Stood and DO-178B

Technical Note

Pierre Dissaux,
TNI-Europe Limited

Mountbatten Court, Worrall Street
Congleton CW12 1DT
UK
+44 (0) 1260 291449
pierre.dissaux@tni-world.com

1 Introduction

Stood is the state of the art solution for mission critical software development in avionics, space, defense, ground transportation and other demanding industrial systems. **Stood** and his companion tool **Reqtify**, offer a full support for high quality ground or embedded real-time software projects in Ada or C/C++.

Stood is the result of a multi-year experience and collaboration with major European clients on high-profile industrial projects, such as the Airbus multi-generation aircrafts, the Tiger helicopter and a wide range of satellites. Leveraging from this experience, **Stood** and **Reqtify**, offer comprehensive and powerful support for the Software Requirements and Software Design activities, as specified in domain-related international industrial standards, such as **DO-178B**, ECSS-E40, ISO/IEC-12207 and EN-50128.

This technical note presents how **Stood** and **Reqtify** used together, comply with the **DO-178B** objectives.

2 Coverage of DO-178B objectives

2.1 Software Planning Process (Section 4)

2.1.1 Software Planning Process Objectives (Section 4.1)

"... The objectives of the software planning process are:"

4.1.a. "The activities of the software development processes and integral processes of the software life cycle that will address the system requirements and software level(s) are defined"

Stood may be included into any existing software development environment. In addition, it is highly customizable, especially to interact with Software Configuration Management (use of a file system database, customization of the interface with configuration management tools, support of change management) and Software Quality Assurance (customizable tools to perform rules compliancy checks, requirements traceability).

4.1.b. "The software life cycle(s), including the inter-relationships between the processes, their sequencing, feedback mecanisms, and transition criteria are determined"

Stood covers a well identified process in the life-cycle. Forward and reverse bridgers between upsteam and downstream tools provide a high flexibility in defining a full support of the development activities, including waterfall and evolutionary approaches. Inter-relationships and transition criteria are well defined and documented, even if they're customizable, and sequencing may be clearly expressed by the use of successive tools or successive steps inside a given tool.

4.1.c. "The software life cycle environment, including the methods and tools to be used for the activities of each software life cycle process have been selected"

Stood offers appropriate methods support and tools for the Design Process, the Coding Process, part of the Verification Process and full requirements traceability throughout the software life cycle.

4.1.e. "Software development standards consistent with the system safety objectives for the software to be produced are defined"

Stood is compliant with SW development standards upto critical level, due to a large experience in aerospace software developments.

2.1.2 Software Development Environment (Section 4.4.1)

"... Guidance for the selection of software development environment methods and tools includes:"

4.4.1.a. "During the software planning process, the software development environment should be chosen to minimize its potential risk to the final airborne software"

Stood is used for the architectural and detailed design phases, and has been developed upon specific requirements to support aerospace software developments. Moreover, it has been used for more than ten years within such projects. In addition, **Reqtify**, the transversal requirements traceability tool of the suite has been set up to avoid redundancies within the project database.

4.4.1.b. "The use of qualified tools or combinations of tools and parts of the software development environment should be chosen to achieve the necessary level of confidence that an error introduced by one part would be detected by another. An acceptable environment is produced when both parts are consistently used together"

Point to point communication features have been developed between **Stood** and its upstream and downstream tools. In addition, the requirements traceability tool, **Reqtify**, can check information handled by all the tools in the development chain. **Stood** is used upto critical level software developments, so have a huge certification credit, and **Reqtify** is **DO178-B** qualified as a verification tool (**DO178-B** section 12.2.2).

4.4.1.c. "Software verification process activities or software development standards, which include consideration of the software level, should be defined to minimize potential software development environment-related errors"

Stood is based on semi-formal techniques (i.e. HOOD, UML, AADL). That means that an extensive use of good software engineering practices contributes to a better control of the software architecture by the development team. However, sensible parts of the software code may not be generated automatically if required to minimize the impact of the tools potential discrepencies.

2.2 Software Development Processes (Section 5)

2.2.1 Software Design Process (Section 5.2)

"The objectives of the software design process are:"

5.2.1.a. "The software architecture and low-level requirements are developed from the high-level requirements"

Stood is used to perform the design process activities. This tool complies with the HOOD design method (Hierarchical Object-Oriented Design), that is based on a modular hierarchical top-down decomposition. A complete list of high-level requirements may be imported during the design activities. Then graphical edition is used to build the software architecture in terms of well defined interacting modules. The standard design rules enforce high cohesion and low coupling. Textual edition, fully consistant with the graphical architecture, is used to support the detailed design activities and identify the low-level requirements.

5.2.1.b. "Derived low-level requirements are indicated to the system safety assessment process"

Stood includes a requirements management feature and can be connected to **Reqtify**. Any reference to a known high-level requirement, and any derived low-level requirement may be explicitely identified and highlighted by **Reqtify** and documented in the traceability matrix.

2.2.2 Software Coding Process (Section 5.3)

"The objective of the software coding process is:"

5.3.1.a. "Source code is developed that is traceable, verifiable, consistent, and correctly implements low-level requirements"

The textual notation used to perform the HOOD detailed design activity includes specific sections to either insert the applicative source code, either express the coding requirements with a pseudo-code. Each coding section is fully consistent with the detailed design data structure encompassing the low-level requirements. Automatic code generation can be used to produce a design-consistent source code skeleton. The "round-trip" engineering feature of **Stood** can be used to maintain design and code consistency throughout the coding process.

2.2.3 Integration Process (Section 5.4)

"The objective of the integration process is:"

5.4.1.a. "The Executable Object Code is loaded into the target hardware for hardware/software integration"

The software interface of the target hardware, as well as the required libraries may be described during the design process activities, and managed by the automatic code generator during the coding process activities. Additionally, the code generator of **Stood** may be customized to fully support any compilation and linking chain, and thus facilitate the integration process.

2.3 Software Verification Process (Section 6)

2.3.1 Review and Analyses of the Low-Level Requirements (Section 6.3.2)

"... These reviews and analyses confirm that the software low-level requirements satisfy these objectives:"

6.3.2.a. "Compliance with high-level requirements: The objective is to ensure that the software low-level requirements satisfy the software high-level requirements and that derived requirements and the design basis for their existence are correctly defined"

Stood can ask **Reqtify** to compile all the high-level requirements that must be satisfied , and keep them available all along the design activities. The step by step top-down modular decomposition process recommended by the HOOD method enforces a rigorous and consistent design structure to support the requirements. At any level in that hierarchy, the standard description framework for each HOOD module (the ODS: Object Description Skeleton) includes a set of sections to describe the compliancy with high-level requirements and justify the existence of derived requirements:

- Statement of the problem
- Referenced documents
- Structural requirements
- Functional requirements
- Behavioural requirements
- Justification of design decisions

6.3.2.c. "Compatibility with the target computer: The objective is to ensure that no conflicts exist between the software requirements and the hardware/software features of the target computer, especially, the use of resources (such as bus loading), system response times, and input/output hardware"

In HOOD, the hardware/software features of the target computer are represented by Environment Objects which interface is precisely defined. The cross-reference table provided by **Stood** also enables the verification of all the interactions between the applicative modules and the Environment Objects. Finally, a section of the ODS can be used to specify the Implementation Constraints.

6.3.2.d. "Verifiability: The objective is to ensure that each low-level requirement can be verified"

It is recommended by the HOOD method that the low-level requirements map software entities of the same module (i.e. operations, types, constants, exceptions, data). All these entities are part of the **Stood** design database and can be accessed individually and documented. It is also possible to use the pseudo code sections to better formalize the low-level requirements, and thus to provide the ability to perform external verifications. In addition, **Reqtify** can be used to analyse the test cases and check the unit testing coverage.

6.3.2.e. "Conformance to standards: The objective is to ensure that the Software Design Standards were followed during the software design process, and that deviations from the standards are justified"

Stood complies strictly to the HOOD method, that is formally defined by the HOOD Reference Manual. Most of this compliancy is hard-wired into the edition features of the tool. However, as the underlying model is semi-formal only, there is a need to provide a verification tool to check the compliancy of the current software design towards the rules of the method. This is achieved by the HOOD checker tool that is included into the standard version of **Stood**. In addition, this rules checker may be customized in order to follow any amendment made to the standard rules by the project. Compliancy to emerging standards like UML2.0, defined by the OMG, and AADL (Avionics Architecture Description Language), defined by the SAE, is also provided.

6.3.2.f. "Traceability: The objective is to ensure the high-level requirements and derived requirements were developed into the low-level requirements"

Stood includes its own requirements coverage analysis feature, but **Reqtify** may be used more widely, to collect the high-level requirements from any file or modeling tool, collect the low-level requirements from **Stood**, and provide all the relevant coverage information dynamically. **Reqtify** has been qualified as a verification tool according to section 12.2.2 of the **DO178-B**, for the AIRBUS A380 developments.

2.3.2 Review and Analyses of the Software Architecture (Section 6.3.3)

"... These reviews and analyses confirm that the software architecture satisfies these objectives:"

6.3.3.a. "Compatibility with the high-level requirements: The objective is to ensure that the software architecture does not conflict with the high-level requirements, especially functions that ensure system integrity, for example, partitioning schemes"

The hierarchical top-down modular decomposition process enforced by the HOOD method leads to a precise and very well controlled definition of the software architecture. This iterative process and its associated rules are fully supported by **Stood** and ensure that the high-level requirements are taken into account, and consistently down to the terminal modules of the hierarchy thanks to the Implemented_By relationship of the HOOD method.

6.3.3.b. "Consistency: The objective is to ensure that a correct relationship exists between the components of the software architecture. This relationship exists via data flow and control flow"

The relationships between the components of the software are restricted by precise visibility rules that are controled by **Stood** during the graphical edition and upon specific request to the design checker.
During the HOOD architectural design activities, all the dependencies between the modules must be represented by an appropriate Use relationship. These relationships cover functional dependencies (control flow and associated data flows and exception flows) and structural dependencies (data type inheritance, aggregation and instanciation). During the HOOD detailed design and coding activities, the dependencies are deduced from an appropriate analyses of the pseudo code or code and are used by various processing features of **Stood**:

- Cross-references tables showing low-level dependencies between all the components of the software architecture.
- Automatic update of the Required Interface sections of the documentation framework (ODS).
- Production of graphical dependency graphs (call trees, data access, type structure, …).

- Design checker, to compare architectural design dependencies and detailed design dependencies.
- Code generator, that use this information to automatically insert the right references to the remote modules, and to order properly the declarations in the source code.

6.3.3.c. " Compatibility with the target computer: The objective is to ensure that no conflicts exists, especially initialization, asynchronous operation, synchronization and interrupts, between the software architecture and the hardware/software features of the target computer"

Each software interface of the hardware and software libraries or other remote utility used by the design are represented by Environment Objects that are fully included into the design model. **Stood** can thus check the correctness of any reference to hardware/software feature of the target computer. In addition, these software interfaces are automatically updated each time the design is opened. This ensures that environment changes are taken into account.

6.3.3.d. "Verifiability: The objective is to ensure that the software architecture can be verified, for example, there are no unbounded recursive algorithms"

Each software architecture element is represented by an appropriate entity of the HOOD design hierarchy. This description is automatically updated by **Stood** during the architectural and detailed design activities. Verification post-processors may access this database to perform any appropriate action and may be customized to fit project recommendations. The verification post-processors that are included in the standard version of **Stood** are:
- A HOOD rules checker
- A design metrics checker
- A requirements coverage checker
- A schedulability checker

6.3.3.e. "Conformance to standards: The objective is to ensure that the Software Design Standards were followed during the software design process and that deviations to the standards are justified, especially complexity restrictions and design constructs that would not comply with the system safety objectives"

Stood complies strictly to the HOOD method, that is formally defined by the HOOD Reference Manual. Most of this compliancy is hard-wired into the edition features of the tool. However, as the underlying model is semi-formal only, there is a need to provide a verification tool to check the compliancy of the current software design towards the rules of the method. This is achieved by the HOOD checker tool that is included into the standard version of **Stood**. In addition, this rules checker may be customized in order to follow any amendment made to the standard rules by the project.

6.3.3.f. "Partitioning integrity: The objective is to ensure that partitioning breaches are prevented or isolated"

Thanks to the HOOD method, each software module is perfectly located towards the others within a tree structure that defines the scope of each software component and the visibility rules. This structure and rules are propagated to the source code when the automatic code generator of **Stood** is used. Additionally, it is possible to export a complete subsystem, represented by a branch of the tree, in order to delegate the corresponding development activities outside the scope of the main system. It is also possible to re-import this subsystem without breaching the structural and visibility rules of the overall design.

2.3.3 Review and Analyses of the Source Code (Section 6.3.4)

"... The topics should include:"

6.3.4.a. "Compliancy with low-level requirements: The objective is to ensure that the Source Code is accurate and complete with respect to the software low-level requirements, and that no Source Code implements an undocumented function"

The use of the automatic code generation feature of **Stood** ensures that the source code will be structured with a one to one mapping with the design entities supporting the low-level requirements. The design structure that is defined for each module (ODS), also contains the documentation, and a visual indicator shows the completeness of this design structure, and especially the documentation sections:
- A red tick indicates that required information is missing.
- An orange tick indicates that required information is uncomplete.

- A green tick indicates that all the required information is present.

Alternatively, **Reqtify** may be used to analyse the design documentation and the source code, and to check that no source code implements an undocumented function.

6.3.4.b. "Compliancy with the software architecture: The objective is to ensure that the Source Code matches the data flow and control flow defined in the software architecture"

Following the rules of the HOOD method, every access to remote data must be encapsulated by operations. The consequence of that rule is that, at design level, all data flows and control flows between modules can be represented graphically by the Use relationships, and textually by the operation declarations (list of typed parameters) and calls. These operation declarations are defined formally in **Stood** (with an Ada like syntax) and are generated in the source code. In addition, it is recommended either to perform the coding activity through the design tool, or to use the round-trip engineering feature to feed the operations body code back to the design structure. It is then possible to also check the control flows defined in the source code comply with the software architecture…

6.3.4.c. "Verifiability: The objective is to ensure that the Source Code does not contain statements and structures that cannot be verified and that the code does not have to be altered to test it"

By customizing the code generator of **Stood**, it is possible to include into the generated source code, any appropriate comment, directive or code, such as conditional compilation, to breach the visibility rules for testing purpose.

6.3.4.d. "Conformance to standards: The objective is to ensure that the Software Code Standards were followed during the development of the code, especially complexity restrictions and code constraints that would be consistent with the system safety objectives. Complexity includes the degree of coupling between software components, the nesting levels for control structures, and complexity of logical or numeric expressions. This analysis also ensures that deviations to the standards are justified"

For the part of the code that is automatically generated, it is possible to customize the code generator of **Stood** in order to comply with the software code standards. Additionally, the HOOD design process is based on the high consistency and low coupling principles, and the design structure is automatically propagated to the source code by the code generator.

6.3.4.e. "Traceability: The objective is to ensure that the software low-level requirements were developed into Source Code.

This objective may be reached either by using the automatic code generator of **Stood**, or by using **Reqtify**, the requirements traceability tool, or both. **Reqtify** has been qualified as a verification tool according to section 12.2.2 of the **DO178-B**, for the AIRBUS A380 developments.

6.3.4.f. "Accuracy and consistency: The objective is to determine the correctness and consistency of the Source Code, including stack usage, fixed point arithmetic overflow and resolution, resource contention, worst-case execution timing, exception handling, use of uninitialized variables or constants, unused variables or constants, and data corruption due to task or interrupt conflicts"

It is recommended by the HOOD method to insert the source code into the design structure either by direct coding or by round-trip engineering. This code may thus be analysed and processed by the verification utilities offered by **Stood**. Static lexical verification are especially provided with the cross-references table and the HOOD rules checker.

3 References

1. RTCA, *Software Considerations in Airborne Systems and Equipment Certification (DO-178B*), 1992.
2. ISO/IEC, *Information technology, Software life cycle process* (ISO/IEC 12207), 1995
3. ECSS, *Space Engineering: Software (ECSS-E40B*), ESA Publication, 2000.
4. HOOD User Group, *HOOD Reference Manual release 3.1*, Masson & Prentice-Hall, 1993.
5. HOOD User Group, *HOOD Reference Manual release 4.0*, HUG, 1995.
6. A. Burns, A. Wellings, *HRT-HOOD: A Structured Design Method for Hard Real-Time Ada System*s, Elsevier, 1995
7. J.P. Rosen, *An Industrial Approach for Software Desig*n, HUG, 1997.
8. P. Dissaux, *HOOD4 and Ada9*5, Proceedings DASIA conference, Lisbon 1999.
9. T. Vardanega, *Development of On-Board Embedded Real-Time Systems: An Engineering Approac*h, ESA Technical Report STR-260, 1999.
10. P. Dissaux, *Real-Time C Code Generation from a HOOD Desig*n, Proceedings DASIA conference, Montreal 2000.
11. P. Dissaux, *HOOD Patterns*, Proceedings DASIA conference, Nice 2001.
12. P. Farail, P. Dissaux, *COTRE: a new approach for modeling real-time software for avionics*, Proceedings DASIA conference, Dublin 2002.
13. P. Dissaux, *HOOD and AADL*, Proceedings DASIA conference, Prague 2003
14. SAE, *Draft Avionics Architecture Description Language (AADL)*, AS2C, 2003