Certification Authorities Software Team (CAST)

Position Paper CAST-25

CONSIDERATIONS WHEN USING A QUALIFIABLE DEVELOPMENT ENVIRONMENT (QDE) IN CERTIFICATION PROJECTS

COMPLETED SEPTEMBER 2005

(Rev 0)

Considerations When Using a Qualifiable Development Environment (QDE) in Certification Projects

Executive Summary

Some applicants are proposing the use of Qualifiable Development Environments (QDE) to improve development efficiency and quality. These QDEs are software tools that support activities to satisfy DO-178B/ED-12B objectives [1][1]. This paper presents certification concerns when a QDE is used to develop airborne software to comply with DO-178B/ED-12B objectives. To more easily identify and highlight these concerns, a generic QDE[2] is used in this paper as an example to illustrate QDE tool use. This generic QDE uses a graphical modeling technique to describe the software requirements and generates source code from this description using a Qualifiable Code Generator (QCG). Use of this example QDE[3] is examined and certification concerns identified.

The main purpose of this paper is to highlight certification concerns. At a very high level, these concerns may be summarized as follows:

- Use of a QDE must be well planned and documented. All planning documents should be submitted to the certification authority for approval early in the project.
- Applicants and certification authorities must fully understand the QDE model and its limitations.
- The qualification of the QDE tool itself including the code generator must be evaluated in the context of a specific project. In addition, a QDE is usually used with a symbol library and/or other source code developed outside the QDE. As such, credit claimed for complying with a DO-178B/ED/12B objective(s) will vary from project to project. That is, any claim for "Full" or "Partial" credit should be evaluated on a case-by-case basis.
- Use of a QDE does not relieve the applicant of complying with applicable DO-178B/ED-12B objectives and other software guidance applicable to a specific project.

A more detailed list of concerns appears in the Conclusion of this paper. Since the use of a QDE, including the code generator, is qualifiable on a project-specific basis only, even a detailed list cannot be assumed to be exhaustive and the airborne software must still be shown to satisfy all applicable DO178B/ED-12B objectives and other guidance, as applicable.

[1] Compliance to DO-178B objectives is discussed in this paper, however, the software approval basis for a specific project or program may include other guidance as well, such as Certification Review Items (CRI's), Issue Papers, etc. that impact the use and acceptance of QDE.
[2] Although this QDE is generic and is based on a real-world product, it is not intended to be representative of all QDE products.
[3] For the remainder of this paper, the term QDE encompasses the tool's full capability including, for example, the QCG.

A secondary purpose of this paper is to show that the need to verify the output of the coding process (DO-178B/ED-12B Table A-5) for the requirements implemented using the example QDE and its associated QCG may be reduced. However, all other source code, i.e., code not generated by the model including symbol libraries, should still be verified, and objectives for which "Full" credit is not received will still have to be met. Additionally, traceability of the generated source code from the example QCG to the object code should be assessed for Level A software, and testing and analyses must be performed (software integration testing, hardware/software integration testing, robustness testing, and system level testing, requirements coverage analyses, structural coverage analyses, scheduling and timing analyses, memory and stack usage analyses, etc.).

This paper addresses certification concerns with respect to compliance of the airborne software to DO-178B/ED-12B objectives. It assumes that the QDE is a qualifiable software development tool. Consequently, qualification of the tool itself is not considered in this paper. Additional information on qualification of automatic code generation tools [8] can be found in CAST-13 entitled "Automatic Code Generation Tools Development Assurance". The purpose of CAST-13 is to clarify DO-178B/ED-12B section 12.2.1.b. regarding the software level assigned to development tools and to discuss potential reduction of that software level [i.e., consider elimination/reduction of some objectives not applicable to ground-based, non-real time, non-critical software] relative to the airborne software.

1.0 Introduction

1.1 Model Based Development

Model-Based Development (MBD) is recognized by some in industry as an efficient and cost-effective way to develop safety-related embedded software. In the MBD environment, the software development process begins with the capture of software requirements using graphical tools that are "domain dependent" and graphical notations commonly used in the engineering community. In this paradigm, the model is defined by the software requirements described in a graphical notation. In order to fully benefit from this approach, automatic code generators are used to produce source code that will be compiled and linked with other manually written code. Some claim that this process is cost effective when the software must satisfy the objectives of DO-178B/ED-12B [1] for software Levels A and B and, especially, when a QCG is used to transform software requirements into source code.

Although mentioned in DO-178B/ED-12B (Section 12), the use of development tools (e.g., an automatic code generator) is just beginning to be commonly used and understood by many applicants in the context of DO-178B/ED-12B. Several aircraft manufacturers have obtained certification credit up to Level A when using such tools, both commercial (e.g., SCADE-KCG from Esterel Technologies) and proprietary (e.g., GALA from THALES Avionics) [5]. The objective of this paper is to highlight certification concerns when a generic QDE and its QCG are used to develop airborne software that complies with DO-178B/ED-12B objectives. This paper is intended to establish a common understanding of some typical issues when using a QDE and to provide information to certification authorities and industry about the use of such tools. Since this paper discusses use of a generic QDE and resulting issues, use of any QDE product should be evaluated on a project-by-project basis with the understanding that any benefits or limitations of using any specific QDE may differ from those described for this generic QDE.

1.2 Organization of Paper

The remainder of this paper presents an example of a generic QDE that relies on a graphical modeling technique and includes a QCG. The paper discusses how the QDE may be used in an example project to satisfy DO-178B/ED-12B objectives and how reduction of some verification activities may be claimed.

The approach for satisfying the objectives of DO-178B/ED-12B (and any other certification guidance) for all software in the airborne application should be established and coordinated with the appropriate certification authority(ies) early in the development life cycle. The applicant should specify the strategy for accomplishing compliance in the Plan for Software Aspects of Certification and other plans and standards. Each application must be evaluated on a case-by-case basis to ensure that all design assurance issues are addressed.

This paper is organized as follows:

- Section 2 presents the software development processes when an example QDE (including a QCG) is used. Certification concerns regarding the development processes are also highlighted in this section.
- Section 3 presents the software verification processes and potential claims that may be made by the applicant when an example QDE is used. Certification concerns regarding the verification processes are also highlighted in this section.
- Section 4 presents conclusions of this paper and summarizes certification concerns.
- Section 5 presents the references used throughout the paper.

2.0 Software Development Processes

This section considers the development processes and characteristics of an example QDE. The example QDE is intended to be sufficiently generic to provide general guidelines but specific enough for those guidelines to be relevant and applicable in real-world scenarios.

2.1 Software Development Processes in DO-178B/ED-12B

To begin, systems requirements are allocated to software (SR_1, …, SR_n). These requirements include the functional requirements of the software, its performance requirements, and its safety-related requirements. This is followed by the software requirements, design, and coding and integration processes.

2.1.1 Software Requirements Process

Within the software requirements process, system requirements allocated to software may be manually translated to software high-level requirements and described in the graphical notation of the example QDE. Other system requirements may be translated to software high-level requirements using natural language or some other notation. That is, they follow the typical requirements definition process. These latter requirements are not described using the graphical notation of the example QDE.

In addition, a system requirement (SR_i) may be refined into several software high-level requirements (HLR_i), as is the case with SR_3 in Figure 1 that is refined into HLR_2 and HLR_3. Furthermore, a given software high-level requirement may be the result of the refinement of several system requirements, as is the case of HLR_3 with SR_3 and SR_4. Finally, some "derived" software high-level requirements that are not directly obtained by refinement of a system requirement may also exist.

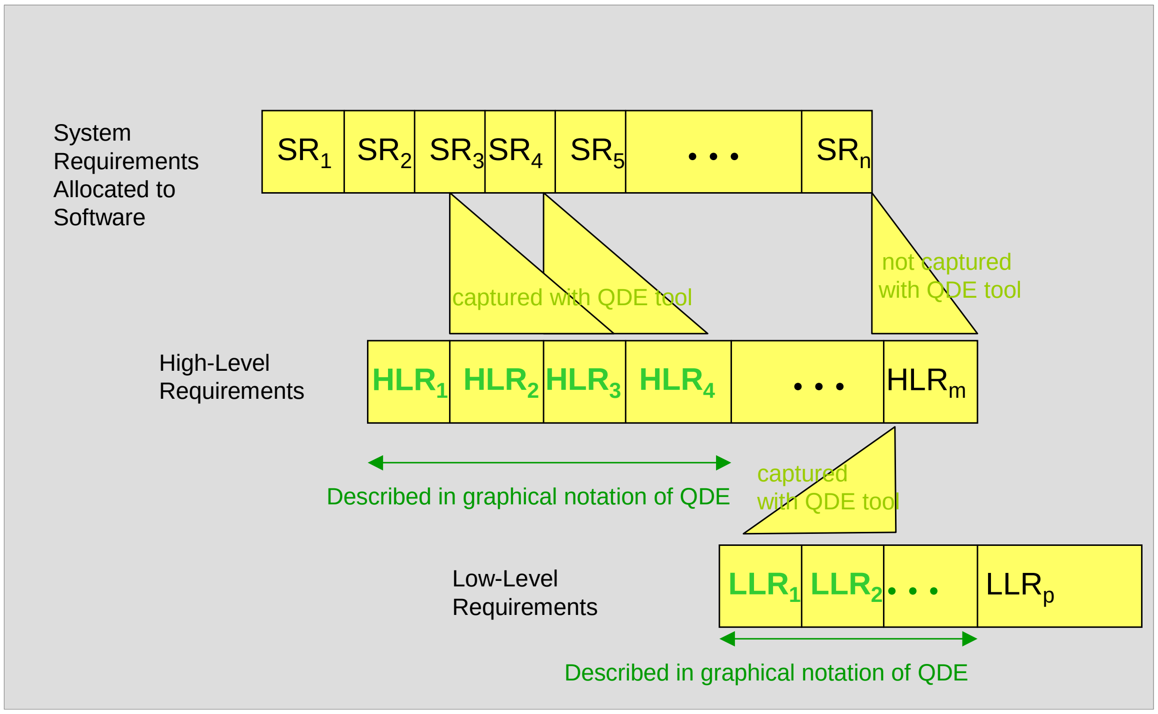

System Requirements Allocated to Software
SR1 SR2 SR3 SR4 SR5 . . . SRn
captured with QDE tool
not captured with QDE tool
High-Level Requirements
HLR1 HLR2 HLR3 HLR4 . . . HLRm
Described in graphical notation of QDE
captured with QDE tool
Low-Level Requirements
LLR1 LLR2 . . . LLRp
Described in graphical notation of QDE

Figure 1: Software Requirements Levels when using a QDE

2.1.2 Software Design Process

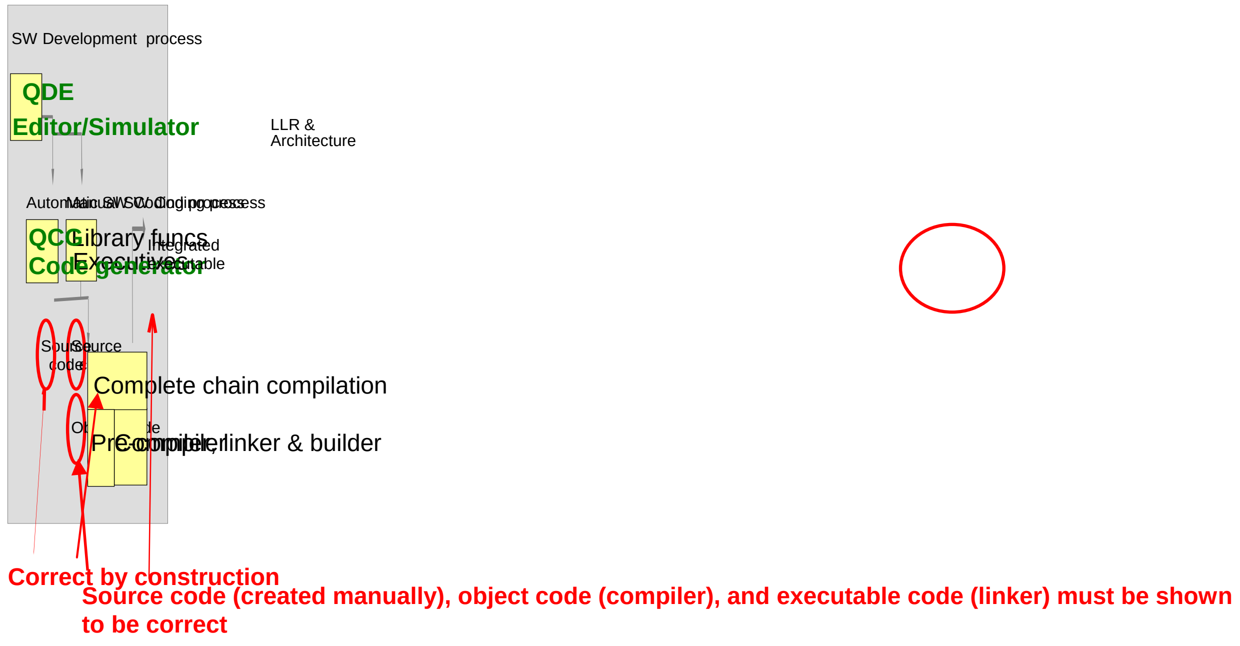
Within the design process, the software high-level requirements that were described in the graphical notation of the example QDE do not need to be further refined since source code will be generated by the QCG directly from this notation. [In this case, these high-level requirements are also considered to be low-level requirements and the guidelines for low-level requirements also apply.] Only those high-level requirements that were not described in the example QDE need to be further developed during the design process. The latter may be manually translated to software low-level requirements expressed in the graphical notation of the example QDE. Or, some software requirements expressed in the form of textual requirements, pseudo-code, or another kind of description may remain, e.g., requirements for a low-level executive functions that interface with the target hardware. It is through refinement of software requirements and their expression either in the QDE's graphical notation or other forms that the software architecture is developed.

2.1.3 Software Coding and Integration Processes

Once the software requirements are defined, source code can be automatically generated for both the software high-level and low-level requirements that are expressed in the graphical notation of the example QDE. As a qualified development tool, the example QDE can be considered to generate a one-to-one relationship between software requirements and code. As a result of this one-to-one relationship, structural coverage analysis may be met at the software requirements level instead of at the code level, as is the usual case, by ensuring that no unintended functionality exists in the implemented model, e.g., detecting dead symbols in the model (see section 3.2.5.3).

Another difference exists when using a QDE since the use of a symbol library must also be considered. Basically, a QDE uses a library of symbols (which may be qualified or not) and a procedure to call these symbols.

The software development process when using a QDE is as shown in Figure 2.


SW Development process
QDE
Editor/Simulator
LLR &
Architecture
QCG
Code generator
Library funcs
Executives
Integrated
Source
code
Complete chain compilation
Pre-compiler, linker & builder
Correct by construction
Source code (created manually), object code (compiler), and executable code (linker) must be shown to be correct

Figure

When using a QDE, the source code (both automatically and manually generated) and object code are used with the linking and loading data to generate executable object code that is then loaded into the target computer. Since not all system requirements allocated to software may be described in the notation of the example QDE, the executable code may implement both software requirements generated by the QCG and requirements manually coded from natural language or some other notation. This includes symbol library requirements. Consequently, credit claimed for executable object code will vary from project to project. That is, any claim for “Full” or “Partial” credit should be evaluated on a case-by-case basis.

Currently, three different ways exist for the QDE to call each needed symbol during the software coding and integration processes:

i) The QDE source code output is a call to each needed elementary symbol source code. The complete compilation chain includes all “needed” symbol code during pre-compilation (expansion) and then compiles, links and builds the executable object code. The “needed” symbol code is that code “called” by another source code component. Depending on some QDE generation options, source code may be an expanded one (as many copies of the code as instances of the ”calls”) or an optimized one (a single copy of the symbol code “shared” by the multiple “callers”). If the source code from the source library contains decisions, there may be some coverage issues (i.e., generated unreachable, deactivated code or dead source code). This way helps ensure that only “used” code is included in the build, and facilitates coverage analyses, requirements to code traceability and source code to object code traceability.

ii) During the source code generation, the QDE includes all symbol code (irrespective of whether the code will be needed or not) in the source library. The complete compilation chain then compiles, links and builds the executable object code. This way may introduce much “unused” code in the build and make coverage analyses and traceability analysis more difficult to demonstrate.

iii) Each individual symbol is compiled, linked and built into executable code. The QDE source code, containing the calls to each symbol is also compiled, linked and built. Each elementary symbol’s object code is loaded into random access memory (RAM) and the executable object code can then call any executable symbol code. If only the “called” symbols are included in RAM, “unused” code is kept to a minimum, and coverage analyses and traceability analysis are simplified. However, if the symbols in RAM are “shared” among partition components or functions of different software levels, then partitioning protection may be difficult to verify. If all library symbols are stored in RAM (e.g., part of the airborne software), then there is the potential for much unused code, and concerns relative to partitioning, coverage analyses and traceability analysis should be resolved.

2.2 Characteristics of the Example QDE

For purposes of discussion in this paper, an example QDE, including its QCG, is characterized in the following way[4]:

A) The QDE accepts system requirements allocated to software that are described in a graphical notation with the following properties:
   a. Notation is rigorously defined.
   b. Notation is deterministic.
   c. Notation is typically based on hierarchical block diagrams and/or state machines[5].

B) The QDE has a graphical editor that allows the user to input the requirements, i.e., develop the model.

C) The QDE comes with a checker that allows verification of user input to the model for conformance to:
   a. Syntactic rules of the notation definition
   b. Semantic rules of the notation definition (e.g., when an input is connected to a variable, it must have the same type)
   c. User-defined rules. While the syntactic and semantic rules are project independent and are imposed by the tool chain, user-specific rules can be added, e.g., context-specific naming rules, architecture constraints, etc.

D) The QDE comes with a simulator that allows the user to execute test cases generated manually by the user from language-based requirements.

E) The QDE comes with a QCG that automatically generates source code (e.g., C or Ada code) from the graphical description. The generated code exhibits the following properties:
   a. Source code is simple, verifiable, and traceable to the requirements.
   b. Source code complexity is limited, e.g., consists of linear constructs only (conditions are not allowed[6]), forbids conditional compilation and calls to another elementary symbol inside a symbol, etc.
   c. Source code exhibits behaviors with safety characteristics. In particular,
      i. Code exhibits deterministic behavior.
      ii. Code performs safe memory management.

[4] The characteristics identified for this example QDE may not necessarily reflect characteristics applicable to all QDEs. Each QDE must be evaluated on a project-specific basis for applicable characteristics and resulting benefits and limitations.

[5] This seems to cover the most widely used notations for the high-level description of embedded systems requirements. SAO, SCADE, Simulink, Stateflow, SDL etc. fall into this category.

[6] An applicant must discuss use of conditions with the tool manufacturer to understand how the model would work and potential impact on credit in satisfying DO-178B objectives.

iii. Code can be considered a safe subset of the language, e.g. Ravenscar Ada, Misra C.

d. Moreover, the execution time of the generated code is bounded and predictable.

F) The QDE, including QCG and simulator, is qualifiable on a per-project basis following the process described in Section 12.2 of DO-178B/ED-12B for software development tools and further clarified in [3]. As stated in DO-178B/ED-12B Section 12.2.1 and clarified in [3], the software development process for the QDE should satisfy the same objectives as for the development processes of the airborne software itself. For information on potential reduction of the software level, see CAST-13 [8]. For the remainder of this paper, the example QCG is assumed to be qualifiable as a development tool at Level A.

Note that, in order to avoid reviewing the source code generated by QCG, the model must be reviewed to ensure that the model accurately represents the language-based system requirements that were manually translated and described in the graphical notation of QDE.

2.3 Development Objectives and the QDE

When using a QDE to claim credit for the development processes, the objectives of Table A-2 of DO-178B/ED-12B should be evaluated on a project-specific basis. Potential compliance claims and certification concerns for the example QDE are documented below.

2.3.1 Table A-2 Compliance Claims:

Table A-2 below summarizes potential claims of using a formalized, graphical notation associated with the QDE for the software development processes. Note that the table addresses only the requirements described in QDE.

#	Table A-2 Objective	QDE Credit	Claims when using QDE
1	High-level requirements are developed	Partial[7]	High-level requirements are described in the graphical notation of QDE which, when using the QCG, generates a one-to-one relationship between software requirements and code.
2	Derived high-level requirements are defined	Partial	Although some derived requirements may be described in the graphical notation of QDE, many derived requirements may be defined outside of the model, for which case, no credit may be given.
3	Software architecture is	Full[8]	Architecture is defined by the graphical notation of QDE.

[7] By "Partial", we mean that using a QDE may facilitate a given development activity. However, this development activity still has to be completed in context of a specific project to gain full credit for the objective. Any claim for "Partial" credit should be evaluated on a case-by-case basis.

	developed		
4	Low-level requirements are developed	Partial	Low-level requirements are described in the graphical notation of QDE which, using the QCG, generates a one-to-one relationship between software requirement and code.
5	Derived low-level requirements are defined	Partial	Although some derived requirements may be described in the graphical notation of QDE, many derived requirements may be defined outside of the model for which case, no credit may be given..
6	Source Code is developed	Full	Benefit of QCG which automatically generates source code for requirements expressed in the graphical notation of QDE. However, since a QDE is usually used with a symbol library and/or other source code developed outside the QDE, <u>any claim for "Full" or "Partial" credit should be evaluated on a case-by-case basis.</u>
7	Executable Object Code is produced and integrated in the target computer	Partial	Benefit of a QCG when compiler and linker are part of QCG. Since not all system requirements may be described in the notation of the example QDE, the executable code may implement both system requirements generated by the QCG and requirements coded manually from natural language or some other notation. This includes symbol library requirements. <u>Consequently, credit claimed for executable object code will vary from project to project. That is, any claim for "Full" or "Partial" credit should be evaluated on a case-by-case basis.</u>

Table A-2: Software Development Process

2.3.2 Table A-2 Compliance Concerns:

The following certification concerns, at a minimum, should be addressed when using a QDE to satisfy Table A-2 objectives:

- Target environment should be fully understood so that the model developed using QDE reflects the software to be implemented (see Figure 2) and discrepancies identified.
- Tool limitations, e.g., constraints allowed, should be understood and their impact on satisfying the objectives determined.
- Derived requirements should still be provided to the system safety assessment process.
- When an applicant uses a QDE to partially satisfy an objective(s), the applicant should identify the means beyond QDE used to fully satisfy the objective(s).
- Code generators may generate complex code which makes the verification process and maintenance activities difficult.

[8] By "Full", we mean that using a QDE will allow elimination of the given development activity only for the software whose requirements are described in the graphical notation of QDE. Since a QDE is usually used with a symbol library and/or other source code developed outside the QDE, <u>any claim for "Full" credit should be evaluated on a case-by-case basis.</u>

- The architecture defined by the QDE notation should be combined with any manually-defined architecture to develop the full software architecture.
- Source code being developed or generated should comply with the project's coding standards.
- Credit claimed for executable object code will vary, depending on whether the compiler, its settings, and linker are considered part of the QDE or not. In addition, a QDE is usually used with a symbol library and/or other source code developed outside the QDE. As such, credit claimed for executable object code will vary from project to project. That is, <u>any claim for "Full" or "Partial" credit should be evaluated on a case-by-case basis.</u>

Other concerns may exist based on the specific QDE/QCG used and resulting model limitations. These should be addressed on a project-by-project basis.

3.0 Software Verification Process

3.1 Software Verification Process in DO-178B/ED-12B

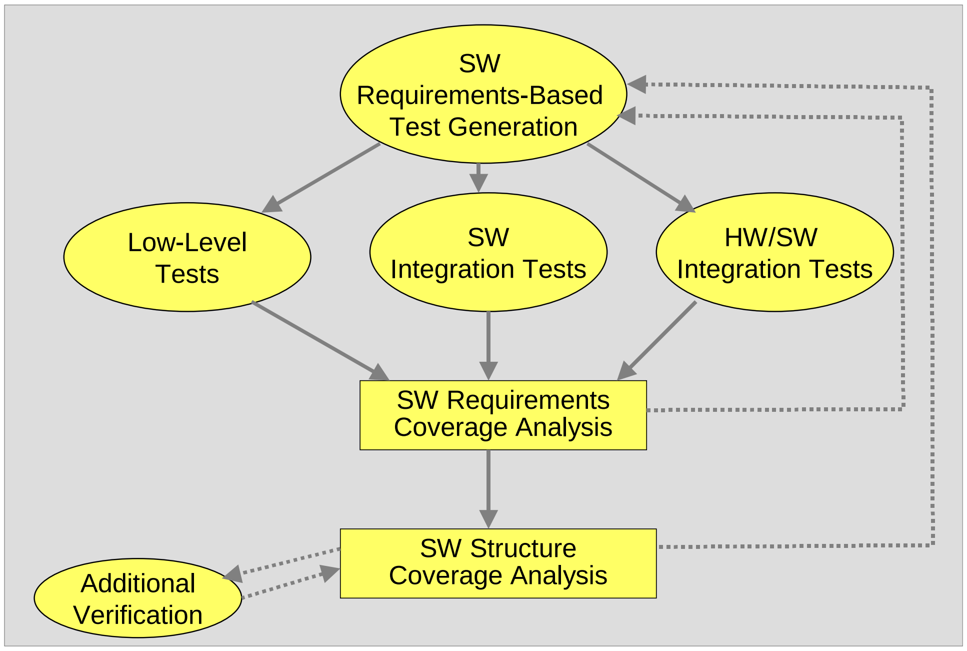

SW Requirements-Based Test Generation
Low-Level Tests
SW Integration Tests
HW/SW Integration Tests
SW Requirements Coverage Analysis
SW Structure Coverage Analysis
Additional Verification

Figure 3: Software Testing Process (DO-178B/ED-12B; Fig. 6-1)

The software verification process provides a technical assessment of the results of both the software development process as it was described above and of the software verification process itself. The verification process objectives are satisfied through a combination of reviews, analyses and tests.

Figure 3 exhibits three types of testing activities:

1. <u>Low-level testing</u> to verify that software low-level requirements (and high-level requirements if source code is generated directly from high-level requirements) are correctly implemented.
2. <u>Software integration testing</u> to verify the interrelationships between software requirements and components, and to verify the implementation of the software requirements and software components within the software architecture.
3. <u>Hardware/Software integration testing</u> to verify correct operation of the software in the target computer environment.

DO-178B/ED-12B emphasizes the development of requirements-based tests, including normal ranges tests cases and robustness (abnormal range) test cases. DO-178B/ED-12B proposes structural coverage analysis as a way to determine what software structures were not exercised and to evaluate the completeness of the requirements-based testing.

3.2 Verification Activities when an Example QDE is Used

In this section of the paper, claims are proposed for establishing elimination, reduction or automation of verification activities when the example QDE is used.

Verification is the area where QDEs can provide the most potential for satisfying DO-178B/ED-12B objectives. In the following subsections, each verification objective is presented in table format with potential credit that may be claimed when using the example QDE. Comments and assumptions about claims for each objective are described in the table although other concerns beyond those identified may exist based on the specific QDE/QCG used and resulting model limitations. These should be addressed on a project-by-project basis. Each table is followed by a summary of certification concerns for the specific table.

3.2.1 Verification of High-Level requirements (Table A-3)

<u>*3.2.1.1 Table A-3 Compliance Claims:*</u>

Table A-3 summarizes potential claims when using the formalized graphical notation of a QDE for the verification of the high-level requirements. <u>Note that the table addresses only the requirements described in QDE.</u>

#	Table A-3	QDE	Claims when using QDE

	Objective	Credit	
1	Software high-level requirements comply with system requirements	Partial[9]	May be facilitated by QDE graphical notation (rigorously defined and rule-based) to express software high-level requirements.
2	High-level requirements are accurate & consistent	Partial	Benefit of the QDE graphical notation, syntax and semantics verification. QDE tools may also help in checking the accuracy and consistency of the requirements and in checking that these requirements comply to other standards.
3	High-level requirements are compatible with target computer	Partial	Benefit of QDE model but depends on how valid the model simulator is relative to the target computer. Performance analysis (WCET[10]) may be easier to perform than on manually developed source code since the code generated by QCG may have some desirable properties, e.g., predictable execution time and bounded memory usage.
4	High-level requirements are verifiable	Full[11]	Benefit of the QDE precise notation. Since the QDE notation is rigorous and not subject to interpretation, every requirement of the model can be unambiguously verified.
5	High-level requirements conform to standards	Full	Benefit of QDE graphical notation rules (syntax and semantics). QDE tools may also help in checking the accuracy and consistency of the requirements and in checking that these requirements comply to other standards.
6	High-level requirements are traceable to system requirements	Partial	Facilitated by QDE notation and proximity to requirements notations although this may be difficult since 'requirements tags' cannot sometimes be expressed on graphical models.
7	Algorithms are accurate	Partial	Benefit of QDE graphical notation. Although verification of the accuracy of the algorithms may be facilitated by the graphical notation, assessment of the algorithms (accuracy and behavior) is typically beyond the QDE's scope.

Table A-3: Verification of Outputs of Software Requirements Process

3.2.1.2 Table A-3 Compliance Concerns:

The following certification concerns, at a minimum, should be addressed when using a QDE to satisfy Table A-3 objectives:

- Applicant should understand under what assumptions the tool operates, e.g., constraints allowed, rules and standards enforced, etc. so that the model reflects the software to be implemented and any discrepancies are identified.

[9] By "Partial", we mean that using a QDE will facilitate a given verification activity. However, this verification activity still has to be completed in context of a specific project to gain full credit for the objective. Any claim for "Partial" credit should be evaluated on a case-by-case basis.
[10] WCET = Worst Case Execution Time.
[11] By "Full", we mean that using a QDE will allow elimination of the given verification activity only for the software whose requirements are described in the graphical notation of QDE. However, any claim for "Full" credit should be evaluated on a case-by-case basis.

- When an applicant uses a QDE to partially satisfy an objective(s), the applicant should identify the means beyond QDE used to fully satisfy the objective(s).
- If full credit for an objective is claimed, this should be evaluated on a case-by-case basis in context of the project.
- HLRs, including derived HLRs, not described by QDE should be shown to fully comply with applicable objective(s).
- The QDE should provide traceability between the requirements levels.
- The QDE standards should be established and addressed as part of the project.
- Accuracy of algorithms can rarely be verified by the QDE. The QDE may be able to implement the algorithm requirements but an assessment of the algorithm accuracy is typically beyond the QDE's scope. That is, credit for objective 7 of table A-3 may be very limited.

Other concerns may exist based on the specific QDE used and resulting model limitations. These should be addressed on a project-by-project basis.

3.2.2 Verification of low-level requirements (Table A-4)

3.2.2.1 Table A-4 Compliance Claims:

Table A-4 summarizes potential claims of using the example QDE for the verification of the outputs of the software design process, mainly the low-level requirements and the software architecture of the application. Note that the table addresses only the requirements described in QDE.

#	Table A-4 Objective	QDE Credit	Claims when using QDE
1	Low-level requirements comply with high-level requirements	Partial[12]	May be facilitated by QDE graphical notation (rigorously defined and rule-based) to express software low-level requirements.
2	Low-level requirements are accurate and consistent	Partial	Benefit of the QDE graphical notation, syntax and semantics verification. QDE tools may also help in checking the accuracy and consistency of the requirements and in checking that these requirements comply to other standards.
3	Low-level requirements are compatible with target computer	Partial	Benefit of QDE model but depends on how valid the model simulator is relative to the target computer. Performance analysis (WCET[13]) may be easier to perform than on manually developed source code since the code generated by QCG has some desirable properties, e.g., predictable execution time and bounded memory usage.

[12] By "Partial", we mean that using a QDE will facilitate a given verification activity. However, this verification activity still has to be completed in context of a specific project to gain full credit for the objective. Any claim for "Partial" credit should be evaluated on a case-by-case basis.

[13] WCET = Worst Case Execution Time.

4	Low-level requirements are verifiable	Full[14]	Benefit of the QDE precise notation. Since the QDE notation is rigorous and not subject to interpretation, every requirement of the model can be unambiguously verified.
5	Low-level requirements conform to standards	Full	Benefit of QDE graphical notation rules (syntax and semantics). QDE tools may also help in checking the accuracy and consistency of the requirements and in checking that these requirements comply to other standards.
6	Low-level requirements are traceable to high-level requirements	Partial	Facilitated by QDE notation and similarity to requirements notations. Two cases have to be considered when a QDE is used: 1) Source code is generated directly from high-level requirements. In that case, there are no corresponding low-level requirements.[15] 2) Some high-level requirements were not described in the graphical notation of QDE but were refined into one or more low-level requirements that were described in the graphical notation of QDE. In that case, traceability has to established between the low-level requirement(s) and the associated high-level requirement(s).
7	Algorithms are accurate	Partial	Benefit of QDE graphical notation. Although verification of the accuracy of the algorithms may be facilitated by the graphical notation, assessment of the algorithm (accuracy and behavior) is typically beyond the QDE's scope.
8	Software architecture is compatible with high-level requirements	Partial	Benefit of QDE for the software architecture described in a (hierarchical) graphical notation.
9	Software architecture is consistent	Partial	Benefit of QDE for the software architecture described in a (hierarchical) graphical notation.
10	Software architecture is compatible with target computer	Partial	Benefit of QDE model. Performance analysis may be easier to perform than on manually developed source code since the code generated by QCG has some desirable properties, e.g., predictable execution time and bounded memory usage.
11	Software architecture is verifiable	Full	Benefit of the QDE precise notation. Since the QDE notation is rigorous and not subject to interpretation, every element of the model can be unambiguously verified for that part of the architecture expressed in the model.
12	Software architecture conforms to standards	Partial	Benefit of QDE graphical notation rules (syntax and semantics) for the software architecture described in a (hierarchical) graphical notation.
13	Software partitioning integrity is confirmed	No Credit[16]	None

Table A-4: Verification of Outputs of Software Design Process

[14] By "Full", we mean that using a QDE will allow elimination of the given verification activity only for the software whose requirements are described in the graphical notation of QDE. However, a<u>ny claim for "Full" credit should be evaluated on a case-by-case basis.</u>
[15] HLRs that are already in the graphical notation of QDE will also be considered as low-level requirements so that the corresponding objectives of Table A-4 have been satisfied for them as well.
[16] By "No Credit", we mean that using a QDE will not facilitate, in any way, a given verification activity.

3.2.2.2 Table A-4 Compliance Concerns:

The following certification concerns, at a minimum, should be addressed when using a QDE to satisfy Table A-4 objectives:

- Applicant should understand under what assumptions the tool operates, e.g. conditions allowed, rules and standards enforced, etc. so that the model reflects the software to be implemented and any discrepancies are identified.
- When an applicant uses a QDE to partially satisfy an objective(s), the applicant should identify the means beyond QDE used to fully satisfy the objective(s).
- LLRs not described by QDE should be shown to fully comply with applicable objective(s).
- For LLRs, including any derived LLRs, not described by QDE, the software architecture should be shown to be compatible with the corresponding HLRs.
- The QDE should provide traceability between HL requirements and LLR.
- Partitioning integrity should still be confirmed.
- Accuracy of algorithms can rarely be verified by the QDE. The QDE may be able to implement the algorithm requirements but an assessment of the algorithm accuracy is typically beyond the QDE's scope. That is, credit for objective 7 of table A-4 may be very limited.

Other concerns may exist based on the specific QDE/QCG used and resulting model limitations. These should be addressed on a project-by-project basis.

3.2.3 Verification of the Source Code (Table A-5)

3.2.3.1 Table A-5 Compliance Claims:

The source code implements the high-level and low-level requirements that are expressed in the graphical notation of QDE. Since we assume in this paper that the example QCG is qualifiable to Level A (i.e., the QCG satisfies all Level A objectives), the generated source code is "correct by construction". That is, the source code generated by the QCG implements the requirements expressed in the graphical notation of QDE correctly and implements those requirements only. Therefore, no further activities are needed to verify that this source code is correct, accurate and complete

However, since a QDE is usually used with a symbol library and/or other source code developed outside the QDE, actual credit claimed for all code generated will vary from project to project depending on the method(s) used to include the non-QCG-generated code. As such, credit claimed for source code will vary. That is, any claim for "Full" or "Partial" credit should be evaluated on a case-by-case basis. Details of the three ways that currently exist for a QDE to call each needed symbol are listed in section 2.1.

Table A-5 summarizes potential claims of using the example QCG for the verification of the outputs of the software coding and integration processes. Note that the table addresses only the requirements described in QDE and code generated by the example QCG.

#	Table A-5 Objective	QDE Credit	Claims when using QDE
1	Source Code complies with low-level requirements	Full[17]	Requirements are described in the graphical notation of QDE which, by qualification of QCG, generates a one-to-one relationship between software requirements and code.
2	Source Code complies with software architecture	Full	Benefit of graphical notation of QDE which is rigorous, deterministic, and typically based on hierarchical diagrams and/or state machines.
3	Source Code is verifiable	Full	Benefit of the QDE precise notation. Since the QDE notation is rigorous and not subject to interpretation, every element of the model can be unambiguously verified.
4	Source Code conforms to standards	Full	Benefit of QDE graphical notation rules (syntax and semantics).
5	Source Code is traceable to low-level requirements	Full	Facilitated by QDE functional notation and similarity to requirements notations. Two cases have to be considered when a QDE is used: 1) Source code is generated directly from high-level requirements. In that case, there are no corresponding low-level requirements.[18] 2) Some high-level requirements were not described in the graphical notation of QDE but were refined into one or more low-level requirements that were described in the graphical notation of QDE. In that case, traceability has to be established between the low-level requirement(s) and the associated source code.
6	Source Code is accurate and consistent	Full	Benefit of QCG which automatically generates source code for requirements expressed in graphical notation of QDE.
7	Output of software integration process is complete and correct	No credit[19]	None

Table A-5: Verification of Outputs of Software Coding & Software Integration Processes

3.2.3.2 Table A-5 Compliance Concerns:

The following certification concerns, at a minimum, should be addressed when using a QDE to satisfy Table A-5 objectives:

[17] By "Full", we mean that using a QDE will allow elimination of the given verification activity only for the software whose requirements are described in the graphical notation of QDE. However, any claim for "Full" credit should be evaluated on a case-by-case basis.
[18] HLR's that are already in the graphical notation of QDE will also be considered as low-level requirements so that the applicable objectives of Table A-5 have been satisfied for them as well.
[19] By "No Credit", we mean that using a QDE will not facilitate, in any way, a given verification activity.

- Objectives 1 – 6 of Table A-5 should be shown to be satisfied for those LLRs not expressed in the graphical notation of a QDE.
- Results of the integration process should still be shown to be complete and correct.
- Since a QDE is usually used with a symbol library and/or other source code developed outside the QDE, credit claimed for code generated will vary from project to project depending on the method(s) used to include such code. As such, credit claimed for source code will vary. That is, any claim for "Full" or "Partial" credit should be evaluated on a case-by-case basis.
- Code generators may generate complex code which makes the verification process and maintenance activities difficult.
- The qualification of the QCG should be evaluated in the context of the project being implemented to ensure limitations and constraints are met and the proper configuration is implemented.
- Applicant/developer should understand under what assumptions the tool operates, e.g., constraints allowed, rules and standards enforced, etc.

Other concerns may exist based on the specific QDE/QCG used and resulting model limitations. These should be addressed on a project-by-project basis.

3.2.4 Testing of Outputs of the Integration Process (Table A-6)

3.2.4.1 Problems

As seen in the previous section, the source code for the high-level and low-level requirements expressed in the graphical notation of QDE is "correct by construction". That is, it fully implements the requirements and only the requirements, for those requirements expressed in the model.

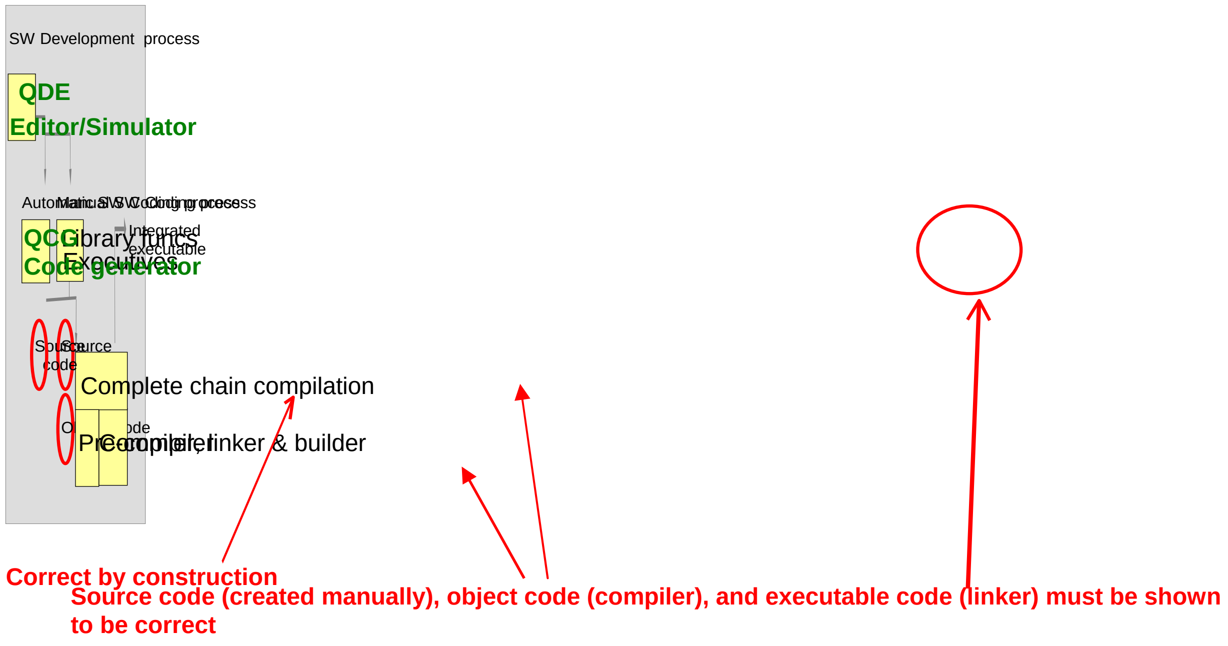
However, when testing the outputs of the integration process, other factors should be addressed:

i) The applicant should define specific rules and procedures to verify the calls to each needed library symbol during the software coding and integration processes (see section 2.1.3).

ii) The applicant should assess the correspondence between the source code generated by the example QCG and object code per DO-178B/ED-12B Section 6.4.4.2 for Level A software.

iii) Use of the library symbols or the entire library may allow 'unused', deactivated, 'unreachable' or dead code into the executable object code (EOC), and thereby produce 'holes' in the structural coverage of the software. The applicant should assess each instance (hole) introduced, and ensure it will result in no anomalous behavior or unintended function.

Production of the executable code when a QCG is used is illustrated by Figure 4 (which is the same as Figure 2).


SW Development process
QDE
Editor/Simulator
Automatic SW Coding process
Manual SW Coding process
QCG
Code generator
Library funcs
Executives
Integrated executable
Source code
Source
Object code
Complete chain compilation
Pre-compiler
Compiler, linker & builder
Correct by construction
Source code (created manually), object code (compiler), and executable code (linker) must be shown to be correct

Figure

3.2.4.2 The Combined Testing Process

An approach similar to that presented in CAST-12 [4] is taken:

- For the requirements that are manually coded in the source code language (e.g., library functions, executives, etc.):
    - o The applicant performs the typical verification activities (including normal and robustness testing of high-level and low level requirements and structural code coverage analysis).
    - o The compiler (including any pre-compiler) used to generate the object code is the same version using the same options (no optimization) in the same execution environment as is used to compile source code obtained from the example QCG.
    - o Analysis of the object code is performed according to CAST -12 [4] to demonstrate that any object code not directly traceable to source code is correct.
- For the source code automatically generated by the example QCG:
    - o By specification, the QCG uses only a small subset of the general purpose source code language, with a low level of complexity (mostly expressions with comparisons, +; -, etc) and generates a safe subset of the language used, e.g., Misra C.
    - o The applicant performs normal testing activities on generated source code that comprises all source code programming constructs specified in the coding

standards in order to demonstrate that the object code generated from this source code is correct and does not introduce erroneous code that is not traceable at the source code level (as in CAST -12 [4]).

  - Also, it is now possible to execute manually developed, requirements-based test cases in simulation using a QDE simulator and to collect the results of this simulation. The QDE simulator allows the user to follow the values of data as execution of the test case progresses. Comparison of the actual output values with the expected output values can be made at any step in the simulation. However, if credit is expected from simulation of the formalized requirements, requirements-based test coverage analysis of the high level specification description should be performed.
  - The applicant should perform robustness testing of both high and low level requirements unless design standards enforce inclusion of robustness requirements in system requirements.
  - The applicant should limit source code complexity (e.g., forbid conditional compilation, forbid conditions in an elementary symbol, disallow calling another elementary symbol inside a symbol, etc.).
- For the whole application:
  - The applicant performs extensive system requirements-based software and hardware/software integration testing.

The combination of all the above activities should give confidence that the compiler (including any pre-compiler) does not introduce any undetected errors in the code generated by the example QCG for the source code programming constructs used.

3.2.4.3 Table A-6 Compliance Claims:

The user should manually develop system requirements-based test cases for the system requirements allocated to software and generated in a QDE. These test cases can then be executed in the QDE using the simulator provided.

Table A-6 below summarizes potential claims of using the example QDE for the verification of the outputs of the integration process. <u>Note that the table reflects only the requirements described in the example QDE.</u> However, since a QDE is usually used with a symbol library and/or other source code developed outside the QDE and, since the compiler and linker may not be part of the QCG, the <u>likelihood of receiving any credit for the outputs of the integration process is minimal</u>.

#	Table A-6 Objective	QDE Credit	Claims when using QDE

1	Executable Object Code complies with high-level requirements	Partial[20]	Benefits from QDE and the combined testing process described above. However, since a QDE is usually used with a symbol library and/or other source code developed outside the QDE, and since the compiler may not be part of the QCG, the likelihood of receiving any credit is minimal. Any claim for "Full" or "Partial" credit should be evaluated on a case-by-case basis.
2	Executable Object Code is robust with high-level requirements	No credit[21]	None, but may be enforced by appropriate design and coding rules
3	Executable Object Code complies with low-level requirements	Partial	Benefits of QDE and the combined testing process described above. However, since a QDE is usually used with a symbol library and/or other source code developed outside the QDE, and since the compiler may not be part of the QCG, the likelihood of receiving any credit is minimal. Any claim for "Full" or "Partial" credit should be evaluated on a case-by-case basis.
4	Executable Object Code is robust with low-level requirements	No Credit	None, but may be enforced by appropriate design and coding rules.
5	Executable Object Code is compatible with target computer	Partial	Benefit of QDE model when the compiler and linker are part of QCG. Code generated by QCG has some desirable properties, e.g., execution time is deterministic. Also, since not all system requirements may be described in the notation of the example QDE, the executable code may implement both system requirements generated by the QCG and requirements coded outside the QDE. This includes symbol library requirements. Consequently, credit claimed for executable object code will vary from project to project. That is, any claim for "Full" or "Partial" credit should be evaluated on a case-by-case basis.

Table A-6: Verification of Outputs of Integration Process

3.2.4.4 Table A-6 Compliance Concerns:

The following certification concerns, at a minimum, should be addressed when using a QDE to satisfy Table A-6 objectives:

- Normal and robustness test cases should be developed by the user for all high-level and low-level requirements.
- Requirements-based integration test cases should still be developed.
- Approach to assess correspondence between source and object code should be repeated when a different compiler or different compiler options are used.
- When an applicant uses a QDE to partially satisfy an objective(s), the applicant should identify the means beyond QDE used to fully satisfy the objective(s).

[20] By "Partial", we mean that using a QDE will facilitate a given verification activity. However, this verification activity still has to be completed in context of a specific project to gain full credit for the objective. Any claim for "Partial" credit should be evaluated on a case-by-case basis.

[21] By "No Credit", we mean that using a QDE will not facilitate a given verification activity.

- Credit claimed for executable object code will vary, depending on whether the compiler, its settings, and the linker are considered part of the QDE or not. In addition, a QDE is usually used with a symbol library and/or other source code developed outside the QDE. As such, credit claimed for executable object code will vary from project to project. That is, any claim for "Full" or "Partial" credit should be evaluated on a case-by-case basis.
- If credit is expected from simulation of the formalized requirements, requirements-based test coverage analysis of the high level specification description should be performed.

Other concerns may exist based on the specific QDE/QCG used and resulting model limitations. These should be addressed on a project-by-project basis.

3.2.5 Verification of the Verification process (Table A-7)

Outputs of the verification processes are verified with concentration in three areas:
i) test procedures and results
ii) test coverage of system requirements allocated to software
iii) test coverage of the software structure (structural coverage analysis).

3.2.5.1 Test Procedures and Results

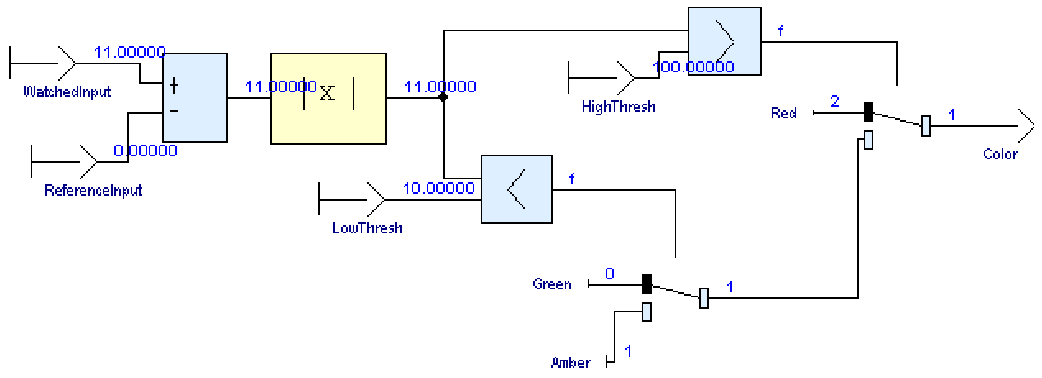
The objective of this review/analysis is to ensure that testing of the code was developed, and performed accurately and completely. Use of a QDE does not provide additional credit for satisfying objectives in this area. As noted in Section 3.2.4.3, the user should manually develop system requirements-based test cases for the system requirements allocated to software and generated in the example QDE. Some of these test cases may be executed in the example QDE using the simulator provided.

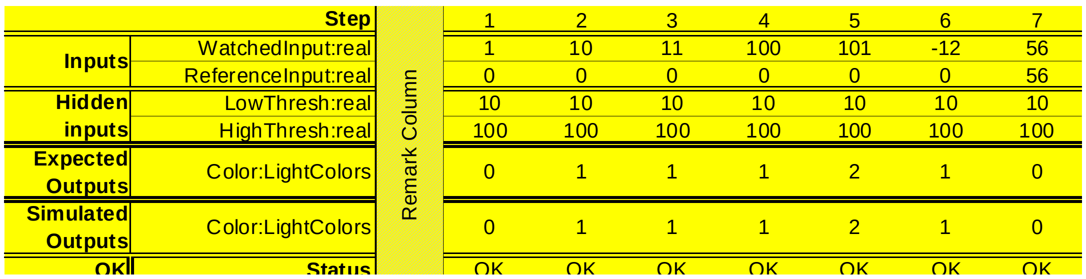
3.2.5.2 Test coverage analysis of the software requirements

a) Typical activities
The objective of this review/analysis is to determine how well the requirements-based testing verified the implementation of the system requirements allocated to software. This analysis may reveal the need for additional requirements-based tests. The requirements-based test coverage analysis should show that:
- Test cases exist for each system requirement allocated to software.
- Test cases satisfy the criteria of normal and robustness testing as defined in Section 6.4.2 of DO-178B/ED-12B.

b) Additional verification

Using the example QDE simulator, a user may execute system requirements-based test cases as shown in Figure 5 and coverage of requirements may be approached in a more formal way.





	Step	Remark Column	1	2	3	4	5	6	7
Inputs	WatchedInput:real		1	10	11	100	101	-12	56
	ReferenceInput:real		0	0	0	0	0	0	56
Hidden inputs	LowThresh:real		10	10	10	10	10	10	10
	HighThresh:real		100	100	100	100	100	100	100
Expected Outputs	Color:LightColors		0	1	1	1	2	1	0
Simulated Outputs	Color:LightColors		0	1	1	1	2	1	0
OK	Status		OK	OK	OK	OK	OK	OK	OK

Figure 5: Software requirements verification with a QDE

For example, in the block diagram notation shown in Figure 5, it is possible to extend the MC/DC criterion that normally applies to source code to these block diagrams, as this is explained in [6]. The set of tests may be characterized in the following manner:

- Cases 1, 3 and 5 cover the block diagram from strictly a MC/DC criterion.
- Cases 2 and 4 are added for checking the accuracy of the comparator.
- Cases 6 and 7 are added to show correctness of absolute value function and are beyond the coverage analysis of the current block diagram.

3.2.5.3 Structural coverage analysis (MC/DC, decision, statement)

The objective of structural coverage analysis is to determine which code structure was not exercised by requirements-based tests. Section 6.4.4.3 of DO-178B/ED-12B explicitly states that structural coverage analysis may reveal code structures not exercised during testing that may be the result of:

- Shortcomings in requirements-based test cases or procedures
- Inadequacies in software requirements
- Dead code

- o Deactivated code

<u>Shortcomings in requirements-based test cases or procedures:</u>

Since only partial credit is given to use of the example QDE for requirements-based testing**,** shortcomings in the tests cases or procedures may exist and must be identified. In particular, robustness test cases must be manually developed for all requirements as well as normal test cases for those requirements not expressed in the example QDE.

<u>Inadequacies in software requirements:</u>

Problems with the software requirements may be exhibited using the testing activity described in the previous section for code both generated by the example QCG and for manually developed code.

<u>Dead code:</u>

Since the example QCG is qualified as development tool, dead code cannot be introduced by the QCG unless a problem exists at the software requirements level. If structural coverage analysis identifies dead code, then the associated requirements problem needs to be addressed. Dead code may exist, however, for requirements that are manually coded, i.e., not generated in the QDE. This is addressed by structural coverage of the manually developed code.

<u>Deactivated code:</u>

Deactivated code may be introduced by the example QCG if a requirement(s) is applicable only to certain configurations of the software or an entire symbol library is linked into the EOC or loaded into target computer, including “unused” library functions.

3.2.5.4 Structural coverage analysis (data and control coupling)

The objective of this analysis is to confirm the data and control coupling between the components of the code. Use of the example QDE does not guarantee additional credit for satisfying the objective in this area. In addition, the analysis of the coupling between the QCG generated code and the manually generated code may be more complicated since each may use different notations that are not compatible. Any claim for credit should be evaluated on a case-by-case basis in the context of the project.

3.2.5.5 Table A-7 Compliance Claims

Table A-7 below summarizes potential claims of using the example QDE for the verification of the verification process results. Note that the table addresses only the requirements described in QDE.

#	Table A-7 Objective	QDE Credit	Claims when using QDE
1	Test procedures are correct	No Credit[22]	None
2	Test results are correct and discrepancies explained	No Credit	None
3	Test coverage of high-level requirements is achieved	Partial[23]	QDE notation may benefit testing and test coverage analysis of software high-level requirements
4	Test coverage of low-level requirements is achieved	Partial	QDE notation may benefit testing and test coverage analysis of software low-level requirements
5	Test coverage of software structure (modified condition/decision coverage) is achieved	Partial	QDE notation may benefit analyses at the software requirements level, e.g., when using the simulator. However, this should be evaluated on a case-by-case basis in the context of the project and the user should demonstrate the required level of traceability.
6	Test coverage of software structure (decision coverage) is achieved	Partial	QDE notation may benefit analyses at the software requirements level, e.g., when using the simulator. However, this should be evaluated on a case-by-case basis in the context of the project and the user must demonstrate the required level of traceability.
7	Test coverage of software structure (statement coverage) is achieved	Partial	QDE notation may benefit analyses at the software requirements level, e.g., when using the simulator. However, this should be evaluated on a case-by-case basis in the context of the project and the user must demonstrate the required level of traceability.
8	Test coverage of software structure (data coupling and control coupling) is achieved	Partial	QDE notation may permit analyses at the software requirements level but a case-by-case analysis is required.

Table A-7: Verification of Verification Process Results

[22] By "No Credit", we mean that using a QDE will not facilitate, in any way, a given verification activity.
[23] By "Partial", we mean that using a QDE will facilitate a given verification activity. However, this verification activity still has to be completed in context of a specific project to gain full credit for the objective. Any claim for "Partial" credit should be evaluated on a case-by-case basis.

3.2.5.6 Table A-7 Compliance Concerns:

The following certification concerns, at a minimum, should be addressed when using the example QDE to satisfy Table A-7 objectives:

- Test procedures and results should still be shown to be correct or discrepancies explained.
- Requirements-based test coverage of both high-level and low-level requirements not expressed in the QDE should be shown to have been achieved.
- Requirements-based test coverage of both high-level and low-level requirements expressed in the QDE should be evaluated for shortcomings in test cases or procedures, inadequacies in software requirements, dead code, and deactivated code.
- Claims for test coverage of the software structure should be considered on a case-by-case basis in the context of the project (i.e., objectives 5 through 8 should be evaluated in the context of the project).

Other concerns may exist based on the specific QDE/QGC used and resulting model limitations. These should be addressed on a project-by-project basis.

4.0 Conclusion

The purpose of this paper is to highlight certification concerns when a generic QDE and its QCG are used to develop airborne software to comply with DO-178B/ED-12B objectives. While some concerns were identified throughout this paper, other concerns may exist on a project-specific basis depending on the specific QDE/QCG used and resulting model limitations. This paper proposes a potential way to address DO-178B/ED-12B objectives when using an example QDE which includes a QCG. It emphasizes that such a process is not just a matter of qualification of the code generator itself. The notation used for the software requirements should be formal (rigorously defined, verifiable, and deterministic), and the requirements of the code generator should include properties such that the generated code exhibits behaviors with safety characteristics, e.g., determinism, and is simple, verifiable, and traceable to the requirements. Additionally, use of a QDE should be addressed in a project-specific context and the airborne software should still be shown to satisfy all applicable DO178B/ED-12B objectives.

A secondary, but important, purpose of this paper is to show that the need to verify the output of the coding process (Table A-5) for the requirements implemented using the example QDE and its associated QCG may be reduced. However, all other source code, i.e., code not generated by the model including symbol libraries, should still be verified. Additionally, traceability of the generated source code from the example QCG to the object code should be assessed for Level A software and testing should be performed (software integration testing, hardware/software integration testing, robustness testing, and system level testing).

A summary of the concerns identified in this paper follows. Note, however, that this may not be an exhaustive list for all QDEs since QDE characteristics may vary and considerations should be made on a project-by-project basis:

- Use of a QDE should be well planned and documented in the Plan for Software Aspects of Certification (PSAC), Tool Qualification Plan, and other planning documents, as appropriate. The PSAC, Tool Qualification Plan, and other software planning documents should be submitted to the certification authority for approval early in the project.
- Applicants, developers, and certification authorities should fully understand the QDE model, the QDE tool suite, and their limitations.
- The qualification of the QCG should be evaluated in the context of the project being implemented to ensure limitations and constraints are enforced, and the proper configuration is implemented. In addition, a QDE is usually used with a symbol library and/or other source code developed outside the QDE. As such, credit claimed for executable object code will vary from project to project. That is, any claim for "Full" or "Partial" credit should be evaluated on a case-by-case basis.
- Use of a QDE does not relieve the applicant of complying with applicable DO-178B/ED-12B objectives. When an applicant uses a QDE to partially satisfy an objective(s), the applicant should identify the means beyond the QDE to fully satisfy the objective(s).
- Code generators may generate complex code which makes the verification process and maintenance activities difficult. If a QCG generates object code directly, i.e., source code is not produced, these actions may be infinitely more difficult, approaching the impossible.
- Source code generation should be a desired output of the QCG versus generation of object code only.
- Source code generated should comply with the project's coding standards.
- The target environment should be fully understood so that the model developed by the QDE reflects the software to be implemented and any discrepancies identified.
- Credit claimed for executable object code will vary, depending on whether the compiler and its settings are considered part of the QDE or not.
- HLRs, including derived HLRs, not described by QDE should be shown to fully comply with applicable objective(s). Likewise, LLRs not described by QDE should be shown to fully comply with applicable objective(s).
- The QDE should provide traceability between the requirements levels.
- QDE should provide traceability between requirements and code.
- The QDE standards should be established and addressed as part of the project.

- Results of the integration process should still be shown to be complete and correct.
- Robustness test cases should be developed and executed for both high-level and low-level requirements.
- Normal test cases should be developed and executed for both high-level and low-level requirements, including those of the symbol libraries, that are not expressed in the QDE.
- Derived requirements should be tested and provided to the system safety assessment process.
- Requirements-based integration test cases should be developed.
- Approach to assess correspondence between source and object code should be repeated when a different compiler or different compiler options are used.
- Appropriate complexity limitations should be set depending on the way the source code is generated.
- Test procedures and results should be shown to be correct or discrepancies explained.
- Requirements-based test coverage of both high-level and low-level requirements not expressed in the QDE should be shown to have been achieved.
- Requirements-based test coverage of both high-level and low-level requirements expressed in the QDE should be evaluated for shortcomings in test cases and procedures, inadequacies in software requirements, dead code, and deactivated code.
- Claims for coverage of the software structure should be addressed on a case-by-case basis in the context of the project (i.e., objectives 5 through 8 of Table A-7 should be evaluated in the context of the project).
- Although the QDE may be able to implement the algorithm requirements, verification of the algorithm accuracy is typically beyond the QDE’s scope.
- If credit is expected from simulation of the formalized requirements, requirements-based coverage analysis of the high level specification should be performed.

5.0 References

1. RTCA/DO-178B (EUROCAE/ED/12B), “Software Considerations in Airborne Systems and Equipment Certification,” December 1992.

2. SAE/ARP-4754, “Certification Considerations for Highly-Integrated or Complex Aircraft Systems,” October 1996.

3. “Guidelines for the Qualification of Software Tools using RTCA/DO-178B,” FAA Order 8110.49, Chapter 9, June 3, 2003.

4. CAST-12, “Guidelines for Approving Source Code to Object Code Traceability,” December 2002.

5. "Cost Effectiveness of Formal Methods on the Development of Avionics Systems at Aerospatiale," François Pilarski (Airbus), 17th Digital Avionics Systems Conference, November 1st 1998, Seattle.

6. "A Practical Tutorial on Modified Condition/Decision Coverage," Kelly J. Hayhurst (NASA), Dan S. Veerhusen (Rockwell Collins), John J. Chilenski (Boeing), Leanna K. Rierson (FAA).

7. EUROCAE/ED-79 (see SAE ARP4754), "Certification Considerations for Highly Integrated or Complex Aircraft Systems," April 1997.

8. CAST-13, "Automatic Code Generation Tools Development Assurance," June 2002.