Frédéric Pothon
ACG Solutions

Dissimilar tools: Use cases and impact on tool qualification level

October 2015

Contributions and Reviews

Jean-Charles Dalbin
AIRBUS

Ben Brosgol
AdaCore

Content

1. Purpose of This Document

Frequently, the lack of existing qualifiable COTS tools raises the question whether two separately developed and thus dissimilar tools that satisfy the same operational requirements may be used instead of a single tool, either to avoid the need for qualifying that tool or to serve as an alternative method for tool qualification.

This question may be answered in different ways by different certification authorities, since no guidance or guidelines are provided in the applicable standards, such as DO-178C/ED-12C. DO-330/ED-215 identifies the use of dissimilar tools as an example of “alternative method for tool qualification” but without definition of the actual impact of such approach. DO we have to understand that the use of non-qualified dissimilar tools replaces the need for qualification of a single tool? Is this applicable for all TQL?

This paper presents several use cases based on the type of tools and certification credit, and examines the possible impact on the need for qualification and the applicable Tool Qualification Level of the use of dissimilar tools.

Disclaimer: Discussion of this topic in the Forum for Aeronautical Software (FAS) (see http://www.rtca.org/content.asp?pl=33&sl=170&contentid=170) raised controversial questions and different opinions. Therefore, it was difficult to reach a consensus on an Information Paper. It was decided that this topic is beyond the scope of the FAS working group, and the proposal was withdrawn. This document reflects the author’s point of view, based on his background and experience. It should help readers to provide an acceptable rationale for a software life cycle using dissimilar tools in place of a single qualified tool.

2. Context

Section 11.5 of DO-330/ED-215 (Alternative Methods for Tool Qualification) mentions the use of dissimilar tools as an example of an alternative method.

An alternative method for tool qualification, once the applicable Tool Qualification Level (TQL) is defined, involves demonstrating that the DO-330/ED-215 objectives have been satisfied with a method or activities beyond those described in the document.

Citing the use of dissimilar tools as an alternative method is not appropriate. Most of the cases the use of dissimilar tools is not an alternative method for tool qualification but rather a modification of the software life cycle processes, so of the need for tool qualification and/or of the applicable TQL.

For all tools used in the software life cycle, the first step is to identify the need for tool qualification based on DO-178C/ED-12C 12.2. The second step is to satisfy the DO-330/ED-215 objectives for the tool that needs qualification at the applicable TQL.

When dissimilar tools are used instead of a single one, the need for tool qualification of each tool and then identification of the applicable TQL are determined based of the applicable Tool Qualification Criteria and on the Software Level. Then, the tool qualification process activities are defined to satisfy the tool qualification objectives for the applicable TQL. These objectives might not be identical for a single tool or for dissimilar tools as they depend on the new TQL applicable to this particular situation.

This document presents an analysis of typical use cases with dissimilar tools and the possible impact on the applicable TQL.

Once the Tool Qualification Criteria and applicable Tool Qualification Levels are determined, the possible TQL adaptation for dissimilar tools, depending of the applicable qualification criteria, will be discussed.

3. Tool Qualification Criteria and Levels

This chapter summarizes the tool qualification criteria in the airborne domain.

The need for qualification and the determination of the applicable TQL is addressed in DO-178C/ED-12C section 12.2 based on two considerations: the impact of a tool error in the resulting software, and the software level:

For the impact of an error in the resulting software, three criteria are defined:

- Criterion 1: A tool whose output is part of the airborne software and thus could insert an error.
- Criterion 2: A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:
    - Verification process(es) other than that automated by the tool, or
    - Development process(es) that could have an impact on the airborne software.
- Criterion 3: A tool that, within the scope of its intended use, could fail to detect an error (if criteria 2 is not applicable)

The Tool Qualification Level is defined in Table 12-1, based on the Software Level and the applicable criteria:

Software Level	Criteria		
	1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

:

4. Tool qualification versus tool output verification

4.1 If a tool's output is verified, qualification is not required…

When the output of a tool is verified as described in DO-178C/ED-12C section 6, qualification of the tool is not required. This verification may be performed by a combination of reviews, analyses, and automated verification performed by one or more other tools to satisfy all objectives applicable to the output from the unqualified tool.

DO-330/ED-215 FAQ D.7 (*How might one use a qualified tool to verify the outputs of an unqualified tool?)* provides guidelines on the use of qualified verification tool(s) to verify the output of a non-qualified tool. This is typically the approach proposed when a non-qualified autocode generator is used.

However the FAQ provides some additional explanation that needs to be taken into account. "*The ability of tools to satisfy these objectives is key to selecting the overall verification approach*". It might not be possible to use the output of another tool to satisfy all applicable DO-178C/ED-12C verification objectives for the output of the tool in question. Therefore, a determination needs to be made about which objectives can be addressed by a qualified tool.

This approach has been successfully applied to autocode generators that produce some "static" code, such as a configuration or a data base, where the full content of the tool output can be precisely defined. For more complex tool outputs, such as source code generated from design models, a combination of qualified verification tools, manual reviews and analyses are necessary to satisfy all the applicable verification objectives.

4.2 … But the certification credit is not equivalent!

It should be noted that the possible certification credit using an unqualified autocode generator (with tool outputs verified) is not equivalent to the use of a qualified autocode generator. DO-330/ED-215 FAQ D.8: "How Might One Use a Qualified Autocode Generator?" identifies different scenarios, and explains that, depending of the qualification effort, certification credit may be claimed on objectives of requirements-based tests (DO-178C/ED-12C Table A-6), and the related verification of verification data (DO-178C/ED-12C Table A-7). The qualification effort may include some activities equivalent to low-level requirement-based tests, performed through equivalence classes of input files. The activity performed in the scope of the tool operational verification and validation process includes, as a minimum, the development of test cases and procedures, execution of the test procedures to obtain the test results; and verification of all these test data to satisfy objectives 1, 2, and 4 of DO-178C/ED-12C Table A-7. To satisfy objective 4 of DO-178C/ED-12C Table A-7 it is necessary to perform coverage analysis of the requirements contained in the representative set of input files.

In the case where an unqualified autocode generator (with tool output verified) is used, these qualification activities are not conducted. The certification credit claimed is thus limited to some of the tool output (typically source code) verification objectives. Alleviation of test effort is not allowed when using an unqualified autocode generator. Additional activities should be performed in the scope of the software life cycle to satisfy these objectives.

5. What are “dissimilar tools”?

Two tools that satisfy the same operational requirements may be considered as “dissimilar” when measures have been taken to prevent the risk of producing the same error in their output for a given input.

The term “protection” is defined in DO-330/ED-215 glossary as “The use of a mechanism to ensure that a tool function cannot adversely impact another tool function”. Such protection is a prerequisite for considering the tools to be dissimilar. This is particularly the case when the two tools are exercised in parallel and/or on the same workstation. As part of the protection mechanism it should be demonstrated that one tool cannot modify the other tool outputs.

To prevent the presence of a similar error in both tools, some constraints need to be observed. This “common cause avoidance” as defined in FAQ D.7 of DO-330/ED-215 “*may take one of several forms, including but not limited to those listed below:*

- *Independent tool development: The qualified tool is developed independently from the unqualified tool, using separate development personnel within a separate tool development life cycle. Demonstrating the independence of tool development may require more tool development life cycle data than is normally necessary for qualification. This approach to independence does not necessarily require that the qualified tool be created by a different organization than the unqualified tool.*

- *Dissimilar technical approach: Depending upon the unqualified tool functionality, the qualified tool may need to use a different technical approach than the unqualified tool. This is similar to error-checking steps commonly used in solving algebraic equations, where one mathematical process is used to arrive at an answer, and a second process is used to check that the answer is correct.”*

To demonstrate dissimilarity, the two tools cannot use same code components, such as libraries. Two tools developed in the same code/script language might not be considered as dissimilar, as they will use the same interpreter/compiler and/or libraries. These are potential error sources, and the same error may be present in the two tools.

Discussion about the effectiveness of dissimilarity between tools is preliminary to further assessment of TQL allocation.

6. “Dissimilar tools” and “Multiple-Version Dissimilar Software”

6.1 Development Assurance Level assignment

The Development Assurance Level (DAL) is based on the possible contribution of an error in a function or item to a failure condition at aircraft level. Containment for the effects of development or design errors may be defined by the system architecture by providing two or more independent functions or items. Multiple-Version Dissimilar software is an example of this architecture

ARP4754A/ED-79A defines the DAL assignment, and conditionally permits a reduction of level in the case of functional failure sets with multiple functions or items.

Here is a simplified extract of table 5-2:

TOP-LEVEL FAILURE CONDITION CLASSIFICATION	DEVELOPMENT ASSURANCE LEVEL					
	FUNCTIONAL FAILURE SETS WITH					
	SINGLE MEMBER	MULTIPLE MEMBERS OPTION 1		MULTIPLE MEMBERS OPTION 2		
		First item	Other items	First item	Second item	Other items
Catastrophic	DAL A	DAL A	(*) but not lower than” DAL C	DAL B	DAL B	(*)
Hazardous	DAL B	DAL B	(*) but not lower than” DAL D	DAL C	DAL C	(*)
Major	DAL C	DAL C	(*)	DAL D	DAL D	(*)
Minor	DAL D	DAL D	(*)	(*)	(*)	(*)
No Safety Effect	DAL E	DAL E				
(*) : the level associated with the most severe individual effects of an error in their development process for all applicable top-level Failure Conditions						

In option 2, the DAL may be reduced by one level in each individual item (e.g., DAL B instead of DAL A).

May we apply the same approach for dissimilar tools? In order words, instead of qualifying a single tool at TQL-1 for example, may we develop two dissimilar tools and qualify each one at TQL-2?

6.2 Aspects of Tools That Differ From Airborne Software

When multiple-version dissimilar software is used, the system architecture is modified. All the system components are installed in the aircraft and may contribute to a failure conditions.

When dissimilar tools are used, the impact is not on the system architecture, but only on the processes to develop a single system component, that is, the software. So as for any software life cycle, all tools used need to be assessed for possible impact on errors in the resulting software. (See DO-178C/ED-12C section 12.2).

Sometimes parallels are made between software and tools. However, this is often not relevant with respect to their impact on safety. The effect of an error in a tool is not the same as for an error in the airborne software. Errors in airborne software could lead directly to an impact on safety and may contribute to a failure condition. It is not the same for tools.

If we consider a criteria 1 tool (which may inject an error in the resulting software), the following consequences of tool errors have **no impact on safety**:

- The tool crashes: Normally, no output file is generated, and the software cannot be produced.
- The tool output (typically source code for most of criteria 1 tools) is incorrect, e.g. syntactic or semantic errors. But such an error makes it impossible to generate the executable object code (the compiler, linker or other integration tool rejects the input). The software is not generated.
- The tool output includes some errors but without impact on the behavior of the resulting software: The error is with comments, headers, naming conventions, presentation rules, etc. The software is generated but it is compliant with the requirements.
- The tool generates non-optimized software, for example creating data that is not used. . The software is generated incorrectly but is compliant with the requirements.

Finally, the only case in which a tool error has a safety impact is when the tool introduces an error in its output that results in non-compliance with tool input and is not detected by other steps within the software life cycle processes.

In consequence, we cannot simply take the approach for Multiple-Version Dissimilar software and apply it to dissimilar tools. We need to analyze the possible impact of each tool upon the resulting software, based on the tool criteria, the interfaces with other tools, the certification credit sought, and the impact on the software life cycle processes

This analysis is the purpose of the next several chapters

- Chapter 7 will discuss criteria 1 tools through several examples; the first example 1 will be presented in full detail, and the other examples will be explained in terms of differences from the first.
- Chapter 8 will discuss criteria 2 tools
- Chapter 9 will discuss criteria 3 tools

These chapters identify several possible software life cycles that may be defined using dissimilar tools. This paper doesn't attempt to be exhaustive, and other life cycles are possible.

7. Use Case n° 1: A Criteria 1 Tool

A criteria 1 tool is a tool whose output is part of the resulting software and thus could insert an error (if its output is not verified).

The applicable TQL depends on the software level (TQL-1 for software level A to TQL-4 for software level D). Typically this criterion is applicable to an autocode generator producing source code from a design model.

Instead of using a single autocode generator chain, a Tool User may use two autocode generators, and thereby generate the source code twice. But it is important to note that only one version of the source code will be used for the integration process. That introduces the need to perform a “comparison” between the two versions.

A tool chain with “dissimilar criteria 1 tools” will thus include two autocode generators and a checker.

Typically two methods for the development of the autocode generators are used in practice:

- **Redundancy method**: Two autocode generators are developed based on the same TOR. . They take the same input (a design model) and produce the source code in either the same language or in different languages.
- **Feedback method**: One autocode generator takes the design model as input and generates the source code. The second autocode generator performs reverse engineering: it takes the source code generated by the first autocode generator and produces the model in a form that allows the verification of complete and correct code generation.

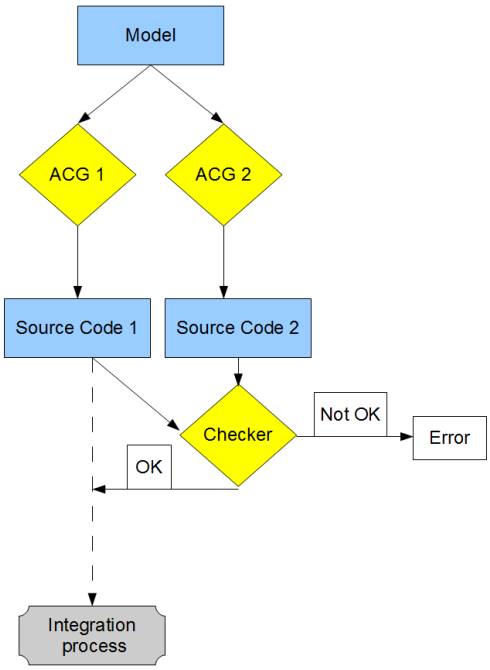
For the checker there are three typical approaches:
- **Diff Checker**: A comparison is performed on the two versions of the code/model, if the same language/formalism is used for both
- **Equivalence Checker**: A comparison is performed on the two different versions of the code/model (if different languages/formalisms are used)
- Verification is performed by **reviews and analyses** (not automated)

The following sections assess the impact of using different combinations of these methods.

7.1 Example 1: Qualification of the second autocode generator with a checker as a tool collection

7.1.1 Description

In this example two autocode generators are used, based on the redundancy method and a checker (diff or equivalence method). In other words, one of the autocode generators produces the “official” source code. The second autocode generator produces the “expected” source code that is used to verify the “official” source code.



In this example, the data to be verified is Source Code 1 (the source code that will be used for integration if correct). Source Code 2 is only necessary for verification.

7.1.2 Tool Qualification Levels

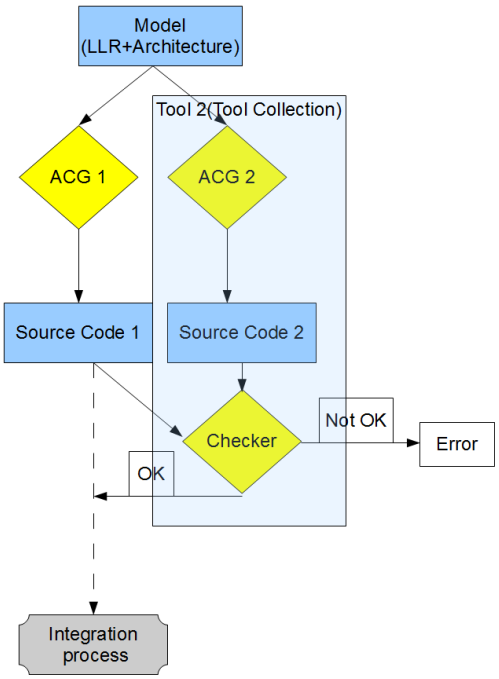
This process is similar to the one described in DO-330/ED-215 FAQ D.7: How might one use a qualified tool to verify the output of an unqualified tool?

Since Source Code 1 is verified, the tool ACG1 doesn't need to be qualified. The collection of tools (ACG2 + Checker) is used to verify the output of the non-qualified ACG1 tool. This tool collection will not inject an error, but may fail to detect an error. It needs to be qualified as criteria 2 or 3 (see below), depending of the certification credit claimed:

According to DO-178C/ED-12C 12.2:

- Criteria 3 is applicable if the certification credit is limited to the verification of the output of ACG1
- Criteria 2 is applicable if the output of the tool collection (ACG2+Checker) is "used to justify the elimination or reduction" of another process

The picture may be displayed as follows:



Source Code 2 (the output of ACG2) is hidden from the Tool User and is only used internally in the tool collection. It is an input to the “Checker” and is used as “expected source code” to verify the correctness of Source Code 1. Finally ACG2 is not a standalone tool that is of interest in itself but rather a tool whose purpose is to produce input for the checker. It is part of a tool collection implementing a process activity that is the verification of source code.

In compliance with FAQ D.7 the relevant TQLs using this approach apply to tool 1 (ACG1) and tool 2 (ACG2+Checker), where “NQ” means “Not Qualified”:

Software Level	**TQL for Criteria 1**		
	Single tool	**Dissimilar tools**	
		ACG1	**Tool 2**
A	TQL-1	NQ	TQL-5**(*)**
B	TQL-2	NQ	TQL-5**(*)**
C	TQL-3	NQ	TQL-5**(*)**
D	TQL-4	NQ	TQL-5**(*)**

(*) under the assumption that the certification credit is limited to some source code verification objectives (i.e., criteria 3 applies)

7.1.3 Limitations and difficulties

Verification objectives applicable to the output of ACG1 (Source Code 1) are defined in DO-178C/ED-12C Annex A table A5. In summary:

a. Compliance with the low-level requirements (Objective A5-1): The objective may be satisfied if the qualified tool is able both to detect that the code is correct from the LLR, and to detect any undocumented functions.

b. Compliance with the software architecture: The objective may be satisfied if the qualified tool is able to detect all instances of non-compliance with the architecture, including data and control flows.

c. Verifiability: The objective may be satisfied if the rules related to verifiability are clearly defined and are verified by the qualified tool.

d. Conformance to standards: The objective may be satisfied by the qualified tool, but is limited to the rules of the coding standard that are verified by that tool.

e. Traceability: The objective may be satisfied if the qualified tool is able to detect all LLR not implemented in the source code. Beyond the tool qualification, this objective needs as a minimum a review of the coding process outputs, typically a log file produced by the autocode generator,

f. Accuracy and consistency: The objective may be partially satisfied by the qualified tool. Some items of this objective may be verified with this process, while others (such as WCET, stack usage, and floating point arithmetic) should be addressed by further verification activities.

- **Tool collection qualification:**

As defined in DO-178C/ED-12C §12.2.1 a tool needs to be qualified "when processes … are eliminated, reduced or automated …" The use of a tool should thus be linked to a process's activity that is performed to satisfy one or several specific objectives.

In this example it is important to emphasize that the two tools, ACG2 and Checker, are qualified as a single tool collection and not separately. This is because the certification credit for using these tools is only defined at the tool collection level:

- ACG2 generates a second version of the source code. This generation is not part of the coding process, and doesn't perform any verification.
- The Checker verifies Source Code 1 based on Source Code 2. This verification doesn't by itself satisfy any verification objective as described above: The equivalence between the two versions of the source code doesn't by itself guarantee that Source Code 1 complies with the LLR, with the software architecture, etc.

- **Limitations:**

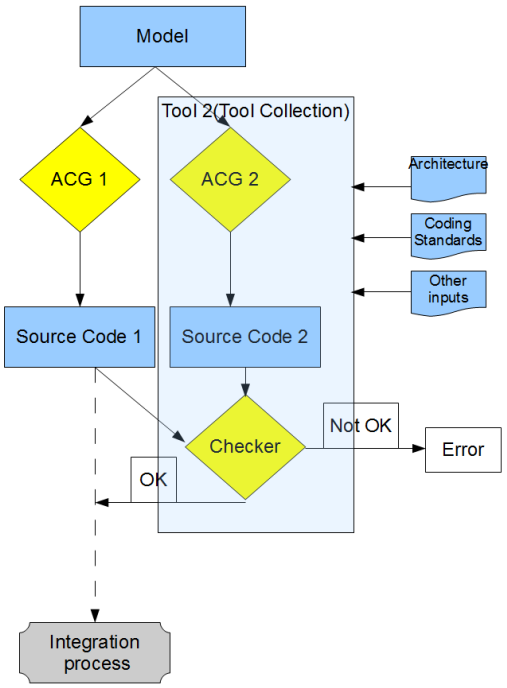
The first difficulty is to determine the **certification credit** for the qualification of Tool 2 (ACG2+Checker) that may be claimed. In other words, which errors in Source Code 1 are detected?

First limitation: The objective "accuracy and consistency" (A5-6) includes topics such as WCET, stack analysis, arithmetic resolution and overflow, that are very difficult to satisfy by source code review only. This is stated in DO-178C/ED-12C §6.3:

There may be cases where the verification objectives described in this section cannot be completely satisfied via reviews and analyses alone. In such cases, those verification

objectives may be satisfied with additional testing of the software product. For example, a combination of reviews, analyses, and tests may be developed to establish the worst-case execution time or verification of the stack usage.

Others limitations concern the capability of Tool 2 to satisfy the source code verification objectives. This is applicable to all Table A-5 objectives. In consequence, Tool 2 must have other input: coding standards, architecture (if not part of the model), additional rules for verifiability, etc.



The Tool User should demonstrate that, through the TOR validation (objective DO-330/ED-215 - 6.2.1.aa), the qualified tool will detect errors related to the objectives defined above. To satisfy this objective, the TOR should be sufficiently complete and accurate to demonstrate the following: for any model allowed by Tool 1 (i.e. based on the modeling standard), and for the resulting source code that is generated (identified as Source Code 1 above), this source code is a correct translation of the model, or, if not, then any errors are detected and identified in the second tool's outputs.

Therefore, the TOR content is the key for claiming that the verification objectives are satisfied through the use of the qualified tool.

- **<u>Tool Operational Requirements considerations:</u>**

Typically the operational requirements for an autocode generator include as a minimum

- Translation requirements: These define the source code and/or data to be generated based on each allowed element of the model
- Sequencing requirements: These define the rules for sequencing each code block in the final code
- Output files, e.g. source code presentation requirements

- Requirements for producing tool execution status and error messages (log information)

Depending of the tool, other requirements may be also defined. Examples:

- Logic conversion and/or numeric calculation on data defined in model elements
- Data declaration or statements for hardware or external software subset compatibility
- The production of additional information or data used for verification activities, for example data accessibility.

The TOR's completeness and accuracy (in particular the definition of the source code sequences to be generated) helps provide sufficient confidence in the correctness of the source code generated by the qualified – criteria 1 - autocode generator.

In this example of using dissimilar tools, there is no credit for the source code generated by Tool 1 as it is not qualified. So we have to consider that this source code

- might not be compliant with LLR
- might not be compliant with the software architecture
- might not be complete
- might not be compliant with coding standards or other rules about verifiability or accuracy
- might include undocumented code or data

The TOR for Tool 2 should thus include all requirements to detect these potential errors. In part this TOR should include all requirements applicable to a qualified autocode generator (translation, sequencing requirements) but reworded for error detection:

As an example of a translation requirement for an autocode generator:

- *For the model element <model-element-id> the generated source code shall be <code-seq>*

Corresponding TOR for Tool 2:

- *Tool 2 shall detect error <error-id> if the source code for model element <model-element-id> is not exactly <code-seq>*

These requirements, if complete, will address the objectives for compliance to LLR and software architecture. But in addition to these requirements the TOR for Tool 2 must also include requirements to satisfy the other verification objectives (completeness, compliance with standards, verifiability).

- **Undocumented functions:**

One issue is how to provide confidence that the source code doesn't include any "extraneous code", e.g. an undocumented function, or non-optimized software, for example creating data that is not used

This is difficult to define in a TOR, except as a general requirement. However, confidence will be based on the accuracy of the requirements and on the related test cases to demonstrate that Tool 2 will detect any extraneous code.

- **Summary:**

The analysis of this example shows that

- The TOR for Tool 2 is not easy to develop, and may be more complex than the development of the TOR for an autocode generator
- The TOR of Tool 2 is based on the certification credit (the objectives to be satisfied), and not on the way the verification is performed. In other words, the TOR is independent of the dual code generation. It is also independent of the method applied to develop the checker: Diff-Checker or Equivalence-Checker.
- The number of tests to be developed based on the TOR may be very large depending on the limitations and complexity of potential models. The tests should demonstrate that for all allowed model elements, in any permitted combination, Tool 2 is able to detect the errors
- A checker that only verifies the correctness of the translation from model to source code will not meet some of the source code verification objectives unless specific additional tool features are implemented (which go beyond a pure comparison).

Despite these difficulties, this approach may be useful in practice. The main concept is the use of an auxiliary toolchain for the automated verification of the output from an unqualified autocode-generator. The use of dissimilar tools is a technical decision when developing the verification approach.

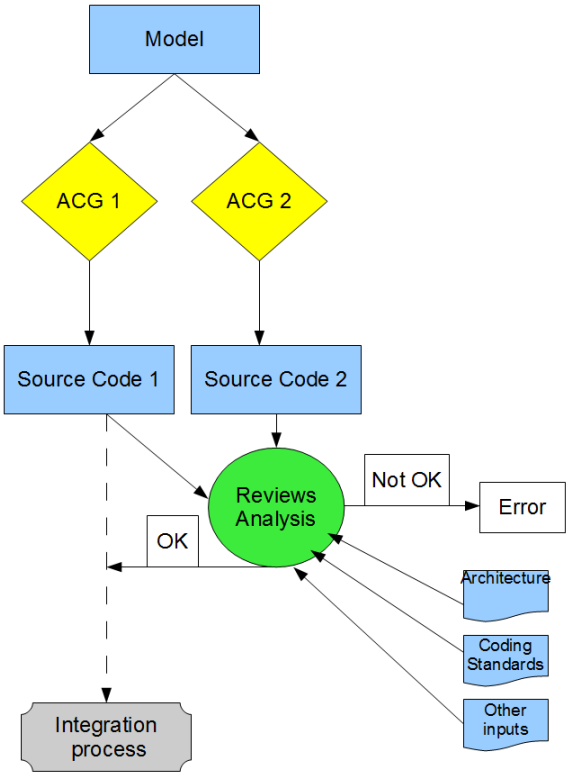
The more complex the input/output of ACG1, the more difficult it will be to develop and demonstrate the coverage of the verification objectives by the TOR of Tool 2. For an ACG1 producing simple (even huge) output, such as configuration files, databases, or other PDI, where it is not so complex to pre-determine the structure of the content of the output, this approach has been successfully applied. But the verification of the translation of a functional design model into source code can generally not be fully automated by a single tool. Most often several tools are used, and are combined with additional manual reviews and analyses to satisfy all the source code verification objectives.

7.2 Example 2: Qualification of the second autocode generator alone, verification performed by manual reviews

7.2.1 Description

This approach is similar to the previous one (use of two autocode generators), except that the verification (the comparison between the two source code) is not automated in a checker tool, but instead is performed through a combination of reviews and analyses.

The non-qualified autocode generator produces the source code that is manually verified. This verification is based on “expected source code” produced by the second autocode generator.



7.2.2 Tool Qualification Levels

ACG2 is considered as an “expected verification result generator”. This tool automates a part of the verification process activity. Instead of performing reviews and analysis of Source Code 1 based on the design model, architecture, and coding standards, the verification is based on Source Code 2.

ACG2 will not introduce an error in the resulting software, but may fail to detect an error. Criteria 1 is not applicable. Criteria 3 is typically applicable, and the certification credit should not be extended beyond the source code verification objective.

Software Level	**TQL for Criteria 1**		
	Single tool	**Dissimilar tools**	
		ACG1	**ACG2**
A	TQL-1	NQ	TQL-5**(*)**
B	TQL-2	NQ	TQL-5**(*)**

C	TQL-3	NQ	TQL-5**(*)**
D	TQL-4	NQ	TQL-5**(*)**

(*) with assumption that the certification credit is limited to some source code verification objectives (criteria 3 applies)

7.2.3 Limitations and difficulties

The approach in this example is

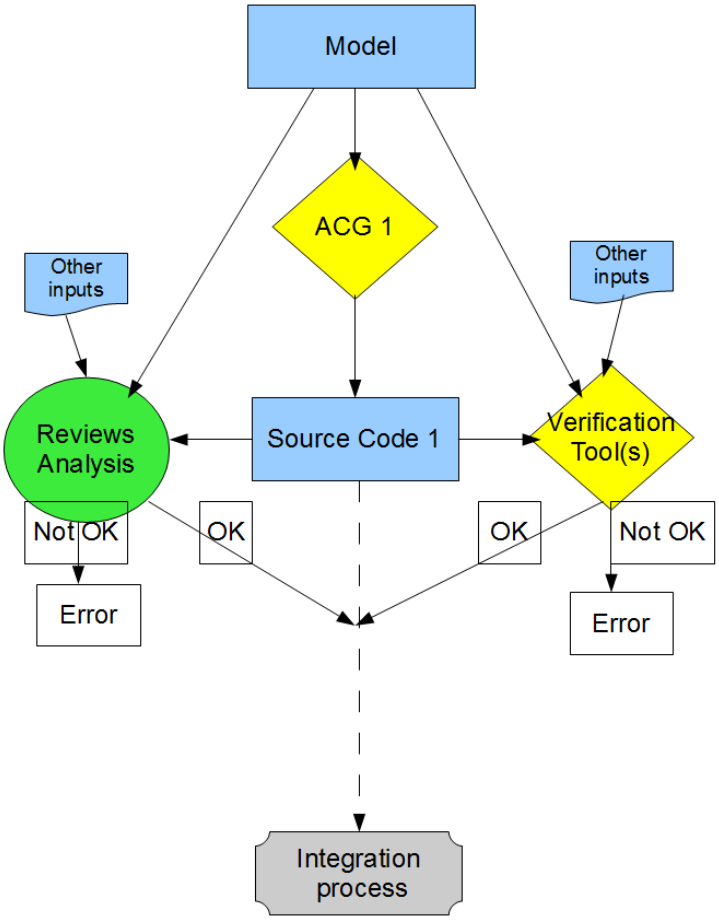
- To avoid the need for qualifying a criteria 1 autocode generator
- To develop a second (and dissimilar) autocode generator, and to qualify it at TQL-5
- To facilitate the manual reviews and analysis using the output of the qualified autocode generator (ACG2)

Similar to example 1, the TOR for ACG2 should include all requirements to detect the potential errors that may be introduced by ACG1 with respect to the verification objectives (compliance with LLR, Software Architecture, coding standards, etc.). But as ACG2 alone doesn't directly detect the errors (reviews and analysis will do this), the TOR for ACG 2 is the same as the one for ACG1. In addition to translation and sequencing requirements, it must also include requirements for the other verification objectives (completeness, compliance with standards, verifiability).

In summary, this approach is not simpler than example 1. It may be used when the source code comparison is difficult to automate. A qualitative assessment is necessary, involving a human activity.

As mentioned in the summary of example 1, a combination of the two approaches is more typical when using an unqualified autocode generator. Verification of the output of the unqualified code generator is partially automated by other tools. Additional reviews and analyses are performed to complete the satisfaction of the verification objectives.

Whichever methods are used to develop the verification tool, the checker (which may or may not be a tool collection with dissimilar tools) needs to be qualified according to criteria 3 (TQL-5). The "other inputs" (identified in the figure below) are whatever data is needed to perform the verification (architecture features, verifiability rules, coding standards, etc.) depending of the work-sharing between the verification tool(s) and the reviews and analysis.



7.3 Example 3: Qualification of the checker alone

7.3.1 Description

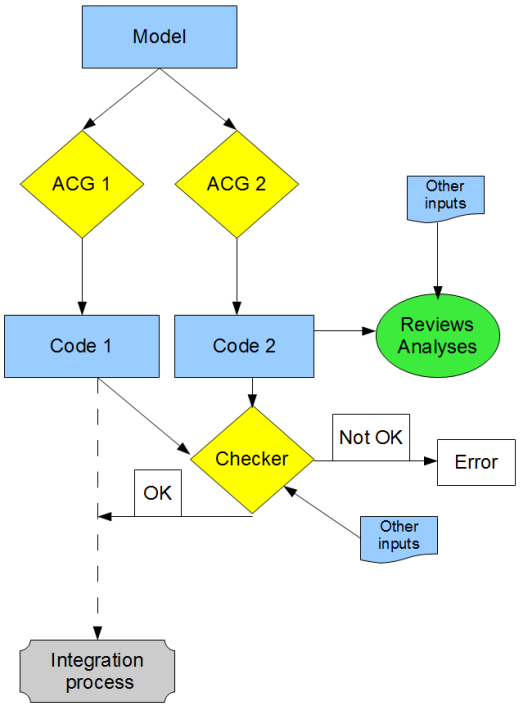
This example is also similar to example 1, but instead of considering ACG2 and the checker together as a tool collection, the output of ACG2 (Source Code 2) is verified by a combination of reviews and analysis. Then the checker verifies Source Code 1 by comparison with the pre-verified Source Code 2.

This approach may be relevant when the design model is translated by ACG 1 into code in a formalism that is required by the integration process but which is judged to be not easily verifiable. The Tool User may then choose to generate from the same design model a second version of the code in another, more easily verified formalism. The checker is then an equivalence checker, since the two versions of the source code do not use the same formalism.

The verification is performed in two steps

1- Source Code 2 is verified by a combination of reviews and analysis (note that some tools may be used as part of these activities)

2- The checker assesses the equivalence between the pre-verified Source Code 2 and Source Code 1



7.3.2 Tool Qualification Levels

As in the two first examples above, the output of ACG1 is verified by the use of the checker, and thus ACG1 doesn't need to be qualified. The output of ACG2 is also verified, but through a combination of reviews and analyses. ACG2 doesn't need to be qualified.

The Checker will not introduce an error in the resulting software, but it may fail to detect an error. Criteria 1 is not applicable. Criteria 3 is typically applicable, and the certification credit should not be extended beyond the source code verification objectives.

Software Level	**TQL for Criteria 1**			
	Single tool	**Dissimilar tools**		
		ACG1	**ACG2**	**Checker**
A	TQL-1	NQ	NQ	TQL-5**(*)**
B	TQL-2	NQ	NQ	TQL-5**(*)**

C	TQL-3	NQ	NQ	TQL-5**(*)**
D	TQL-4	NQ	NQ	TQL-5**(*)**

(*) with assumption that the certification credit is limited to some source code verification objectives (criteria 3 applies)

The output of ACG2 is verified (manually or with other tools), and the checker verifies the "equivalence" between the two generated codes. This verification should demonstrate that the verification performed on Source Code 2 will detect the same errors as if it had been performed on Source Code 1. This demonstration will be based on the content of the TOR developed to qualify the checker as a criteria 3 tool.

7.3.3 Limitations and difficulties

The main difficulty in this approach is how to develop the TOR for an equivalence checker. If the two formalisms (i.e., programming languages) are sufficiently alike, it may be possible to define some sort of syntactic comparison. It will be not the case when languages will differ in terms of semantics, organization, data handling, etc, even if they seem to have similar functionality.

A strong limitation of this approach is also its ability to demonstrate the coverage of verification objectives. As an example, the two versions of the source code are in two different languages, so each one should have its own coding standard. The verification performed on Source Code 2 cannot assess the compliance of Source Code 1 to its own coding standard. A similar problem arises for several other objectives such as architecture and data definition, rules for verifiability, etc. Some very stringent limitations should be imposed on the two autocode generators.

In summary, it is possible to automate the verification of equivalence between two versions of the source code, but there will need to be limitations on the input/output of each ACG. These limitations are driven by the need to be able to define the operational (comparison) requirements of the checker, for all allowed ACG input (design models).

But a pre-condition in this approach is to perform the reviews and analysis on the output of ACG2. So we can compare this approach with direct review/analysis of Source Code 1:

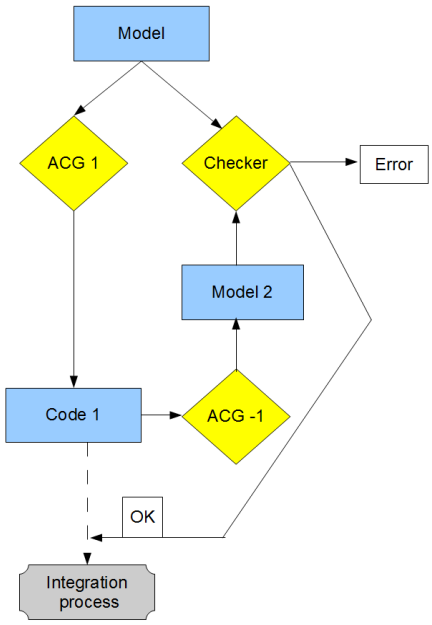
Verification of ACG1	**Use of ACG2 and an equivalence checker**
Tool activities	
Develop ACG1	Develop ACG1
	Develop ACG2
	Develop checker
	Qualify checker
Software activities	
Coding: generate code with ACG1	Coding: generate Source Code 1 with ACG1
	Generate Source Code 2 with ACG2
Verification: Reviews/Analysis of source code 1	Verification: Reviews/Analysis of source code 2
	Compare Source Code 1 and Source Code 2

The assumption in choosing this approach was that Source Code 2 was easier to verify than source code 1. But to implement this approach, the two versions of the source code should be very similar. Thus the effort in verifying one or the other is likely also similar and this approach will therefore not be attractive in general.

7.4 Example 4: Verification performed at model level

7.4.1 Description

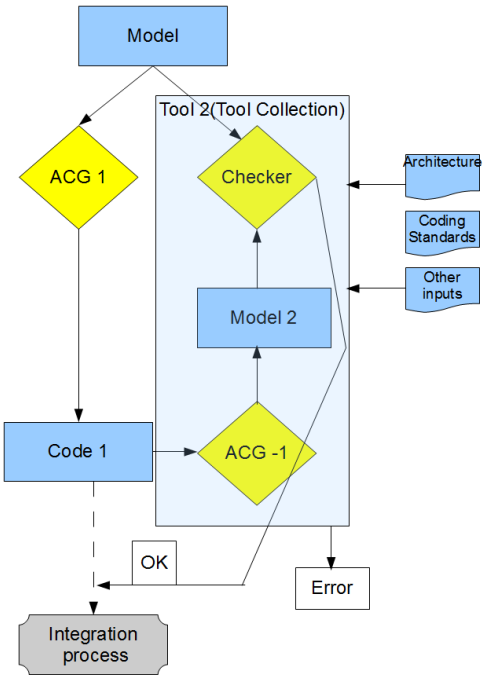
In this example, the second autocode generator is developed based on the “feedback method: It takes the output of ACG 1 and rebuilds the model in a form allowing the verification of complete and correct code generation.



The checker performs the verification using the original Model and also the Model rebuilt by ACG-1 (Model 2). The checker may be a diff checker or an equivalence checker.

In example 1, we explained that the way to implement the verification tool(s) is method independent. This example is only a different technique and different tool architecture for verifying the output of the non-qualified ACG 1.

The picture may be displayed as following:



7.4.2 Tool Qualification Levels

As in example 1, the tool collection (ACG-1 + Checker) should be qualified. Further, it has the same input/output as in example 1, and it should satisfy exactly the same Tool Operational Requirements.

7.4.3 Limitations and difficulties

The same limitations and difficulties stated in example 1 are applicable to this approach

We can also combine this reverse method with other checker technique, with the same issues, advantages, and disadvantages as presented in the previous examples.

8. Use Case n° 2: A Criteria 2 Tool

8.1 Description

A criteria 2 tool is one that automates verification process(es) and thus could fail to detect an error, but also whose output is used to justify the elimination or reduction of:

- Verification process(es) other than that automated by the tool, or
- Development process(es) that could have an impact on the airborne software.

The applicable TQL depends on the software level (TQL-4 for software level A or B, TQL-5 for software level C or D).

This criterion is applicable to a tool verifying some life cycle data, with an extended "certification credit" claimed. For example, this criterion applies to a "proof" tool or static code analyzer whose results are used to justify the elimination of certain tests.

Instead of using a single analyzer tool, a Tool User may use two dissimilar analysis tools. In this case, it means that some software life cycle data is verified twice, if the verification performed by the two tools is the same (and not a complementary verification where the first tool verifies some features, and the second tool verifies others).

To receive credit for such an approach it is necessary to demonstrate the dissimilarity of the two tools. As defined in FAQ D.7, separation and protection between the two tools are needed to avoid the presence of a common error.

Once these precautions are taken, a greater level of confidence may be obtained for the absence of errors in the software life cycle data being verified.

8.2 Tool Qualification Levels

It may be possible to modify the TQL as follows:

Software Level	**TQL for Criteria 2**		
	Single tool	**Dissimilar tools**	
		Tool 1	**Tool 2**
A and B	TQL-4	TQL-5	TQL-5
C and D	TQL-5	TQL-5 or NQ	NQ or TQL-5

For Software Level A and B, the applicable TQL may be lowered to TQL-5 for the two tools.

For Software Level C and D, when using a single tool, the required TQL is TQL-5. It is not possible to lower this level, as the principle of qualification is to ensure that a tool provides confidence at least equivalent to the process eliminated, reduced, or automated. So there is

no credit, even with two tools, that may be given to non-qualified tools when the outputs are not verified. Thus no credit may be given to a process eliminated, reduced, or automated by two non-qualified tools.

However, the Tool User may propose to qualify only one of the two tools at TQL-5. The use of a second tool doesn't influence the need for qualification of one tool at TQL-5.

8.3 Limitations and difficulties

The distinction between criteria 2 and criteria 3 tools is based only on the certification credit claimed. This is explained and illustrated with examples in DO-330/ED-215 FAQ D.5 "What is the rationale for Tool Qualification Criteria definition".

This approach is relevant for software level A or B. The increase of effort between TQL-5 and TQL-4 is that for TQL-5 the qualification doesn't need any data from the Tool Development processes (and related verification activities) as defined in DO-330/ED-215 §5.2. TQL-5 only requires evidences concerning the Tool Operational Requirements definition process and Tool Operational Integration Process.

In the case of a COTS tool, it may be difficult to obtain the data from the Tool Development process, while the data from other development processes should be typically produced by the Tool User. The absence of data from tool developers may prevent qualification at TQL-4.

The use of dissimilar tools to extend the certification credit of this automated verification may address this issue.

Another solution may be the use of a single tool with a suitable service history.

9. Use Case n° 3: A Criteria 3 Tool

9.1 Description

A criteria 3 tool is one that, within the scope of its intended use, could fail to detect an error

The applicable TQL is TQL-5 for all software levels.

Criteria 3 typically applies to a tool verifying some software life cycle data, as far as criteria 2 is not applicable.

A Tool User may use two dissimilar tools instead of using a single verification tool. This case is similar to the example for Criteria 2: the software life cycle is verified twice. Again, this is true only if the verification performed by the two tools is the same (and not a complementary verification where the first tool verifies some features, and the second tool verifies others).

As long as the two tools demonstrate the required level of common cause avoidance, dissimilarity, and protection between tools, a greater level of confidence in the absence of errors in the software life cycle data being verified is obtained.

9.2 Tool Qualification Levels

It may be possible to modify the TQL as follows:

Software Level	**TQL for Criteria 3**		
	Single tool	**Dissimilar tools**	
		Tool 1	**Tool 2**
All levels	TQL-5	TQL-5 or NQ	NQ or TQL-5

For all levels, a Tool User may propose to qualify only one of the two tools at TQL-5. In this case, for the same reason as for criteria 2 tools, no credit may be given to non-qualified tools, as the outputs are not verified - even with two tools. One tool should be qualified, regardless.

9.3 Limitations and difficulties

The discussion for criteria 3 tools is based on the fundamental concept that there is no credit for using an unqualified tool. Thus there is no more credit for using several unqualified tools (n x 0 = 0 for any n). So theoretically, the qualification of a criteria 3 tool cannot be avoid by the use of dissimilar tools; there is no benefit in using two tools instead of one, since at least one of them requires qualification.

Nevertheless, even without any qualification data for the tools, the use of several tools may help to detect error in the data submitted to verification.

In some cases, the use of two unqualified tools has been accepted rather than using and qualifying one single tool. This has occurred with systematic and low-level verification activities such as automating the comparison of two files (for example new test results versus previous test results). This kind of tool (“diff” or equivalent) is commonly used, and in most of the cases qualification is not required. This may be based on the wide usage of this tool (a kind of service history although not really documented). But theoretically this tool “may fail to detect an error” and should be qualified. The problem is that qualifying such a widely used tool (TQL-5) will not really increase the confidence in its use, since errors will likely not be detected by the qualification activities. In such cases, the use of two dissimilar (and unqualified) tools may be an answer.

10. Dissimilar tools used in the tool qualification process

10.1 Description

When a tool is used in the software life cycle process, its need for qualification and the applicable TQL are determined based on DO-178C/ED-12C §12.2. In the scope of the tool verification process and/or Tool Operational verification and validation process, the test environment may include a dissimilar tool. This may be a mean to automate the verification of the tool test outputs.

In this case the need for qualification of the second tool is not assessed with regard to the software life cycle process, but with regard to the Tool life cycle process. DO-330/ED-215 §4.4.e applies.

"An assessment on all the tools used in the framework of the tool life cycle processes should be conducted in order to identify the need for qualification of these tools. Qualification of these tools is needed when processes of this document are eliminated, reduced, or automated by the use of a tool without its output verified as specified in section 6. For a tool that can introduce an error in the outputs of a tool, the applicable TQL is the same as the tool being developed. For a tool that cannot introduce an error in the output of the tool, but may fail to detect an error in the tool life cycle data, the applicable TQL is TQL-5. "

10.2 Tool Qualification Levels

The applicable TQL should be defined as followed

Software Level			
	Single tool	**Dissimilar tools**	
		Tool 1	**Tool 2**
All levels	TQL-1 to 4 (*)	Same	TQL-5 (Note 2)
All levels	TQL-5 (*)	Same	NQ (Note 1)

(*) As determined by DO-178C/ED-12C §12.2 with regard to the applicable criteria and software level

Tool 2 is used only in the context of Tool 1 verification or validation activities, so it cannot introduce an error. Thus TQL-5 typically applies to this tool. In some cases, it may also be possible to use this tool without qualification. This can arise:

- Note 1: When Tool 1 is qualified at TQL-5. The most important aspects of demonstrating confidence in the tool are the accuracy and completeness of TOR and the representativeness of the tests performed, rather than the environment in which the verification activities are performed.

- Note 2: When Tool 2 is used for the validation activities. The validation tests complete the verification tests. Validation tests are performed based on representative samples of tool inputs. Verification tests are performed based on the TOR and TR. That's why the use of a dissimilar tool may be considered as a relevant approach to optimize error detection.

10.3 Limitations and difficulties

The use of a dissimilar tool in the context of the qualification process of another tool should still be compliant with the constraints defined in chapter 5.

Of course this use has no impact on the applicable TQL for Tool 1. But this approach avoids having an impact of the software life cycle. The systematic use of two tools in recurrent activities is an additional workload for developers. Usage of the second tool is only a decision of the tool developer to define the methods/environments to qualify the first tool.

Considering the limited number of objectives to be satisfied for TQL-5, this approach may help optimize the TQL-5 tool qualification, if the proposal to not qualify Tool 2 is accepted.

Summary

When dissimilar tools are used in the scope of a software project, the applicable TQL may be modified (lowered). To obtain agreement for such a modification, the first step is to demonstrate the tools' dissimilarity, based on the protection between them, and the absence of common errors.

Due to the absence of available guidance, the possible alleviations of qualification level cannot be easily determined. Such alleviation will need to be discussed with the certification authorities. The rationale for these alleviations, including evidence of dissimilarities, should be documented in the PSAC. This data should be submitted to the certification authority as early as possible.

This paper discussed potential alleviations depending of the kind of tools and possible use cases. However, it was not possible to address all the technical issues arising from the usage of dissimilar tools. Thus the decision on the allocated TQL for each tool will likely need to be determined on a case by case basis.

But based on an analysis of several examples illustrating multiple methods for developing and using dissimilar tools, our conclusion is that the benefits of such approaches are generally very low.

- For criteria 3 tools, there is no benefit, since one of the tools should still be qualified at TQL-5.

- The major benefit is probably for criteria 2 tools, where two dissimilar tools may be qualified at TQL-5 instead of qualifying one of them at TQL-4 (For software level A or B). This alleviation allows the use of commercial tools despite the potential lack of data from the tool vendors. However, the benefit is small due to the low number of commercial tools of this category.

- For criteria 1 tools, due to the amount of effort for qualification at TQL-1, the use of dissimilar tools might be expected to result in significant savings. However, major limitations exist. The largest impediment is probably the difficulty in developing a checker tool, and in demonstrating coverage of the verification objectives. The checker cannot be a simple "comparison" between the two tool outputs. The equivalence between the two versions of the source code doesn't by itself guarantee that the source code from the nonqualified autocode generator is compliant with LLR, Software Architecture, etc. However, the use of dissimilar tools seems more relevant for a generator of Parameter Data Items (Configuration files, database, etc.) than for functional code.

… for any questions, to ask for a DO-178C/ED-12C or DO-330/215 training, or to propose additional inputs and improvements to this document, please contact :

Frédéric POTHON – ACG SOLUTIONS

(+33) 04.67.60.94.87 – (+33) 06.21.69.26.80

frederic.pothon@acg-solutions.fr

www.acg-solutions.fr