Qualifying Software Tools, a Systems Approach

Fredrik Asplund, Jad El-khoury, and Martin Törngren

KTH Royal Institute of Technology, Department of Machine Design, Division of Mechatronics, Brinellvägen 83, 10044 Stockholm, Sweden

Abstract. Modern safety standards designed to ensure safety in embedded system products often take a descriptive approach, focusing on describing appropriate requirements on management, processes, methods and environments during development. While the qualification of software tools has been included in several such standards, how to handle the safety implications of tools integrated into tool chains has been largely ignored. This problem is aggravated by an increase both in automation of tool integration and the size of development environments.
In this paper we define nine safety goals for tool chains and suggest a qualification method that takes a systems approach on certifying software tools as parts of tool chains. With this method, software tools are developed and pre-qualified under the assumption that certain properties will be supported by the development environment they are to be deployed in. The proposed method is intended to (1) achieve a stronger focus on the relevant parts of tool chains in regard to safety and (2) separate the extra effort these parts imply from the effort already stipulated by safety standards.

Keywords: Certification, Safety, Tool Integration

1 Introduction

The development of embedded systems is a multi-disciplinary effort, undertaken by organizations working with software tools provided by external suppliers. These tools are often *Commercial Of The Shelf* products, i.e. *generic* software programs designed to function on as many operating systems as possible, built to support a multitude of different use cases, etc. This generalization has its cost in relation to safety, since it makes it harder to ensure that tools operate flawlessly when deployed in a specific context. There is a risk that software tools introduce errors in the development artifacts during the development process. These errors may then lead to *hazards*[1] in the end product. This has prompted the introduction of guidelines on the qualification of software tools in several safety standards [1]. Unfortunately the guidelines focus on qualifying tools in isolation, leaving the importance of the interaction between tools largely ignored. This omission has

[1] A hazard can be defined as a system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to an accident.

been of less importance in the past, since related safety issues could then be handled by restrictions on the processes used by the operators[2] manually handling these interactions. However, modern development environments are introducing an increased support for automated tool integration, decreasing the possibility for operators to monitor and act on safety issues due to tool integration.

In this paper we present nine safety goals and propose a method for qualifying software tools as parts of tool chains, which together highlight the hitherto obscured safety issues caused by tool integration and allow for being more exact when identifying software in need of qualification during certification. The safety goals and method build upon systems thinking [2] to approach a modern development environment as a *whole*, i.e. they do not analyze parts of development environments in isolation (which would risk failing to take the relationships between the parts into account).

In Section 2 we present the relevant *State of the Art* to orient the reader within the field of software tool qualification. In Section 3 we describe the domain of tool integration and the possibilities to take a systems approach to allow for the qualification of software tools as parts of tool chains. This is followed in Section 4 by a summary of a technical report in which we explored the relationship between tool integration and safety through a detailed case study. The results from this report, after being put into a system context in Section 4 through a mapping into safety goals, allow us to propose a method for qualifying software tools as parts of tool chains in Section 5. Conclusions are found in Section 6.

2 State of the Art

Several modern safety standards such as IEC 61508:2010 [3] (and domain adaptations of this standard such as IEC 61511:2003 [4], ISO 26262:2011 [5] and EN 50128:2001 [6]) and DO-178C [7] increasingly deal with the issue of software tool qualification (noticeable both when comparing between standards and when comparing between different versions of the same standard). This has lead to a large effort by the scientific community on analyzing tools, categorizing tools, discussing how to certify tools, etc. ([8] gives examples related to DO-178B). Much of the effort seems to be focused on finding a good combination of tools and then enabling them to communicate in a reliable way, while the safety implications of the tool integration is not explicitly discussed (see [9] for an example).

This is not surprising, since either the safety standards themselves or the discussion in relation to them try to limit qualification efforts to avoid software associated primarily with tool integration (see Subsection 3.1 for a detailed discussion on this subject). This means that there is a limited number of approaches on how to benefit from the fact that tools are deployed in tool chains. In fact, we could only identify one such approach, which suggests the use of *reference*

[2] This text uses the word operators in the generic sense, i.e. to indicate anyone who at some point of time is involved in the day-to-day matters of a development effort (such as engineers, tool chain developers, managers, etc.).

tool chains[3] [1]. There has been little effort to analyze the implications on safety due to tool integration, leaving methods and metrics for evaluating different approaches to tool integration for qualification purposes largely unknown.

3 Tool Integration and Software Tool Qualification

To develop an embedded system there is typically a need for a multitude of *engineering tools*, i.e. tools that provide functionality to fulfill one or more activities without which the end product cannot be cost-efficiently developed. Examples of engineering tools include requirements tools, design tools and test tools.

Each engineering tool executes in the context of a *development environment*, defined as the integrated set of all other tools and any supporting software used during development. An important aspect of a development environment is that the *development processes* it supports will define orderings of the engineering tools it contains, i.e. *tool chains*, which the development of a product will transit through. *Tool integration* can be defined as what supports the development process in correctly moving from one engineering tool to another in a tool chain. Examples of tool integration include data transformation tools, scripts that react to the actions of a user and process engines. When discussing tool chains, tool integration software is usually *implicitly* included as something needed "behind-the-scenes" to enable the transition between engineering tools.

3.1 Qualification of Software Tools

There are a number of classification schemes for safety standards, although standards are not always possible to strictly assign to one category. For the discussion in this paper we differ between two types of safety standards, those that take a *primarily prescriptive* approach and those that take a *primarily descriptive* approach on how to enforce safety. By prescriptive standards we refer to those that focus on giving an exhaustive list of product features that a safety-critical product should exhibit (for instance CS-VLR:2008 [10]). By descriptive standards we refer to those that instead focus on defining requirements on appropriate environments, methods and processes to develop safety-critical products (for instance ISO 26262:2011, EN 50128:2001 and DO-178C).

Due to the fact that malfunctions in tools during development can lead to the introduction of hazards in the end product, some of the descriptive standards contain guidelines on qualification of software tools. For this reason, we focus in this paper on the latter set of standards, where assurance is partly provided by an evaluation on how the product was developed. For certification according to the former set of standards the discussion in this paper is only of reference value, since assurance according to these standards is typically provided by inspection or a means detailed in the standard itself.

[3] A reference tool chain is a pre-qualified set of tools and an associated workflow, which only requires a limited qualification effort in regard to the changes made during deployment.

Tool qualification guidelines can take different forms, such as requiring tools to be suitable and of a certain quality [6], or requiring the development of relevant tools to fulfill the same objectives as the development of the products handled by the standard itself [7], etc. The many different forms are not surprising, since the standards state different objectives for qualifying tools. Nevertheless, from such standards as ISO 26262:2011 and DO-178C one can deduce *generic safety goals* for software tool qualification:

- No tool shall introduce errors in development artifacts that may lead to hazards in the end product.[4]
- No tool shall fail to detect errors in development artifacts it is designed to analyze or monitor that may lead to hazards in the end product.[5]

In practice one can discern between two approaches to tool integration in relation to these safety goals.

- The approach that allows a stricter limitation of the qualification effort, exemplified by for instance DO-178C. In DO-178C the objective of the qualification effort is only to ensure that tools provide confidence at least equivalent to that of relevant process(es) they *eliminate*, *reduce* or *automate* [7].
- The approach that strives towards the same generic applicability, exemplified by ISO 26262:2011. One could interpret this standard to indicate that almost everything in the development environment has to be qualified [11] (including all *tool integration mechanisms*[6]).

Both of these approaches are associated with practical problems.

In the first approach one needs to draw a line between tools that need to be qualified and other parts of the development environment that only warrant attention in an indirect fashion. In traditional development environments this was acceptable, since they consisted of separate tools between which users handled the transition of the development manually. This meant that the control of processes and methods could ensure safety to a high degree. Modern development environments can consist of several hundreds of tools [11] (albeit the number of engineering tools is probably far less). Hazards can, in such a development environment, be introduced into the end product from such diverse sources as contradicting data transformations between artifacts at different development phases, scripts causing tools to be executed in the wrong order, the wrong version of data being handed over to the next process activity, etc. These sources can be difficult to identify when the focus of a qualification effort is primarily focused on separate tools, since they are not necessarily directly associated

[4] The introduction of errors in development artifacts can lead to hazards in a direct fashion (i.e. if the artifact is used directly, like the output from a compiler) or in an indirect fashion (i.e. if the artifact is used indirectly, like the output of a high-level design tool).

[5] Tools can be assigned to handle the output of other tools or include functionality to verify its own output.

[6] A tool integration mechanism is a tool integration software (or a part of it) that provides a distinct integration functionality.

with any of the actions of a particular tool (if the output of a certain tool is later transformed, that action does not necessarily have to be associated with the tool itself during qualification). Furthermore, the control of these sources through processes and methods is aggravated by the difficulty users usually have with comprehending automation [12]. This leads to the problem that as development environments scale up and become more integrated, the first approach becomes less sufficient for ensuring safety.

In the second approach the qualification effort becomes tremendous in a modern development environment [11]. If one keeps to this approach the likely outcome is sweeping generalizations of the danger posed by most parts of the development environments at hand. This leads to the problem that as development environments scale up and become more integrated, the second approach also becomes less sufficient for ensuring safety.

The reason for these mutually exclusive approaches sharing the same practical problem is that they stem from the miscomprehension that the increased integration of software tools through automation is nothing more than the introduction of more software. To overcome this miscomprehension we need a third approach which analyzes the result of the increased integration of tools as a whole (i.e. as highly integrated tool chains) and not simply as a collection of separate parts. We need to take a systems approach to tool integration.

3.2 A System Approach to Tool Integration

To take a systems approach to tool integration we have defined a *hierarchy* of levels of organization[7], which reflects the state of a likely development effort. In such a hierarchy, an upper level is more complex than the one below and characterized by having emergent properties[8]. These properties are given by the relationships of the components at the lower levels, which in turn are *controlled* by the upper levels through the *constraints* they impose. Our hierarchy is illustrated by the conceptual illustration of *Figure 1.*

– The top level of our hierarchy consists of the management of the development effort. This *management level* imposes constraints on the next level, the *operator level,* by for instance establishing a safety culture.

[7] We recommend [2] for an overview of *systems theory*, [12] for how systems theory relates to safety and [13] as an introduction to *hierarchy theory.*

[8] In systems theory an emergent property at one level of organization cannot be predicted even by a thorough understanding of the parts of lower levels. Such a property is meaningless at lower levels, i.e. it is irreducible. The opposite are composable properties, which can be shown to originate from one or several parts clearly discernible at lower levels. Checkland uses the genetic coding of DNA as an example of an emergent property [2], since it cannot be explained at the level of the bases (i.e. at the chemistry level any arrangement of nucleobases is allowed, yet not all are meaningful at the higher level of biology). A related composable property is the inability of humans to cope with too much ultraviolet light, since this can be directly attributed to the way two adjacent thymine or cytosine nucleobases may form pyrimidine dimers (which cause skin cancer).

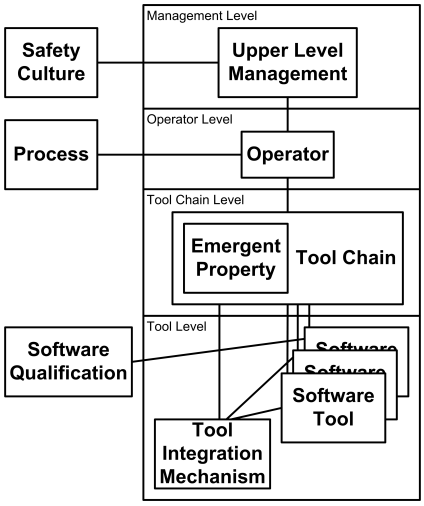

Fig. 1. A hierarchy of levels of organization for development efforts

- The operator level consists of the separate operators. It imposes constraints on the next level, the *tool chain level*, through for instance the processes actually used during development.
- The tool chain level consists of the tool chains used during development. It imposes constraints on the next level, the *tool level*, by for instance specifying the order of the engineering tools.
- The tool level consists of the tools and any supporting software used during development (such as tool integration mechanisms). At the tool level safety is ensured by software qualification, constrained by the requirements on this activity by higher levels (as mentioned in Subsection 3.1).

In earlier publications we have searched for safety-related *characteristics* of tool chains [14],[15], i.e. properties that both decrease the possibility of introducing hazards into end products and are relevant for more than one part of a tool chain (i.e. more than an individual tool). In the next section we put these characteristics into the context of certification and then continue with proposing a method for qualifying software tools as parts of tool chains in Section 5.

4 From Characteristics to Safety Goals

In [15] we performed a *System-Theoretic Process Analysis* (STPA) [12] of three different versions of a tool chain used in an industrial case study for the development of embedded, closed-loop control systems ([15] includes a detailed account of a tool chain that exhibits the safety issues discussed below, an account which could not be included here due to space limitations). STPA defines generic risks for inadequate control or enforcement of safety constraints that can

lead to hazards. We first translated these generic risks into risks associated with tool integration by use of a reference model for tool integration that takes tool integration related to platform, control, data, presentation and process into account (the details of this reference model are given in [16]). The risks associated with these aspects of tool integration were then further translated into *programmatic risks* (i.e. risks associated with the specific tool chain we were analyzing). Based on programmatic risks, STPA provides guidance on identifying their causes through general casual factors defined for the various parts of a generic control loop. We substituted this generic control loop for context-specific ones, for instance one through which operators and tool integration mechanisms (the controllers) can introduce hazards in the end product (the controlled process) through development artifacts (the actuators), hazards which can be identified through verification and validation activities (the sensors). Analysis of the subsequently identified causes allowed us to define nine safety-related characteristics of tool chains [15].

To put the characteristics in the context of certification we below map each characteristic to a safety goal for which assurance could be required. Additionally, we also prepare for the discussion on assurance in Section 5 by grouping the safety goals into subcategories. First we divide all safety goals according to whether they are *fully composable* or *emergent* [17]; secondly we divide the fully composable safety goals according to whether they support *manual* tool integration or *automate* it.

Examples of risks identified in [15] are provided below together with the safety goals which will mitigate the causes of said risks. These risks are often critical research areas in themselves, but to discuss them in detail are outside the scope of this paper.

4.1 Fully Composable Safety Goals for Operator Support

The safety goals in this subsection are fully composable to subgoals at the tool level and support manual tool integration (i.e. ensuring them will require an effort to ensure relevant properties of distinct parts at the tool level and the proper behavior of involved operators).

Traceability for Completeness and Consistency (TCC). Relevant parts of a tool chain shall support the possibility to trace between artifacts to ensure that those artifacts are consistent and complete in regard to each other. An associated risk is the limitation of feedback on which parts of the development artifacts correspond to parts of the end product where hazards have been introduced.

Well Defined Data Semantics (WDDS). Relevant parts of a tool chain shall use unambiguous data semantics. An associated risk is inconsistencies in development artifacts due to uncertainty on the part of operators regarding the semantics of other engineering domains.

Possibility to Create Customized GUIs (PCCG). Relevant parts of a tool chain shall support the creation of additional GUIs for tools, GUIs that are either simplified or adapted to operators from a different engineering domain (for instance a customized GUI for a requirement tool that simplifies the interaction

for use by operators other than requirement engineers). An associated risk is the introduction of errors in development artifacts due to unsafe experimentation by non-expert operators.

Coherent Time Stamp Information (CTSI). Relevant parts of a tool chain shall support common time stamps on development artifacts. An associated risk is the use of incomplete versions of artifacts by operators who misunderstand how recently they were created.

Process Notifications (PN). Relevant parts of a tool chain shall support event notification to operators. An associated risk is the use of obsolete development artifacts by operators unaware of the release of new versions.

4.2 Fully Composable Safety Goals for Automation

The safety goals of this subsection are also fully composable to subgoals at the tool level, but their implementation will not involve operators (i.e. they only require an effort to ensure relevant properties of distinct parts at the tool level).

Automated Transformations of Data (ATD). Relevant parts of a tool chain shall support the automated transfer of data between tools. An associated risk is the incorrect reproduction of artifacts by tired or untrained operators.

Possibility to Automate Tool Usage (PATU). Relevant parts of a tool chain shall support the automation of tool usage. An associated risk is the introduction of errors in development artifacts due to untrained operators.

4.3 Emergent Safety Goals

The safety goals in this subsection are emergent at the tool chain level. These safety goals cannot be fully ensured through the properties of different parts at the tool level, since they depend on the interaction of parts at that level.

Data Integrity (DI). A tool chain shall ensure that the data used reflects the current state of the development. An associated risk is that obsolete or erroneous versions of development artifacts lead to the incorrect reproduction of data. DI can manifest itself locally (for instance when a file is corrupted by a file system service or a database updates the wrong parameters), but also emerge from how tools are used or interact with each other (for instance when a process engine chooses the wrong version of a data artifact).

Data Mining (DM). A tool chain shall ensure that it is possible to (1) extract all the data necessary to handle all safety goals correctly during development and (2) present this data in a human-understandable form. An associated risk is that operators are not aware that project deadlines are forcing the premature release of artifacts and fail to take mitigating action. Which data needs to and can be extracted emerge from the dynamic interaction between tools (for instance through different sequences of service calls that determine what data can be gathered and when).

5 A Method for Qualifying Software Tools as Parts of Tool Chains

The safety goals defined in Section 4 highlight the hitherto obscured safety issues caused by tool integration, but they would be of limited value if they (1) could not solve the problems with current software tool qualification approaches described in Subsection 3.1 or (2) could not be combined with the qualification efforts stipulated by modern safety standards. In this section we suggest a method for dealing with each of the subcategories mentioned in Section 4 while taking both of these issues into account. This means the safety goals detailed in Section 4 that are relevant to a particular part of a tool chain first need to be identified and then associated with a limited part of the development environment (a part that then needs to be qualified).

5.1 Fully Composable Safety Goals for Operator Support

Fully composable safety goals for operator support could be ensured by the brute-force approach of analyzing each and every tool integration mechanism that fulfills a related subgoal, but with similar disadvantages to those described for the second approach in Subsection 3.1. To limit and guide the qualification effort we take inspiration from the tool chain approach in [1] and the *Safety Element out of Context* concept from [5] and suggest the following four steps:

1. *Pre-qualification of engineering tools* is performed by tool vendors based on *representative* tool use cases and a relevant safety standard.
2. *Pre-qualification at the tool chain level* by tool vendors or certification agencies is made possible by the deduction of *requirements* on which safety goals need to be supported at which points of tool chains to avoid unacceptable risks. This is (at least partially) based on the information defined in step 1 and one or several reference workflows. These requirements, the points in the tool chains where they apply and mitigating efforts required (for instance qualification means, qualification of tool integration mechanisms, guidelines for processes and requirements for operator training) are documented and included in reference tool chain descriptions. In effect, one is decomposing the *relevant* safety goals at the tool chain level to the *relevant* subgoals at the tool level.
3. *Qualification of the tool chain* identifies the differences between the assumptions of step 2 and the actual use cases, workflow and development environment used when deploying the tool chain.
4. *Qualification at the tool level* is based on the actual use cases, workflow and development environment used when deploying the tool chain and performed according to a relevant safety standard. Three issues could therefore require further efforts at this step:
   (a) An engineering tool has not been qualified by the tool vendor.
   (b) The relevant safety standard differs between tool vendors and tool users and requires additional efforts in regard to engineering tools if the vendor and user are different (an example of such a standard is DO-178C).

(c) The actual use cases, workflow or development environment are different from those assumed during pre-qualification, which means that tools and/or tool integration mechanisms will have to be (re)qualified by the tool user.

These four steps allow qualification of a development environment to be distributed, but one can of course envision the tool user performing all these steps. What is important is that these steps do not have the problems mentioned in the introduction to this section. They both allow (1) a stronger focus on the relevant parts of tool chains in regard to safety and (2) a clear separation of the engineering tool qualification stipulated by safety standards (step 1, 4.a and 4.b are consistent with safety standards such as ISO 26262:2011, IEC 61508:2010 and DO-178C) and extra efforts to ensure safety goals relevant to tool integration. They also have the additional benefits of allowing comparisons between different setups for mitigating safety issues already after step 2 (giving an early indication of the effort required) and favoring early planning in regard to the development environment (helping to avoid fragmentation of the development environment into several *islands of automation* [16]).

5.2 Fully Composable Safety Goals for Automation

Fully composable safety goals for automation can also be qualified according to the steps described in Subsection 5.1. The difference is primarily the implications on step 2. Fully composable safety goals for operator support will indicate what parts of the development environment require qualification efforts, but as long as an operator is still providing oversight these qualification efforts could focus more on reliability. Efforts at the operator level could instead ensure that more complicated risks are handled properly.

Automation, on the other hand, will require additional efforts to transfer expert knowledge from tool chain users to tool chain developers. An example is when ATD is supported to mitigate the lack of training of application designers, but then requires tool chain developers to be trained to understand the domains to a corresponding level of detail. One can argue that this will be more difficult to achieve than simply assuring that the manual handling is correct, however, the automation can be verified more formally through analysis and comparison.

5.3 Emergent Safety Goals

Safety related to the emergent safety goals at the tool chain level cannot be fully ensured only by efforts at the tool level, because there will always remain some uncertainty in regard to how to decompose these safety goals [17]. However, if a safety goal is not fully composable, it may still be possible to isolate some portion of the goal behavior that is partially composable [17]. We have focused on the safety implications of modern, highly integrated development environments, i.e. the way increased automation of tool integration hides what occurs to the operators, rendering efforts (such as processes) at high levels of organization

less effective. Based on this new problem, we below define relevant, partially composable safety goals of the emergent safety goals. Another focus could define more parts of the development environment as important to qualify for DI and DM. Partially composable safety goals therefore have to be defined during step 2 of a qualification effort, based on any considerations specific to the domain of the development effort.

Data Integrity A tool chain shall ensure that development artifacts cannot become corrupted or handled inappropriately without detection in automated, unsupervised process activities. This safety goal can be partially decomposed into the requirement for qualified tool integration mechanisms for verification of version handling and avoidance of data corruption (for instance a qualified version control system that can notify operators of such things as file corruption).

Data Mining A tool chain shall ensure that data regarding safety-related development artifacts that is not handled by operators directly is automatically gathered, analyzed and presented to operators. This safety goal can be partially decomposed into the requirement for qualified tool integration mechanisms for data mining and analysis (for instance a qualified project dashboard).

6 Conclusions

The implications on safety due to tool integration are largely ignored, even though it leads to practical problems for the approaches stipulated by modern safety standards in regard to software qualification. Based on previously defined safety-related characteristics of tool chains we are able to identify nine safety goals for ensuring safety in regard to tool integration. Based on these safety goals, we suggest a systems approach to qualification for dealing with software tools as reusable entities deployed in the context of different tool chains.

This approach allows for a stronger focus on the relevant parts of tool chains in regard to safety, solving the problems that current safety standards have with either stipulating a too wide or a too narrow qualification effort in regard to tool integration. The approach also provides a clear separation of the engineering tool qualification stipulated by current safety standards and extra efforts to ensure safety goals relevant to tool integration, allowing for combining the approach with said standards.

Important issues in need of validation and elaboration in future publications include quantifying the effects of the method on cost efficiency by distributing the software qualification effort, allowing a stronger focus on software which actually has implications for end product safety and a stronger focus on early planning.

Acknowledgments. We thank all participants of the ARTEMIS iFEST project, who have given us continuous access to an additional breadth of expertise on

and experience of software engineering in relation to the life cycle of embedded systems.

8 References

1. Conrad et al., "Qualifying software tools according to ISO 26262," in *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VI*, 2010, pp. 117–128.
2. P. Checkland, *Systems Thinking, Systems Practice.* John Wiley & Sons Ltd., 1985.
3. *BS/IEC 61508:2010, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*, International Electrotechnical Commission Std.
4. *BS/IEC 61511:2003, Functional safety - Safety instrumented systems for the process industry sector*, International Electrotechnical Commission Std.
5. *ISO 26262:2011, Road vehicles - Functional safety*, International Organization for Standardization Std., 2011.
6. *BS/EN 50128:2001, Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems*, CENELEC, European Committee for Electrotechnical Standardization Std., 2001.
7. *DO-178C, Software Considerations in Airborne Systems and Equipment Certification*, Special Committee 205 of RTCA, Inc. Std., 2011.
8. Kornecki et al., "Certification of software for real-time safety-critical systems: state of the art," *Innovations in Systems and Software Engineering*, vol. 5, pp. 149–161, 2009.
9. Gönczy et al., "Tool support for engineering certifiable software," *Electronic Notes in Theoretical Computer Science*, vol. 238, pp. 79–85, 2009.
10. *Certification Specifications for Very Light Rotorcraft, CS-VLR*, European Aviation Safety Agency Std., 2008.
11. Hamann et al. (2011, April) ISO 26262 release just ahead - remaining problems and proposals for solutions, SAE 2011 world congress & exhibition.
12. N. Leveson, *Engineering a Safer World, Systems Thinking Applied to Safety (Draft).* MIT Press, 2011.
13. Ahl et al., *Hierarchy Theory, A Vision, Vocabulary, and Epistemology.* Columbia University Press, 1996.
14. Asplund et al., "Tool integration, from tool to tool chain with ISO 26262," in *SAE 2012 World Congress & Exhibition*, 2012.
15. F. Asplund, "Safety and tool integration, a system-theoretic process analysis," KTH Royal Institute of Technlogy, Tech. Rep., 2012.
16. Asplund et al., "Tool integration beyond Wasserman," in *Advanced Information Systems Engineering Workshops: CAiSE 2011 International Workshops, London, UK, June 20-24, 2011, Proceedings*, 2011, pp. 270–281.
17. J. A. Black, "System safety as an emergent property in composite systems," Ph.D. dissertation, Carnegie Mellon University, Carnegie Institute of Technology, 2009.