respect to data exchange between tools. The key considerations for the next generation of the *Catalyst* environment is in the areas of integration and user interface. In addition to these areas, there must be decisions made as to the applicable ISEE standards to consider for version 2 and beyond.

Standards to investigate are in a variety of areas. For example, MIL–STD–1838A, PCTE+, ATIS (or CIS), and several other efforts target tool integration; DIANA and IRIS target data structure representation; and MOTIF and Open Look target a common window presentation definition. Attention must also be given to emerging ISEE standards. Two such emerging standards are the National Institute of Standards and Technology (NIST) ISEE Reference Model[3] and the European Computer Manufacturers Association (ECMA) Task Group TC33/TGRM Reference Model[4]. Another effort which may be considered to have major influence in the establishment of ISEE standards is the government sponsored Software Technology for Adaptable, Reliable Systems (STARS) program.

Catalyst, version 2, will remove the tool–to–tool interface and replace it with a tool–to–framework interface. It will also remove the tool–to–platform interface and replace it with a framework–to–platform interface. The implementation of an environment user interface that separates the users from direct communication with the tools will be the first step towards the full implementation of a user interface generator tool.

With *Catalyst,* version 2, in place, the next evolutionary step will be to implement a user interface generator tool and to transition the environment user interface to its control. This step also requires the transitioning of the COTS products' user interface control to this ISEE component. Because of the technical and societal complexities involved in achieving this ISEE characteristic it is unclear whether or not this will be directly achievable in a "near" timeframe. If not, *Catalyst,* version 3 will embody the maximum degree of support possible for this ISEE characteristic.

Catalyst's evolutionary environment development plan allows Rockwell's projects to immediately benefit from the use of a SEE and supports *Catalyst*'s continued growth towards full implementation of a standard ISEE concept.

User Support

A software engineering environment can not be easily and successfully transitioned into use without adequate user support. This is a key assumption in the SEE section. There are many ways that assistance is given to both new and current users. They are briefly summarized in the paragraphs below.

Documentation. The main documentation medium is the *Catalyst Environment Reference Manual.* It includes engineering development procedures, document generation and verification procedures, implementor's guides to assist with project specific tailoring, and model definitions. It is divided into life cycle activities and tool sets.

Training. Members of the SEE section provide training for tool use within the *Catalyst* process, as well as for individual tools. Whenever possible, training for methodologies, workstation specific training, and tool specific training of a general nature is provided via the Collins Avionics Divisions' Software Engineering Training Program.

Consultation. SEE group personnel provide initial demonstrations of *Catalyst* capabilities to users, discuss the application of methodologies within the environment to projects, and assist with transitioning from previously used tools and techniques to those within *Catalyst*. Tailoring *Catalyst* to a particular project's needs is also handled via consultation.

Catalyst User's Group. Recently, a user's group was started in order to acquaint the users with each other and to facilitate discussion concerning how various users applied the environment's capabilities to their projects. The group is also to be used as a way for the SEE section members to discuss future plans and priorities with active users.

Summary

The *Catalyst* software engineering environment provides a workstation–based approach which gives language independent support for software requirements analysis and software preliminary design activities; it then narrows its scope to the support of Ada and C specific implementation work for the rest of the life cycle. *Catalyst* currently provides a set of loosely integrated tools which assist project members in their work as opposed to just using point solutions.

A significant benefit that has already been demonstrated by several Rockwell projects is that even partial usage of *Catalyst* by one project results in the opportunity for project artifact reuse by other projects. This opportunity for software development artifact reuse has been taken many times by new Rockwell projects; and has, in turn, encouraged those new projects to utilize *Catalyst* in their development.

Users of *Catalyst* have received benefits in the areas of automated requirements engineering, requirements management, software development process, and project management as a result of using version one of *Catalyst*. There have been several production projects, both from the government and commercial sectors of the Avionics Divisions, using version one and it has been very well received.

It is emphasized that *Catalyst* is not only tools, but also involves the procedures and training which support its use. Users have assistance with their learning curve with respect to how to use the environment capabilities within the context of the work that they do. *Catalyst* provides a baseline for commonality for software projects — across host platforms, across software development projects and assists with reuse.

References

1. Romanowsky, H. E., *Ada Programming Support Environment Development Plan, Version 1.0*, Rockwell International internal document, August 1985.

2. U. S. Department of Defense, *Defense System Software Development*, DOD–STD–2167A, 29 February 1988.

3. Wong, W.,"Summary of the 4th Workshop on Integrated Software Engineering Environments (ISEE), NIST, October 30, 1990.

4. Earl, A., ECMA Task Group TC33/TGRM's "A Reference Model for Computer Assisted Software Engineering Environment Frameworks," version 4.0, August 17, 1990.