DYNAMIC POSITIONING CONFERENCE
October 09 - 10, 2018

Risk SESSION

Properties of evidence and evidence generation

By Odd Ivar Haugen

DNV GL

Abstract

There has been a debate about the capability of Failure Mode and Effect Analysis (FMEA) and FMEA proving trials in providing useful evidence about important aspects of a DP system that are relevant for safety. In 2006, HIL testing was introduced to the maritime industry by Marine Cybernetics (now part of DNV GL). This filled an evidence gap that was not covered by traditional FMEA. Both the FMEA and the HIL approaches generate a body of evidence that is designed to prove safety/reliability claims about a DP system. This paper discusses the properties concerning evidence and the generation of evidence. It is time to develop a framework about which the properties a body of evidence can be assessed against. What is the *capability* of a certain type of evidence? An FMEA is not capable of proving the consequences of interactions between system constituents that are important for safety in a software-intensive complex system. Is the body of evidence *trustworthy*? A question to be asked when planning to increase the level of vessel autonomy is how to prove the safe operation of an autonomous vessel? Stepping back, and asking questions about evidence and the process of evidence generation, we may be able to develop a more optimal assurance methodology than at present when technology challenges current development, integration and verification practice. Most of the aspects outlined in this paper should be familiar to the verification practitioner, but collecting the aspects and labelling them may help us to be more precise in our discussions, and thus decrease the possibility of misunderstandings.

A few aspects of verification and assurance

Verification is defined as: "*The process of providing **objective** evidence ...*" (Institute of Electrical and Electronics Engineers, Inc, 2016).

Moreover, (International Organization for Standardization, 2013), describes the term *assurance* as being "*...ground for justified confidence...*".

Within an assurance framework, systems such as the DP system are examined (reviewed, analysed, inspected, tested) by different parties to **generate evidence** in order to build confidence that the system is safe and complies to the class rules and other requirements.

Evidence must possess some particular properties to be useful in the assurance process to provide such confidence. Although there may be several definitions of verification, in the above definition, *objectivity* is one central aspect or property. However, there are also other properties that should be considered when evidence is used in creating grounds for justified confidence in a system. These properties are examined and discussed in this paper. But first, we must discuss different aspects of the verification process to understand how these aspects may influence evidence properties.

Roles: Verification Organization vs Assessor vs Test Witness

There may be some confusion about a stakeholder's role and position within the verification process. Moreover, a certain organization may have several roles within the same project, depending on the system life-cycle phase. A role is defined by the position of a stakeholder within the current life-cycle phase, such as a stakeholder's function (actions/tasks), responsibility, and intension. In the following we discuss three roles: Verification Organization (VO), Assessor, and Test Witness.

What should be attributed to a stakeholder in order to hold the position as a Verification Organization (VO)? First, some have argued that witnessing a test activity, qualifies the witness's organization to be a VO. This is not our understanding. We state that a VO must conduct its own analysis of the target system; must *specify* and *create* its own testing products; must *perform* the verification activity; and must follow up possible findings to be qualified as VO. Testing products may be test plans, test design specification, test case implementation, and test procedures. Moreover, the tools used in the verification should preferably be partly or fully independent of the tools used by the development organization (i.e. vendor). Using shared tools both in the development and in the verification of a system may mask biases that may hide defects. A position as the VO is not only determined by its actions, but also by its intensions and motivations. The intrinsic intension of the VO should be to identify weaknesses and possible defects in the system under test, without being "held back" by loyalty or other attachments towards the developer.

Of course, the VO must have in-depth technical and operational knowledge about the system to be capable of generating evidence of adequate quality from the activity (more about this later).

The organization must *generate primary evidence* to qualify as a VO. An organization assessing the properties of such primary evidence or verifying the process which is used in the development and integration of the system is termed an assessor (i.e. generating *circumstantial evidence* about the target system). A test witness, witnesses test activities to assess the process of generating primary evidence, and other properties of the generated evidence, such as correctness or validity. The test witness potentially raises the trustworthiness of the evidence generated by the VO, and therefore contributes by generating confidence in the system, but the evidence is still generated by the VO, and not by the test witness. **A VO generates primary evidence; an assessor/test witness does not**. When talking about the level of independence in the verification effort, we must firstly talk about the level of detachment of the VO from the developer/integrator of the system, not the assessor/test witness, although their attachment is relevant to the overall trustworthiness of the whole verification effort and to the generated evidence.

Early in a new-build project of a DP vessel, the classification society performs plan approval based on a review of the design documentation. The class society generates primary evidence based on a certain type of artefact representing the real system; the class society therefore has the role of VO. Later, the class society witnesses the Factory Acceptance Test (FAT) based upon a class-approved test programme made by the vendor of the equipment; here the class society is both a test witness, and an assessor. The vendor/developer generates *primary* evidence and therefore has the role of VO. Later in the project, in the FMEA proving trials, the FMEA company has analysed and created a proving trial programme, and is in charge of conducting the trials, making the FMEA company the VO. Although the class society is a test witness and an assessor also in this activity, it will follow up findings related to a breach of class rules, which means that class performs part of the VO's responsibility.

As seen from the previous paragraph, a stakeholder changes role during the project, it may have several roles at once, and a stakeholder may perform parts of the functions of another role. The changing roles for stakeholders mean that the level of VO detachment from the developer also changes.

Intensity, rigour and detachment in the verification effort

The safety/mission criticality level differs between systems, and the different operations may also be more or less safety critical. The rigour and intensity of the verification effort should reflect these aspects. A first approach may be to just state that the level of verification should increase with criticality. But what do we actually mean by expressions such as "increased verification"?

There are several dimensions in statements like the above, three of these are rigour, intensity, and detachment. In this paper, detachment reflects the level of independence of the VO from the developer/supplier/system integrator. In some safety standards, the independence level required in the standard reflects the independence of the *assessor*, not the VO. It is very important to be aware of this difference.

Intensity

Increased *intensity* means that the verification includes a greater scope across normal and abnormal system conditions. For a safety-critical system like the DP system, it is inadequate to only perform *positive* testing, which means that only the normal system condition is tested. This is not the case in today's practice. Abnormal system conditions are in focus both from a class rule perspective and from an FMEA perspective. However, the number, the relevance, and type of abnormal conditions tested may vary from project to project. It is important when testing abnormal system conditions, to include and vary a broad number of system variables and parameters in the test cases. In other words, it may be equally interesting to keep the equipment failure mode fixed while altering other seemingly unrelated system

variables and parameters. Performing this kind of testing systematically is called *combinatorial testing* in software testing terminology (more on this in the section about rigour).

Intensity may also encompass more system artefacts being verified. This includes additional types of documents, source code, and perhaps digital models of the asset or system. Including additional documents into the verification effort, quickly leads to the need of shifting the verification effort towards earlier phases in the system life-cycle. Reviewing a document like the Concept of Operation (CONOPS) after the asset has been built is not very effective. Moreover, the number of artefacts also require an increased number of test methods, such as white-box and black-box testing, and development process verification.

Intensity may therefore also mean that a number of system life-cycles are included into the verification effort even before the (current) system is built. Examples are type approval and verification of the process which the vendor follows during development.

Rigour

Any verification activity may be performed with different levels of formalism with respect to techniques and documentation; these are different levels of *rigour*.

Class rules and other standards stating that the system should be tested under abnormal system conditions, such as equipment failure indicate the *intensity* in the testing. The level of rigour must therefore be decided within the verification effort in each project.

While the number of equipment fault modes tested relates to *intensity*, the testing techniques applied when testing these fault modes relate to *rigour*. As mentioned earlier, an equipment fault mode may be tested under different system conditions, or parameters. On the surface, these conditions may seem unrelated to the possible system effect of the fault mode, meaning that there is no apparent reason to believe that the test result will differ when altering the system conditions. However, most of us have experienced that a fault ended up in a system failure because some other seemingly unrelated factor appeared at the same time. Increasing the rigour means that the system is analysed so that these conditions can be selected and systematically tested. Systematically utilizing established testing techniques and processes and formal documentation of them, also adds to the level of rigour. In the example above, a software test technique called *combinatorial testing* may be used. Creating one test case for each combination of all the included system conditions will result in a large number of test cases. Instead, one may choose to only test all pairs of conditions which decreases the number of test cases considerably. The thinking behind this approach is that most failures are the result of two conditions occurring at the same time. Therefore, there is no need to test all combinations, only make certain to test all pairs of conditions. Of course, there is a risk that a certain failure may only appear as a result of a combination of three or more conditions, which may be missed in all-pair testing. Analysis, justification, documentation and implementation of such testing techniques increases the level of rigour in the verification effort.

Detachment

As stated earlier, in some of the life-cycle phases (e.g. plan approval) and upon some of the system artefacts or documents, the class society has the role of VO, and generates primary safety evidence about the system. The VO is detached from the developer so that the verification effort is independent. At FAT however, the vendor possesses the VO role (see Roles: Verification Organization vs Assessor vs Test Witness), resulting in a close attachment and dependency between the VO and the developer, which is a non-independent verification effort. Of course, vendors must, and always will, verify their own system and equipment as part of their development process. Verification is a crucial part of every development process.

Integrating and verifying systems and equipment supplied by different vendors is the responsibility of the system integrator (i.e. yard). The FMEA and proving trials are often performed by an FMEA company,

but may also be performed by the yard. The yard can be termed as the "developer" of the integrated system, again resulting in a non-independent verification effort in case the yard performs the FMEA and proving trials. To the extent that the verification effort of the integrated system relies upon equipment evidence generated from sub-systems, they are provided by the vendor of the sub-system/equipment, or by the class society as part of their type approval, or plan approval. This situation creates a complicated picture of the true detachment in the verification effort.

Increasing the "distance" (mental, organizational, financial, procedural, and technological) between the VO and the developer is seen as desirable to minimize the possibility of cognitive and societal biases that may mask defects. According to (International Organization for Standardization, 2013B), increasing this distance, increases the objectivity. Objectivity is instrumental in creating a trustworthy body of evidence, and will be discussed in more detail later.

Another way of depicting independence and detachment is to borrow two terms from the social and behavioural sciences: *Emic and Etic* (not ethic). *Emic* is an investigation and explanation from the viewpoint of the people within a culture using their own perspective and concepts; *an insider's viewpoint*. In the case of the verification effort, an *Emic verification* would be performed by people within the development organization, culture and belief. *Etic* is an investigation of the behaviour of people or system from an *outsider's viewpoint* using universal reference points. An *Etic verification* approach realizes that developers are too involved to be impartial in their process of generating objective evidence about the system they have developed.

It is realized and widely accepted in the social and behavioural sciences that both approaches generate valuable information and insight into a social structures and functions. The same can be said about the verification effort, the two viewpoints are complimentary, and both are needed to build the required level of confidence in a safety critical system such as the DP system.

One aspect *of objectivity* may be said to be "value-neutralness". As no VO can be said to be "value-free", more than one VO may contribute to a value-totality that can be said to be "value-neutral". Moreover, even though sometimes a complimentary approach at first seems to be overlapping, meaning that the scope of emic and etic verification seems redundant, we should not dismiss such redundancy immediately, because the "redundancy" in scope may, if generated adequately independently, increase the objectivity of the body of evidence, and thereby increase the level of confidence in the system. While in the context of certification of DP systems, the *Etic* verification limits its scope to safety, while an *Emic verification* effort will include all other quality characteristic such as usability, maintainability, availability, learnability and other "-ilities". One may also ask the question whether stakeholders should request an *Etic verification* of other (important) "-ilities" as well, if they are deemed important enough.

Different kinds of verification (in context of systems and software)

The verification effort may be divided into two categories: *Product* verification and (Development & Integration) *Process* verification. Both types of verification generate evidence that ultimately indicates the degree of confidence stakeholders can place in a system.

There has been a debate between verification practitioners about which type of verification is more effective in generating confidence when performed in an *etic* verification context. We believe that both types can, and must be performed both in an emic and etic verification context in order to generate adequate confidence in safety-critical control systems such as the DP system.

Product verification: Generating primary evidence

Evidence generated through product verification is termed *primary* evidence. The body of evidence is directly linked to the behaviour of the system or equipment. The test artefact, or the test target, is the *real* system; the *actual* system or some representation of it, such as design documents, source code, or perhaps

simulated models, i.e. a *virtual* system. All of which are more or less accurate representations of the real system.

Verification in the context of testing, may be categorized into two types, static testing and dynamic testing. Static being activities such as a review of documents. Also, static source code analysers perform a type of static testing. Dynamic testing includes activities such as FAT, sea trials and simulator-based testing. In addition, such testing is typically conducted in the development process such as *unit testing*.

Furthermore, testing may also be divided into white (or glass)-box testing and black-box testing. As the names indicate, the internals of the system (or software) are known to the VO when performing white-box testing, and are hidden when performing black-box testing. Within an etic verification context, if the (independent) VO does not get access to the internals of the test target (e.g. due to IPR concerns), black-box testing is the only option. Sometimes, black-box testing is termed *specification-based* testing, or *functional* testing, while white-box testing is termed *structural* testing.

The vendor will perform both types of dynamic testing as part of their development process, while etic verification is often restricted to black-box testing.

Within both types of testing, practitioners have developed distinct techniques and coverage measurements, which influences the intensity and rigour of how the activity is performed, and thereby the level of objectivity of the generated evidence.

(Development & Integration) Process verification: Generating circumstantial evidence

Evidence generated through process verification may be termed circumstantial evidence. Such evidence requires *inference* in order to draw a conclusion about facts, such as the facts concerning the safety of a system.

Circumstantial evidence from process verification may be generated through reviewing documents describing the development procedures, or it may be artefacts showing details of how the development is conducted, i.e. proof of conformance towards stated procedures (vendor's own description and/or standards etc.). These activities are conducted throughout the life-cycle of the system.

As mentioned, there has been a debate between verification practitioners about the degree to which such evidence is capable of providing proof concerning a conclusion about the product, such as the DP system. In general, *inference* is needed to draw a conclusion from circumstantial evidence, so the question becomes a question about the strength of the inference.

No doubt, a systematic, explicitly stated, and well understood development process increases the probability of the outcome of that process having adequate quality; however, such body of evidence is generally insufficient. In the same way as we cannot assume that the product will satisfy the requirements just by verifying that the developer follows a set of procedures, there is no way we can "test" quality into a product. Quality is built into the product from the earliest phases of the specification and implementation, and throughout its entire life-cycle.

A process verification must be accompanied by product verification (i.e. generating primary evidence) to create adequate level of confidence in the product or system. However the importance of the recipe - "*The proof of the pudding is in the eating*".

As with all verification efforts, the process verification may be more or less intense and rigorous within an emic or etic verification regime.

The position of algorithms in the verification effort: Algorithm-based Verification Agents

It is anticipated that new technology can be used to develop verification and test functionality which can be incorporated as integral parts of the DP system, such as a Condition Monitoring system (CM). Such verification functionality may be based on advanced algorithms. This challenges the traditional

verification scheme by the introduction of a new role; an *Algorithm-based Verification* A*gent (AVA)* (Haugen, 2018). An AVA is an algorithm with a dedicated role, purpose, function, and responsibility (i.e. an agent) in conducting verification (i.e. generating primary evidence) by interacting with the target system. In a wider context, an AVA may not interact directly with the target system, but for the purpose of this paper, we assume that it does.

A test witness's and assessor's responsibility is to assure the quality of the process and outcome (evidence) from the verification effort and the development process. If the product verification is conducted by an algorithm, there may be no witnesses to the verification "activity", and details about the inference used in generating the outcome may either be deemed a secret by the vendor, or may not be readable by humans at all. It may only become visible as a result of the inference, such as a "green light". If we are not vigilant, the content of the assessor role may become degraded to just "standing" at the end of the verification "assembly line", looking at "green and red lights" produced by the AVA, and then give a "thumbs-up" if it is green, and "thumbs-down" if it is red. In this situation, the value of the assessor role is close to zero.

In the light of the above, it is obvious that such an agent must be assured in order to ensure that the generated evidence is genuine, trustworthy, and possesses all other evidence properties to an adequate degree. Close attention must be paid towards the (integrated) automated and data-driven verification functionality.

This verification functionality may go under different names, such as “Built-In-Test-Equipment” (BITE), or Self-verifying systems. They are perhaps based upon Machine Learning (ML). Even some parts of the (system) assessment itself may become autonomous through the use of algorithms ("green boxes"), increasing the importance of the rigour and intensity by which these algorithms must be verified. They are, in their own right, complex software-intensive systems, posing the same challenges in the verification as any other complex system, such as lack of transparency (black-box), their intractability, difficulties of establishing expected test results, and the possibility of built-in cognitive and societal biases that decrease the level of objectiveness in the generated evidence.

System complexity affecting test artefacts and verification methodologies

Systems and software consist of many possible verification artefacts, i.e. test items, produced in different phases of the development process. Test items can be documents, like design specifications, produced early in the development process, or operator manuals, often produced just prior to release. The source code is of course another category of test item.

Static testing of requirements and design documents is very important in discovering flaws as early as possible, decreasing the cost of re-work. Static testing increases the confidence in the correctness and completeness of the documents. A fresh viewpoint, in the context of etic verification, increases the probability of discovering omissions, which is a known source of unsafe software behaviour (Ericson II, 2013), (Leveson, 2011). Omissions are especially difficult to discover for a person who is too close (emic VO) to the software development process.

Ideally, all defects are identified in the development phase when they are introduced. Unfortunately, this is often not the case. The verification effort is not perfect, and can never find all defects, and some defects will always find their way to the next phase of the development process and into the next produced item (artefact). As stated earlier, a particularly difficult type of error to discover early in the development process is omissions - what is missing?

The items produced along the development process transform their physical property (from paper to code) and functional properties. The level of complexity will also increase. The test process (method, tools, documentation) of each test item must adequately reflect its physical and functional properties, complexity, safety level, and risk factors. The test method must change as the test items transform.

Although it is important, finding all defects by reviewing the system documentation is impossible. Early in the development/integration process, the system only exists as mental models on a "piece of paper" that must be interpreted by humans, and projected onto the finished system. These cognitive processes will inevitably lead to misunderstandings, and defects such as omissions.

Omissions may be explained by the lack of *tacit* knowledge, knowledge presumed by the author(s) of the requirements or specifications, and therefore not made *explicit*. Omission in the requirements leading to coding defects can then be explained as a result of a communication gap between the author(s) of the document, and the developer of the source code. The question will then be whether we can close the gap by increasing the level of detail (and the length of the documents). Or, perhaps a document, like the software requirement document, can never describe a complex system in such a way that it can be coded without flaws. Will the software tester be able to envisage the requirements correctly? It might be that the requirements must be transformed to something more tangible, like running code and digital models, to make the requirements (and defects) more observable, also for the tester.

Of course, requirements are also transformed to records in a database. This is still not enough, and it can even make the communication gap wider. After all, prose suits humans relatively well in the visualization of thoughts, ideas, and systems. How well would we visualize the content of "The Lord of the Rings" by reading the book as standalone records in a database, like bullet points? We should not underestimate the power of good requirement prose.

When the system (or test items) is transformed from paper to lines of code, further to running programs, and finally to the complete integrated system, it becomes more and more concrete and complex. Increased concreteness enables a better understanding of the system properties and therefore provides a better chance of discovering defects, not only defects introduced in the current development phase, but also missed defects from earlier phases.

Safety vs. Reliability in complex systems (e.g. DP system) and what it means for the validity of the evidence

Just as with the debate about product vs. process verification, there is also a debate around safety vs. reliability in complex systems. Safety and reliability have historically been mixed together (Leveson, 2011), as in: reliability is both necessary and sufficient to ensure safety. Surely, the mix between these two properties may have been more nuanced, nevertheless, a discussion about the difference between the two properties in the context of complex software-intensive systems, and safety evidence is in place because it will affect verification methodologies and the capability of the generated types of evidence.

First of all, we can consider a few definitions of reliability:

- Reliability definition no. 1: "...degree to which a system, product or component performs **specified** functions under specified conditions for a specific period of time..."
- Reliability definition no. 2: "...the ability of a system or component to perform its **required** functions under stated conditions for a specified period of time..."
- Reliability definition no. 3: "...the probability that a device will perform its **intended** function, during a specified period of time..."

The detailed reference is not important because there are many definition of reliability with slightly different wording. The important words, in the context of this paper, in the three above definitions are: *Required*, *Specified*, and *Intended*.

These (small) nuances may be the origin of the debate: If, after the fact, a system is revealed to be unsafe, does it necessary follow that the system also was unreliable?

If reliability is defined as in the first item in the above list (*specified*), the system is still reliable. In other words, the specification made the system perform unsafe actions: *a safe system may or may not be reliable*, Figure 1.

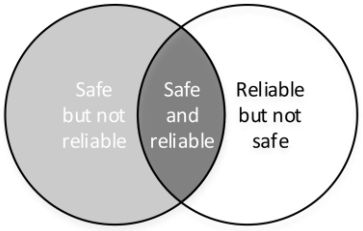

Figure 1 Safety and reliability in complex software-intensive systems

If reliability is defined as in the third item in the above list (*intended*), the system may always be said to be unreliable because it was never *intended* that the system should be unsafe. This definition opens the way for hindsight, and*: safety = reliability*.

If reliability is defined as in the second item in the above list, either of the above statements can be said to be true, depending on what is meant by *required*.

We believe that using the word *specified* is a better choice, not opening for hindsight in our definitions. This means however, that safety may be something different from reliability, and ensuring a reliable system does not necessary mean that the system is safe: As Figure 1 illustrates, an unreliable system may be safe (light grey area), and a reliable system may be unsafe (white area). Of course, there is nothing that hinders a system being both reliable and safe (dark grey area) in principle, but **safety cannot be concluded through a body of evidence showing adequate reliability**.

Following the "reliability"-path to safety may be adequate for single components where cause and effect are well understood. An example is predicting the behaviour of a "simple" valve. There are no, or very few surprises, partly because the system is simple, and because a valve is a well-known and understood concept. The causes and effects are predictable through a priori analysis, and the specifications may be said to be complete, or sufficiently comprehensive to make a valid link between safety and reliability.

Software, however, enables system complexity, and it is sometime said that through the use of software we design complex *machines*. We can make a (SW) *machine* as complex as we want, not limited by (physical) production, or any physical law. It is easy to make a *machine* that is not intellectually manageable. Adding to this picture, changing a few lines of code, may change the very nature of the *machine.* It is therefore easy to grasp that we are dealing with something fundamentally different from a simple valve, and the methodologies used in the verification effort must change accordingly.

In the safety verification effort, i.e. generating evidence concerning the safety of complex software-intensive systems such as the DP system, methodologies must target safety as a unique property. Of course, that does not make reliability (or availability for that matter) less important for the stakeholders, only that it represents a different focus.

The challenges lie in the system complexity and emergent properties (see Appendix: Complexity and emergent properties). As system complexity increases, its behaviour becomes more and more un-predictable and difficult to analyse (even impossible according to some researchers). The specifications may therefore never become complete, and ensuring that the system complies to the incomplete specifications, i.e. the reliability, is inadequate for safety.

Evidence properties

As seen from the discussion so far, the type of evidence in the present instance both as a piece of evidence and as a collective body of the evidence, must possess some particular properties and qualities that enable the stakeholders to reach to a *justified level of confidence* that the system is adequately safe in operation.

Immediately, there are some properties that come to mind, such as relevance, correctness, completeness and objectiveness. Objectiveness is at the very core of verification. Three such properties regarding type and instance are according to one source (Hawkins & Kelly, 2014): type capability, instance capability, and instance trustworthiness. We suggest expanding and modifying this list to also encompass the collective body of evidence and to rephrase one of them, and add others:

- evidence **type capability**
- evidence **instance quality**
- evidence **instance trustworthiness**
- evidence **instance validity**
- **body** of evidence **type completeness**
- **body** of evidence **instance coverage (and depth)**

Although the above list might (still) be incomplete, and for sure, the research and discussions about evidence does not end here, we will argue that the list represents some core properties which the evidence can be assessed against. Another list of aspects concerning evidence can be found in (Menon, Hawkins, & McDermid). Depending upon the meaning attributed to each item, one could perhaps merge two or several items, or expand the list with more items. However, we believe that the list gives an appropriate framework for evidence assessment.

In the following sections we will give a brief description of each property, followed by a discussion of causes of deficiency and interaction between them.

Evidence type capability

Verification is conducted in all system life-cycle phases using different methods for different artefacts, all of which are capable to show different aspects about the system under test. Stakeholders must analyse the target system, operational conditions, its operational environment, and rules and regulations to decide on the capability of the required evidence to generate *justified grounds for confidence.* In the context of maritime classification, the activities, methods and artefacts are to large extent predetermined through the class rules and other class-related requirements. Some examples of evidence type capability are given below:

Starting with a trivial example:
Design document review (plan approval) will identify design defects early so that they can be corrected before the vessel is built at a lowest possible cost; however, this cannot reveal implementation errors. Moreover, the capability of the review determining emergent system behaviour and system robustness against equipment errors is limited. We need other verification activities and methods to cover these aspects such as FAT, sea trials and simulator-based testing.

A more controversial example:
FMEA was originally intended as a reliability analysis method, and claims have been made (Leveson, 2011) that it is not capable to generate safety evidence, only evidence about reliability. Moreover, we have argued that safety is not a "sub-property" of reliability in complex software-intensive systems such as the DP system, meaning that FMEA to lesser degree is adequate in such a context for safety analysis. It fails to catch the very nature of an emergent property (interaction between system constituents), such as safety. However, as in any aspect of life, few things are “one or the other”, as in “perfect or useless”, life is always lived somewhere in-between extremes, so FMEA is neither completely useless nor perfect in

generating safety evidence. We must re-analyse the *capability* of the evidence type generated by FMEA and complement it in areas where it is found to be inadequate.

A few more examples:
Evidence from development process verification (circumstantial evidence) cannot prove system properties adequacy - product verification is also needed (primary evidence).

A well-known statement from software testing: "*No testing can prove the absence of defects, only the presence*". Only formal (mathematical) verification possesses this capability; however, it can only be used on extremely simple test artefacts, and definitely not on a complete DP system.

Evidence instance quality

Even if the evidence type is capable of showing particular information about the system, the actual instance may be of such quality that it is less useful in creating confidence; the evidence instance did not fulfil its evidence type capability or potential.

The test cases in a test programme may lack the necessary descriptions to make the test result *repeatable*. The document review may not follow a prescribed procedure, resulting in some aspects not being assessed. Such deficiencies may result in the evidence becoming *incorrect*. This may be caused by an organizational deficiency, or the lack of professional competence in the field of testing.

As mentioned earlier, for the evidence to be of adequate quality, the VO must have in-depth technical and operation knowledge about the system and its working environment. Moreover, the VO must be competent in the field of verification and testing. For software testing, the latter means that the VO must master testing techniques, e.g. (International Organization for Standardization, 2013B); must master the test process; must master test documentation; must master the test environment. A VO that fails to master any of the previous items may jeopardize all evidence instance properties, including its quality.

Evidence instance trustworthiness

Objectivity in the processes serves the function of promoting trust in the outcome of that process - the products. In the case of a verification process, the product is a body of evidence that is supposed to support a safety claim/expectation about the system.

Objectivity is not a rigidly defined property, a description could be: "*a set of norms that obliges persons or group of persons to apply impersonal modes of reason in the course of their inquiries or deliberations*" (Axtell, 2016). This is a property of a process, not a product. Let us assume that an objective process leads to objective outcomes, or products: "*To call the result (product) of inquiry objective is on a social level to endorse those products as trustworthy due to characteristics of the process by which they were produced.*" (Axtell, 2016). This must not be confused with the quality of the outcome. As earlier mentioned, ensuring that the process of developing a product maintains a standard (process verification), does not ensure adequate quality in the outcome of that process. To ensure adequate quality of the product, we must perform product verification (generating primary evidence).

From the above discussion, we can conclude that an objective process of generating evidence results in the evidence being trustworthy. The key is therefore that the process by which the verification effort produces evidence, must be objective.

Objectivity is a normative concept and places requirements on our thinking. It requires us to distinguish facts from opinions. The "thinking" (reasoning and decisions) of the VO must be objective, in order to create trustworthy evidence.

Objectivity is unfortunately not the only way of creating trust. Authority can do the same, like in: "*Trust me, I'm a doctor...*". Creating trust through authority is like saying: "*You (should) trust me, therefore you should accept my proposition without evidence.*" Authority also influences our critical thinking in that we

have an inherited *authority bias* (Authority bias, u.d.): "*Since the opposite is anarchy, we are all trained from birth to believe that obedience to authority is right*". Whether we yield to a proposition out of trust, or as a result of authority bias is hard to say, the effect remains: lack of demand for scrutinizing the body of evidence.

Trust may also and quite easily be created by constructing a coherent story. This story may not be objective, or even true, the only criterion for its ability to create trust is that it is coherent. "*Subjective confidence in a judgement is not a reasoned evaluation of the probability that this judgement is correct. Confidence is a feeling, which reflects the coherence of the information and the cognitive ease of processing it*" (Kahneman, 2011).

Objectivity in the verification process is a complex matter, and an in-depth discussion is not within the scope of this paper. However, two aspects influencing objectivity in the verification effort are described in this paper: level of detachment of the VO, see Detachment, and the level of rigour in the verification effort, see Rigour.

Evidence instance validity

There are a number of incidents in the verification effort that may invalidate the evidence. The test activity may not be conducted with the correct version of the software or hardware. The data on which the evidence is based upon may have been tampered with, perhaps in the context of an AVA, see The position of algorithms in the verification effort: Algorithm-based Verification Agents.

The process of following up findings from a test activity may be careless, increasing the probability of introducing safety-related side-effects, or uncertainty about what has been corrected and what remains to be corrected.

In general, evidence instance validity may be jeopardized by the VO's professional incompetence, or deficiencies in the verification tools and processes. The generated evidence may become *irrelevant*.

The proven-in-use is sometimes used as evidence to prove the certain quantitative reliability of the DP system. This practice is seen by some to be problematic because every (even small) alteration in the SW code invalidates the proven-in-use principle. Moreover, the working environment and properties of the operation must also be taken into consideration for proven-in-use to be a valid piece of evidence. Of course, as seen from previous discussion, reliability and safety should be treated as two different properties in the context of complex software-intensive systems.

Body of evidence type completeness (compared to the required scope) and the body of evidence instance coverage (and depth)

The verification effort must identify the needed verification intensity, and dedicate evidence types, with adequate capability to the different parts of the scope. In total, the complete (theoretical) instance of body of evidence should have the capability of filling the required scope. In other words, the body of evidence must adequately address the properties e.g. *safety* in a classification framework of interest of the system under test, with adequate intensity and rigour. In the case of safety verification, the required scope should be developed through a risk-based approach.

Two types of evidence may address the same parts of the scope, apparently making them redundant. Redundancy in evidence type (or instance) capability should not automatically be dismissed. If two types of evidence are generated by using independent methods targeting the same part of the scope, the evidence will act as reinforcement of both and thereby increase the confidence in that particular part of the scope. Moreover, even if the methods depend on each other, or even are identical but are performed by two different VOs, the confidence in that part of the scope may increase, especially if the targeted part of the scope is less tangible and important to the stakeholders.

The evidence *instance* is not completely determined by the evidence *type*, e.g. FMEA performed by two different companies may differ considerably. A FMEA test programme may also be different in intensity.

A particular test case, targeting a particular tangible part of the scope, may be deemed redundant even if performed by two different VOs. This is the case if there is strong (and simple) inference between the evidence and the particular part of the scope.

An example of this is the test case to show that UPSs can uphold the connected equipment for at least 30 minutes after loss of UPS input power. This part of the scope can be said to be tangible, and (fairly) straightforward to execute and record, so there is no need for redundancy in the evidence instance. On the other hand, showing that the DP can hold position after a thruster failure is not so trivial to prove and is less tangible, even with an evidence type created through simulator-based testing. Two different VOs may challenge the system differently, and the simulator (i.e. the tool, or test environment) capability may also differ, decreasing the actual redundancy, even if the evidence type is identical.

A particular body of evidence instance coverage of the potential body of evidence type capability may differ by performing the verification with different levels of *intensity* and *rigour*. Again, in the case of showing the DP's capability of station-keeping after a thruster failure, this is sometimes done using a very limited number of test cases: “For each thruster, disconnect (or shut-down) the thruster, and observe that the DP keeps the position.” This limits the evidence instance coverage to one thruster fault mode (loss); to one sea state; to one thruster allocation mode; to one operational condition (stationary); to one weather condition (current), and so on. Another instance of this evidence could vary the fault mode; vary the thruster allocation mode; vary the operational conditions, and so on, increasing the coverage (and intensity) considerably. Of course, the test environment may place limitations upon the possibility of variations in the testing: during a sea trial, the weather conditions may not vary very much, and of course it is not controllable.

These variations may result in executing different parts of the software code, increasing the intensity of the verification. The number of test cases may become large if all possible combinations of the above factors are to be tested. However, software testing techniques may be utilized to ensure an adequate coverage, while at the same time limit the number of test cases.

Reviewing documents such as requirements and specifications, produces the same type of evidence, but both the number of reviewed documents, and the intensity and rigour of which each document is reviewed may vary, which will affect the evidence instance coverage.

Causes and property inter-influence leading to the body of evidence failing in creating justified grounds of confidence

Failing to create *ground for justified confidence*, may be caused by flaws in the body of evidence. In order to mitigate the occurrence of assurance deficiency, we should analyse the possible causes of flaws in the body of evidence. A "HAZOP"-like approach may be used in identifying these causes (Hawkins & Kelly, 2014).

Whether creating primary evidence (VO), or circumstantial evidence (assessor), lack of organizational or professional competence about the target system, the operational aspects, or its operational environment, may compromise most of the evidence properties.

The VO's proficiency in *analysis techniques* is sometimes (and with good reason) emphasized; however, another crucial competence requirement is the VO's (or assessor's) competence in the field of testing and verification - as a profession. Evidence *instance quality*, *validity*, needed *type completeness*, and *scope coverage* may be jeopardized if the VO lacks the necessary testing and verification competence.

Lack of VO detachment from the developer compromises the *evidence instance trustworthiness*. However, lack of transparency or documentation into/about the target system/equipment, may restrict an independent VO in generating evidence instance with adequate *intensity*. Moreover, it may create

difficulties in examining the body of evidence instance *scope coverage*. Although any kind of black-box testing should be possible to perform by an independent VO, there might become an increasing need for transparency into the structure and learning data of machine learned algorithms, because of the vast input/output space. The inability of decomposing requirements into specific lines of SW code emphasizes the need for a tighter coupling between white-box and black-box testing in order for the black-box testing to become adequately efficient. The verification practitioner must identify "areas" of interest through white-box testing, and use that as a starting point for black-box testing. This will also increase the VO's ability to justify the body of evidence instance scope coverage.

Inadequate verification tools and processes can also jeopardize the evidence properties as mentioned earlier. Even if the tools used are adequate, if the VO for different reasons is not capable of utilizing them, it may compromise properties such as *quality*, and *validity*.

Although the listed taxonomy of evidence properties can be said to depict distinct viewpoints, they are in some way also interrelated, or may influence each other. We may think of the ability of the evidence fulfilling its potential, and the ability to create confidence as an *emergent property* of its individual constituents. Figure 2 depicts such a model. At the bottom level resides aspects of the verification effort: Roles, Rigour, Detachment, and Intensity. Some of these aspects are influenced by the intention, motivation, responsibility, and knowledge within the verification effort. Moreover, they will influence each individual evidence property (validity, quality, and trustworthiness), and the entire body of evidence coverage. At the “top” of the hierarchy is the level of confidence that can be placed upon the system. The hierarchy (model) is relevant for both for primary and circumstantial evidence (left).

This is just one simple mental model of how one may think about such a framework. We are certain that other models can be equally well used to depict and analyse these relationships (e.g. FRAM (Hollnagel, 2012)). Following approximately the words of the famous British-American statistician Georg Box: “All models are wrong; some are useful”, we are certain that aspects can be shown to be missing or perhaps even completely wrong, or should be depicted in another way. However, a more relevant question is: How useful is the model to get an overview of such a framework?

As mentioned, verification aspects at the bottom level have some influence upon the resulting evidence properties at the second level. The aspects may also influence each other within the bottom level, e.g. the level of detachment may influence the intensity of which the verification is performed because of cognitive or societal biases. Moreover, several aspects at the bottom level will influence one or more evidence properties on the second level. Finally, at the top, confidence is influenced also by other matters than evidence, such as traditions, supplier maturity, and stakeholder authority. The nature and degree of influence must be analysed.

The above discussion indicates that the evidence properties are emergent, with both upwards and sideways causality.

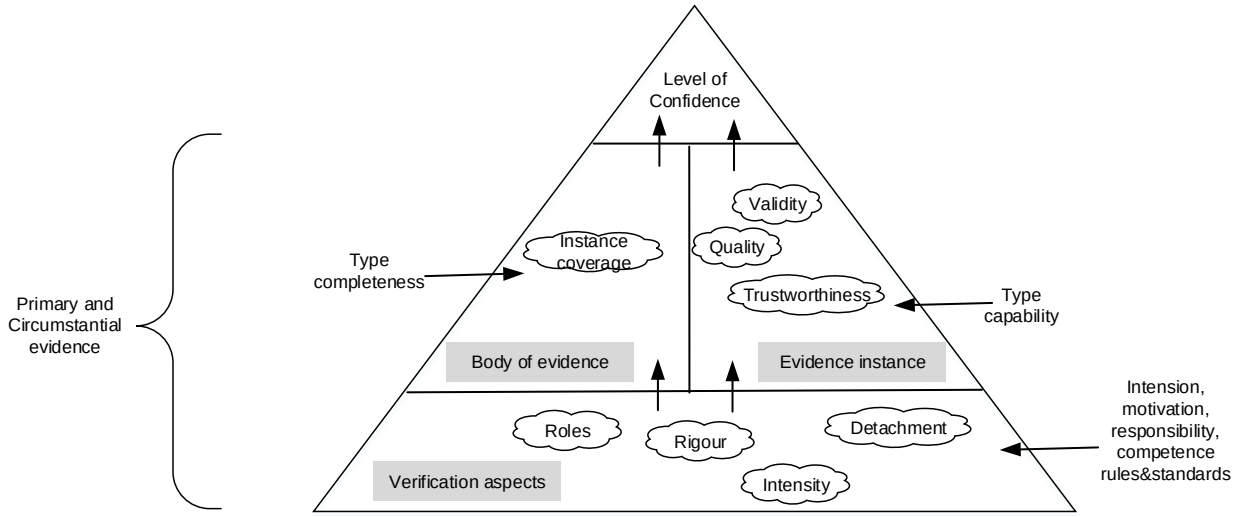

Figure 2 Evidence properties and their inter-influence

A few simple examples:

> By failing to generate valid evidence, evidence trustworthiness is also compromised. However, trustworthiness is not a sub-property of validity because trustworthiness, as mentioned above, is also compromised when there is a lack of detachment between the VO and the system developer.
>
> The *scope coverage* refers to the body of evidence *type completeness.* If the *type completeness* is inadequate in creating justified grounds for confidence, even fulfilling the body of evidence scope coverage 100%, will still not be adequate.

The complexity in the previous analysis lies in the constituents (the separate properties and aspects) interacting and influencing one another, and that the “totality” is not a “sum” but rather emergent, not a simple sum-like "total score". In other words, the ability of the body of evidence to create *justified grounds for confidence* cannot be analysed by reducing the properties into discrete units where they are analysed in isolation, all properties must be analysed in the context of a totality.

Conclusion

As technology enables more and more complexity to be implemented in maritime control systems, such as the DP system, we need to take a step back and re-evaluate the evidence and methodology by which evidence are generated. This is required so that we can maintain a necessary level of confidence in the systems; evidence-based grounds for justified confidence or trust. This paper has discussed different evidence properties and aspects of the generation process that may influence these properties. In order to be able to assess their influence on the trust we can put into the systems, it is important to have a framework like this which can be used to scrutinize new types of evidence and their instances. In addition, we need new methods concerning how they are generated, and possibly the new roles entering the verification effort (e.g. the Algorithm-based Verification Agent, AVA).

References

Authority bias. (n.d.). Retrieved August 8, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Authority_bias
Axtell, G. (2016). *Objectivity (Key concepts in philosophy).* Polity Press.
Bedau, M. (1997). Weak Emergence. *Philosophical perspectives: Mind, Causation, and World*, 375-399.
Ericson II, C. A. (2013). *Software Safety Primer.*

Haugen, O. I. (2018). The importance of assuring algorithm-based verification agents. *International Cross-industry Safety Conference.* Amsterdam.

Hawkins, R., & Kelly, T. (2014). A Structured Approach to Selecting and Justifying Software Evidence. *5th IET International Conference on System Safety.*

Hollnagel, E. (2012). *FRAM: the Functional Resonance Analysis Method, Modelling complex socio-technical systems.* Ashgate Publishing Limited.

Institute of Electrical and Electronics Engineers, Inc. (2016). Standard for System and Software Verification and Validation. *IEEE Standard 1012*.

International Maritime Organization. (2017). GUIDELINES FOR VESSELS AND UNITS WITH DYNAMIC POSITIONING (DP) SYSTEMS. *MSC.1/Circ.1580*.

International Organization for Standardization. (2013). Systems and software engineering, Part 1 Concepts and vocabulary. *ISO/IEC 15026*.

International Organization for Standardization. (2013B). Software and systems engineering Software testing - Part 1: Concepts and definitions. *ISO/IEC/IEEE 29119-1*.

Kahneman, D. (2011). *Thinking Fast and slow.* Farrar Straus and Giroux.

Leveson, N. (2011). *Engineering a Safer World, Systems Thinking Applied to Safety* . MIT Press.

Menon, C., Hawkins, R., & McDermid, J. (n.d.). *Defence Standard 00-56 Issue 4: Towards Evidence-Based Safety Standards.* Heslington, York: Software Systems Engineering Initiative, Department of Computer Science, University of York.

Appendix: Complexity and emergent properties

The American scientist and philosopher John Henry Holland once said: "*emergent behaviour is an essential requirement for calling a system complex*".

Complexity and emergent behaviour, or emergent properties are closely related. There is no single definition of either complexity or emergent property. In the same way, neither among scientists nor among philosophers is there a universal understanding of what the two terms mean. This appendix will give a brief introduction to the two terms.

Scientists, such as Nancy Leveson and Erik Hollnagel have tried to put a sort of taxonomy on complexity:

1. *Interactive complexity* – Related to interaction between sub-systems
2. *Decompositional complexity* – structural decomposition is not equivalent to functional decomposition (functions are distributed among different components and sub systems)
3. *Ontological complexity* – non-formal system models
4. *Non-linear complexity* – causes and effects are subtle and not obvious
5. *Mathematical complexity* – Related to the number of system states
6. *Dynamic complexity* – change over time – what is the consequence of a "minor" change?
7. *Pragmatic complexity* – Related to the number of system variables
8. *Epistemological complexity* – Related to the number of parameters needed to describe the system

Other scientists explain complexity using the second term: Emergent behaviour: *"In complex systems, many parts are irreducible entwined*".

An example concerning a drop of water by John Henry Holland: "*There is no reasonable way to assign wetness to individual molecules; **wetness is an emergent property** of the aggregate. This is different from the weight; the weight of the aggregate is simply the sum of the weights of the component parts*."

Emergent behaviour can be explained by behaviour or a property that emerges from the interaction between system constituents. These kinds of properties do not exist in each of the constituents, but only as a result of the interaction among them. By reducing the system into its constituents (i.e. reductionism), they are lost, and therefore becomes non-observable.

Macroeconomics, the stock market, the social life of army ants, the wetness of a rain drop (see above), human culture, the global climate, a city's resilience against a catastrophe, **system safety**, are all examples of emergent behaviour or properties. Again: **System Safety is an emergent property**.

Most conceptual models of complex systems use hierarchies as a fundamental tool for modelling these kinds of systems. This is consistent with the cognitive capabilities and preference of humans, and therefore often creates useful models for our understanding. Not all scientists (and methods) agree on the use of hierarchies, and is shown that the dependencies of a certain process become circular, and therefore do not fit into a strict means-end hierarchy.

Many, if not all, traditional methods for system analysis use *reductionism* as the fundamental principle. This has been the tradition for centuries in science. This is the notion that a system may be reduced into its constituents, following an analysis of each of the constituents in order to understand the system, and predict its behaviour. This has been a successful recipe in understanding (and changing) the world around us. In the middle of the nineteenth century, the "British Emergentism" started with the work by John Stuart Mill: *System of Logic* (1843). Out of this tradition, a doctrine of "emergent laws" became central. Although the special science of emergent laws is no longer central in the investigation of emergent properties, this was where it all started.

Unfortunately, this knowledge was perhaps overlooked by the people inventing system hazard and reliability analysis methodologies during and after World War II. FMEA, FTA, Cause and effect, are focused upon single components/events and have been found to be inadequate for complex systems. The

short version is that this is because of their inability to analyse the interaction between the components, and thereby they are incapable of predicting the system behaviour adequately. Or, "*The models are so oversimplified that their validity is questionable*" (Hollnagel, 2012). Moreover, the author asks with respect to Event Trees: "*Does binary branching reflect reality adequately?*"

We need new methods, and artefacts that enable us to adequately analyse and synthesize the system in order to predict the consequences of interaction between the constituents. This might not turn out to be an especially easy task, taking the words of the American mathematician Steven Strogatz: "*I think we may be missing the conceptual equivalent of calculus, a way of seeing the consequences of myriads of interactions that define a complex system. It could be that this ultra-calculus, if it were handed to us, would be forever beyond human comprehension.*"

This statement is perhaps the reason for the following definition of emergent properties: A system "S" with macro-states "P", consists of micro-level parts with dynamics "D" governing the time evolution of "S".

Definition of (weak) emergence:

"Macro-state P of S with micro-dynamics D is weakly emergent iff ("iff": if and only if) P can be derived from D and S's external conditions but **only by simulation**." (Bedau, 1997).

In other words: In order to determine emergent properties, the system must be simulated, or be observed in real life. There is no way (except for known patterns), emergent properties, such as safety, can be determined through a priori analysis.