



Performing Verification and Validation in Architecture-Based Software Engineering

Edward A. Addy
Logicon Advanced Technology

Verification and validation (V&V) now is performed during application development for many systems, especially safety-critical and mission-critical systems. The application system provides the context under which the software artifacts are validated. This article describes a framework that extends V&V from an individual application system to a product line of systems that are developed within an architecture-based software engineering environment. The product line architecture provides the context for evaluation in this approach.

Introduction

The implementation of architecture-based software engineering not only introduces new activities to the software development process, such as domain analysis and domain modeling, it also impacts other activities of software engineering including configuration management, testing, quality control, and verification and validation. Activities in each of these areas must be adapted to address the entire domain or product line rather than a specific application.

V&V methods are used to increase the level of assurance of critical software, particularly that of safety-critical and mission-critical software. Software V&V is a systems engineering discipline that evaluates software in a systems context [1]. The V&V methodology has been used in concert with various software development paradigms, but always in the context of developing a specific application system. However, an architecture-based software development process separates domain engineering from application engineering in order to develop generic reusable software components that are appropriate for use in multiple applications.

The earlier a problem is discovered in the development process, the less costly it is to correct the problem. To take advantage of this, V&V begins verification within system application development at the concept or high-level requirements phase. However, an architecture-based software development process has tasks that are performed earlier — possibly much earlier — than high-level requirements for a particular application system. In order to bring the effectiveness of V&V to bear within an architecture-based software development process, V&V must be incorporated within the domain engineering process.

On the other hand, it is not possible for all V&V activities to be transferred into domain engineering, since verification extends to the installation and operation phases of development, and validation is primarily performed using a developed system. This leads to the question of which existing and/or new V&V activities would be more effectively performed in domain engineering rather than in — or in addition to — application engineering. Related questions include how to identify the reusable components for which V&V at the domain level would be cost-effective, and how to determine the level to which V&V should be performed on the reusable components.

Differences Between V&V and Component Certification

Much work has been done in the area of component certification, which is also called evaluation, assessment, or qualification. These terms can have slightly different meanings, but refer in general to rating a reusable component against a specified set of criteria. Reuse libraries often use levels to indicate the degree to which the library has evaluated a component. The Asset Source for Software Engineering Technology (ASSET) library and the Army Reuse Center library both have four levels of certification, although the use of the term “levels” is operationally different in the two libraries [2]. Component-based libraries evaluate reusable components against criteria such as reusability, evolvability, maintainability, and portability, as well as expending various levels of effort to ensure the component meets its specification. Other schemes for component certification include the certification framework developed by the Certification of Reusable Software Components Program at Rome Laboratory [3], and the suitability testing performed by the National Product Line Asset Center on behalf of the Air Force Electronic Systems Center [4].

The common thread through all of these certification processes is the focus on the component rather than on the systems in which the component will eventually be (re)used. Michael Dunn and John Knight [5] note that with the exception of the software industry, customers purchase systems and not components. Ensuring that components are well designed and reliable with respect to their specifications is necessary, but not sufficient, to show that the final system meets the needs of the user. Component evaluation is but one part of an overall V&V effort, analogous to code evaluation in V&V of an application system.

Another distinction between V&V and component certification is the scope of the artifacts that are considered. While component certification is primarily focused on the evaluation of reusable components (usually code-level components), V&V also considers the domain model and the generic architecture, along with the connections between domain artifacts and application system artifacts. Some level of component certification should be performed for all reusable components, but V&V is not always appropriate. V&V should be conducted at the level determined by an overall risk mitigation strategy.

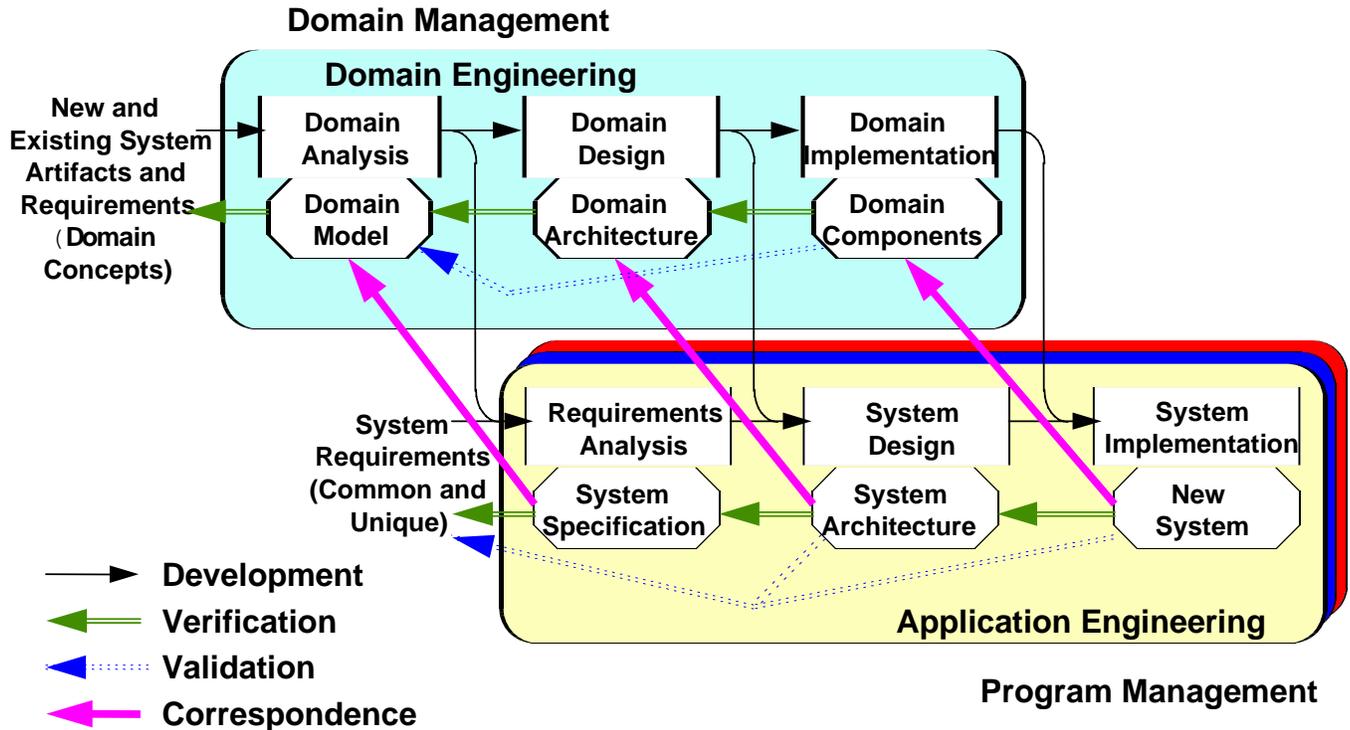


Figure 1. Framework for V&V in architecture-based software engineering.

Framework for Performing V&V within Architecture-based Software Engineering

A draft framework for performing V&V within architecture-based software engineering is formed by adding V&V activities to a two life cycle model involving both domain engineering and application engineering. The application-level V&V tasks described in IEEE STD 1012 [6] serve as a starting point. Domain-level tasks are added to link life cycle phases in the domain level, and transition tasks are added to link application phases with domain phases. This draft framework was refined by a working group at Reuse '96 [7], and the resultant framework is shown in Figure 1.

Domain-level V&V tasks are performed to ensure that domain products fulfill the requirements established during earlier phases of domain engineering. Transition-level tasks provide assurance that an application artifact correctly implements the corresponding domain artifact. Traditional application-level V&V tasks ensure the application products fulfill the requirements established during previous application life cycle phases. More details on the framework than allowed by the space of this article can be found in [8].

Performing V&V tasks at the domain and transition levels will not automatically eliminate any V&V tasks at the application level. However, reduction in the level of effort for some application-level tasks might be possible. The reduction in effort could occur in a case where the application artifact is used in an unmodified form from the domain component, or where the application artifact is an instantiation of the domain component through parameter resolution or through generation. Domain maintenance and evolution are handled in a manner similar to

that described in the operations and maintenance phase of application-level V&V. Changes proposed to domain artifacts are assessed to determine the impact of the proposed correction or enhancement. If the assessment determines that the change will impact a critical area or function within the domain, appropriate V&V activities are repeated to assure a correct implementation.

Although not shown as a specific V&V task for any particular phase of the life cycle, criticality analysis is an integral part of V&V planning. Criticality analysis is performed in V&V of application development in order to allocate V&V resources to the most important (i.e. critical) areas of the software [9]. This assessment of criticality and the ensuing determination of the level of intensity for V&V tasks also are crucial within architecture-based software engineering. Not all domain products will be used in critical application systems, and some of those used in critical application systems may not be in a critical area of the software. Some reusable components may be used in multiple systems, but may be a part of the critical software in only one or two of the systems. V&V should be performed only on domain products that are involved in the critical software in one or more application systems, and V&V tasks should be performed at a level of intensity appropriate to the level of criticality.

Determining the domain products for which to perform V&V, and the appropriate level of intensity for the V&V tasks, is complicated by the use of the products in multiple systems, some of which may only be in early stages of planning. If a component is used in only one critical application system, it may be more cost-effective to perform V&V during application engineering for that system rather than during domain engineering. Extension of criticality analysis from application engineering to domain engineering is an important area of this framework.

V&V of Domain Artifacts

Many of the same justifications for performing V&V in a product line that includes critical systems also apply to V&V of general purpose reusable components. These general purpose components include domain artifacts for systems that are not critical, as well as reusable components that are developed for general usage rather than for a specific product line. The Component Verification, Validation, and Certification Working Group at WISR 8 found four considerations that should be used in determining the level of V&V of reusable components [10]:

- Span of application — the number of components or systems that depend on the component
- Criticality — potential impact due to a fault in the component
- Marketability — degree to which a component would be more likely to be reused by a third party
- Lifetime — length of time that a component will be used

The domain architecture serves as the context for evaluating software components in a product-line environment. However, this architecture may not exist for general use components. The working group determined that the concept of validation was different for a general use component than for a component developed for a specific system or product line. In the latter case, validation refers to ensuring that the component meets the needs of the customer. A general use component has not one customer, but many customers, who are software developers rather than end-users. Hence validation of a general use component should involve the assurance — and supporting documentation — that the component satisfies a wide range of alternative usages, rather than the specific needs of a particular end-user.

Related Work

Although work is lacking specifically in the area of V&V as applied to architecture-based software engineering, there is related work that is applicable to some of the tasks within the framework. Component certification was discussed in a previous section, and this work is certainly applicable (although not sufficient) for V&V activity at the domain level. The analysis of architectures is the focus of attention and discussion [11, 12], but there is not as yet consensus on methods and approaches and much of this work is directed toward system architectures rather than product line architectures. One of the approaches being researched is a scenario-based analysis approach, Software Architecture Analysis Method [13]. In the area of correspondence tasks, the Centre for Requirements and Foundations at Oxford is developing a tool (TOOR) to support tracing dependencies among evolving objects [14].

Future Work

An initial, high-level framework for performing V&V in architecture-based software engineering has been developed. Once completed, this framework will allow the V&V effort to be amortized over the systems within a domain or product line. However, this framework is an outline with few details. V&V tasks that now are performed at the application level need to be

adapted for the domain level, and traceability tasks need to be adapted for the transition level. New methods not used on applications but appropriate for domain models or architectures need to be considered. Since V&V should be performed as part of an overall risk mitigation strategy within the domain or product line, methods of domain criticality analysis need to be developed, with attention paid to support from emerging architecture description languages. The methods identified need to be validated by use in projects having an architecture-based software engineering approach to producing applications that require V&V. ♦

About the Author



Edward A. Addy is currently a project manager with Logicon Advanced Technology. The work on which this article is based was done while Addy was a research associate with the NASA/WVU Software Research Laboratory, a cooperative effort between West Virginia University and the NASA Ames Software IV&V Facility in Fairmont, W. Va. His research interests are in the areas of IV&V, software product lines, component-based software reuse, software safety, and risk analysis. Addy is a doctoral candidate in computer science at West Virginia University.

Logicon Advanced Technology
2003 Apalachee Parkway
Suite 211
Tallahassee, Fla. 32301
Voice: 850-219-8033
Fax: 850-219-8034
E-mail: eaddy@logicon.com
Internet: <http://research.ivv.nasa.gov/~eaddy>

References

1. Wallace, Dolores R. and Roger U. Fujii, "Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards." NIST Special Publication 500-165, National Institute of Standards and Technology, Gaithersburg, Md. 1989.
2. Poore, J.H., Theresa Pepin, Murali Sitaraman, and Frances L. Van Scoy, "Criteria and Implementation Procedures for Evaluating Reusable Software Engineering Assets." DTIC AD-B166803, prepared for IBM Corporation Federal Sectors Division, Gaithersburg, Md. 1992.
3. Software Productivity Solutions Inc., "Certification of Reusable Software Components, Vol. 2 - Certification Framework." Prepared for Rome Laboratory/C3CB, Griffiss AFB, N.Y. 1996.
4. Unisys, Valley Forge Engineering Center, and EWA Inc., "Component Provider's and Tool Developer's Handbook." STARS-VC-B017/001/00, prepared for Electronic Systems Center, Air Force Material Command, USAF, Hanscom AFB, Mass. 1994.
5. Dunn, Michael F. and John C. Knight, "Certification of Reusable Software Parts." 1993 Technical Report CS-93-41, University of Virginia, Charlottesville, Va.

6. IEEE STD 1012-1986 (R 1992), IEEE Standard for Software Verification and Validation Plans, Institute of Electrical and Electronics Engineers, Inc., New York, N.Y.
7. Addy, Edward A., "V&V Within Reuse-Based Software Engineering." In proceedings of the Fifth Annual Workshop on Software Reuse Education and Training, Reuse '96, <http://www.asset.com/WSRD/conferences/proceedings/results/addy/addy.html>. 1996.
8. Addy, Edward A., "A Framework for Performing Verification and Validation in Reuse-Based Software Engineering." *Annals of Software Engineering*, Vol. 5, 1998.
9. IEEE STD 1059-1993, IEEE Guide for Software Verification and Validation Plans, Institute of Electrical and Electronics, Inc., New York, N.Y.
10. Edwards, Stephen H. and Bruce W. Wiede, "WISR8: 8th Annual Workshop on SW Reuse." *Software Engineering Notes*, 22, Sept. 5, 1997, pp 17-32.
11. Tracz, Will, "Test and Analysis of Software Architectures." In proceedings, International Symposium on Software Testing and Analysis (ISSTA '96), ACM Press, New York, N.Y, pp 1-3, 1996.
12. Garlan, David, "First International Workshop on Architectures for Software Systems Workshop Summary." *Software Engineering Notes*, 20, July 3, 1995, pp 84-89.
13. Kazman, Rick, Gregory Abowd, Len Bass, and Paul Clements, "Scenario-Based Analysis of Software Architecture." *IEEE Software*, 13, Nov. 6, 1996, pp 47-55.
14. Goguen, Joseph A., "Parameterized Programming and Software Architecture." In proceedings of the Fourth International Conference on Software Reuse, IEEE Computer Society Press, Los Alamitos, Calif., pp 2-10, 1996.

The Data & Analysis Center for Software (DACS) announces another new technical report

"Using Defect Tracking and Analysis to Improve Software Quality"

This state-of-the-art report discusses five defect categorization and analysis efforts from four different organizations. The analysis efforts at these organizations generally focus on one of three goals: finding the nature of defects, finding the location of defects, and finding when the defects are inserted. The intent is to use this information to characterize or analyze the environment or a specific development process. The report also presents some suggestions for how companies could begin or expand their defect classification efforts.

This report may be viewed free on the Internet or downloaded for free in PDF at: <http://www.dacs.dtic.mil/techs/defect/>

A bound, hard copy of this report, is available for \$50 and may be ordered from the DACS product order form at: <http://www.dacs.dtic.mil/forms/orderform.shtml> or by calling the DACS at (800) 214-7921.