Nevalainen, R., Clarke, P., McCaffery, F., O'Connor, R.V. and Varkoi, T., 2016, September. Situational Factors in Safety Critical Software Development. In European Conference on Software Process Improvement (pp. 132-147). Springer International Publishing.

Situational Factors in Safety Critical Software Development

Risto Nevalainen [1], Paul Clarke [2,3], Fergal McCaffery [3,4],
Rory V. O'Connor [2,3], Timo Varkoi [5]

[1] FiSMA Association, Finland
risto.nevalainen@fisma.fi
[2] Dublin City University, Ireland
[3] Lero, the Irish Software Research Centre
{paul.m.clarke, rory.oconnor}@.dcu.ie
[4] Regulated Software Research Centre, Dundalk Institute of Technology, Ireland
fergal.mccaffery@dkit.ie
[5] Spinet Oy, Finland
timo.varkoi@spinet.fi

Abstract. The generic software development situational factors model has been developed in order that environments within which software is developed can be profiled and better understood. Situational context is a complex concern for software developers, with a broad set of situational factors holding the potential to affect any one software development project. Safety critical software development is broadly similar to other kinds of software development / engineering. But there are some additional or more dominant situational factors. In this article we conduct a conceptual experiment to define safety critical software development context using situational factors. Eleven such factors are identified, with some of the factors requiring elaboration beyond the detail presently available in the generic situational factors model. We firstly discuss the appropriateness of the selected factors in generic safety critical software development context. Thereafter we apply the selected factors to the medical device and nuclear power domains. Selected situational factors can be used as a high level profile and starting point for more detailed process and safety assessment. Discussion about potential use cases and further development needs is also presented.

Keywords: Situational factors reference model, safety context, safety critical software development.

1 Introduction

Software development is a complex activity [1] and there are a rich variety of products, applications and domains for which software can provide effective solutions. In an initial effort to identify the factors of a context (such as the product or application or domain) that affect the manner in which software is developed, the situational factors model [2] was produced (by the authors of this paper) as a generic set of high level concerns that may influence the choice and form of software

development processes. For the avoidance of confusion, we wish to explicitly identify "environment", "setting" and "context" as synonyms of "situation" in this instance and therefore, we may also refer to these factors as "contextual factors" or "environmental factors" or "factors of the setting" but our preference is to use the term "situational factors" as this is the terminology used in the most complete reference for such factors [2].

The safety critical domain is concerned with "systems whose failure could result in loss of life, significant property damage, or damage to the environment" [3] and software may form part of such systems, for example in anti-lock braking systems (ABS) in cars and in flight control systems in airplanes and rockets. All safety critical software therefore has certain common situational factors that strongly influence the choice of software development processes. For example, it is common for safety critical software systems to be subject to regulation/legislation which demands that risk management is actively and robustly implemented throughout the software development lifecycle (as a mechanism to reduce the risk of events occurring that will adversely affect safety).

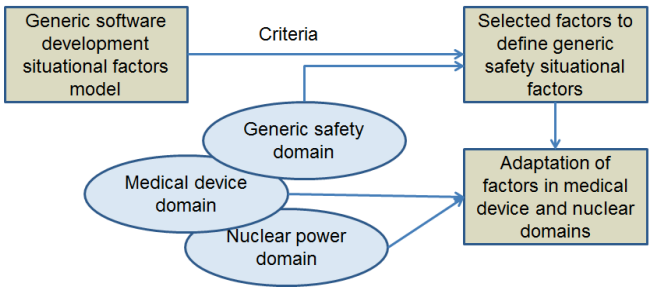

Fig. 1. Rationale in our conceptual experiment

2 Situational factors in software engineering

Although numerous earlier attempts hinted at the existence of (or provided only partial descriptions of) certain contextual factors that affected software development, the situational factors reference model [2] represents the first substantial initiative to unify all factors into a single, comprehensive model. This unified situational factor reference model contains 44 individual factors (and a further 170 sub-factors), and serves to demonstrate that there are a large number of situational variables to be considered when defining and elaborating software development processes (refer to Figure 2), perhaps too large to perfectly satisfy as it has also been shown that the interaction between a software process and its context is analogous to a complex system [1]. Complex systems are characterised by emergent behaviour, which essentially means that our ability to predict and control complex systems may be limited, as is evidenced in ecosystems where a single change holds the potential for unforeseeable, far-reaching and large-scale effects [4]. It is perhaps for this reason that process adaptive capability has been shown to be positively associated with business performance in the general software engineering field [5]. Clearly, however, emergence is not a behavioural property that we want to foster in safety critical

systems and it is (at least partially) for this reason that there are a reduced number of dominant situational factors that affect safety critical software development processes. For this reason, we also find that processes are often formally defined and audited in safety critical domains. The reduced set of safety critical situational factors is perhaps so dominant that they almost eclipse other factors (or they least exercise significant priority over the other factors). We elaborate more on these factors in Section 3.

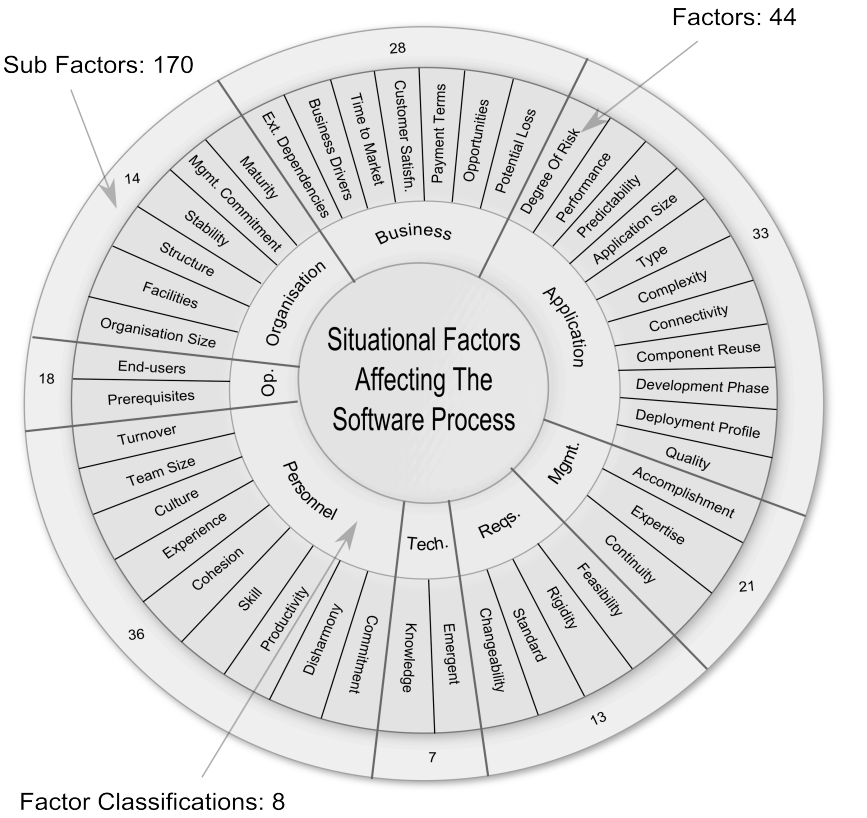

Fig. 2. The Situational Factors Affecting Software Development

3 Dominant situational factors in safety critical domains

3.1 Criteria for identification of specific situational factors in safety critical software development

The starting point in this article is that software development for safety critical systems is quite similar to any other context. So, what is said about software

development at general level is valid also in safety domains. Most software safety standards specify additional requirements and do not repeat the basic principles. Otherwise the standards would be very long, heavy and difficult to use. Of course, there is still some overlap between generic and safety contexts.

Furthermore, different traditions exist in different safety related domains. For historical and regulatory reasons, some requirements that may be highly relevant in one safety related domain may not be considered to be highly relevant in another domain.

The generic situational factors model (refer to Figure 2) is wide and quite detailed. Obviously, some selection criteria are needed to focus on such factors in safety contexts which are more specific from a scoping perspective. Our criteria are the following:

- High importance for safety, based on normative sources (standards, regulations)
- Scalability according to safety requirements (at various safety levels etc.)
- High safety impact and potentially in overall quality of software and also system
- Possibility to create, establish and manage safety culture

The expectation in our experiment is that the following differences between generic and safety context factors may occur:

- Selected safety critical situational factors will remain the same as in the generic sense, but the abstraction level may vary. Typically, a safety context factor may be more detailed and may also be more constrained. We mean that individual factors in the generic model [2] may require further detail for application to a safety critical domain, while at the same time, many of the factors in the generic model can be descoped in light of safety critical concerns. (Ref. Table 1).
- Selected generic safety critical situational factors are partially the same, but the language and definition should be different to be accurate and best aligned with each safety community. A good example of this is the fact that what the medical device domain refers to as “risk” management is referred to in certain other domains as “safety” management [6]. Furthermore, the application of safety levels depending upon the potential for harm can vary between domains.
- A safety critical situational factor may be so different from the generic situational factors model as to be considered additional. It may be also candidate to be added in the generic situational factors model in due course.

3.2 Selected situational factors for further elaboration

Given the argument that there are certain situational factors that dominate safety critical software development, our initial efforts focused on the identification of these factors. Following a number of concept elaboration sessions, each of the factors in the generic situational factors reference model was evaluated for its relevance to the safety critical domain.

From a total of 44 factor classifications in the generic situational factors model, 8 have been identified as being particularly important for safety critical software development situations, the corresponding analysis for which was performed by the authors given their experience in both safety-critical software development and

situational factors affecting software development processes. The exercise to identify the situational factors involved an iterative process whereby the creators of the generic situational factors reference framework proposed factors that might affect generic safety critical software development, following which the authors most familiar with safety critical software development evaluated the proposed factors for relevance and importance in safety critical domains. A total of three iterations were required to render the results published in this paper. From the 8 generic situational factor classifications, a total of 11 situational sub-factors have been identified (refer to column 3 of Table 1).

Table 1. Mapping from generic situational factors to generic safety critical situational factors

Factor	**Description**	**Sub-factor(s)**	**Argument(s): Why dominant in a safety context**
(Operational) Prerequisites	Concerns that must be satisfied prior to operationalisation	Applicable standards; Applicable laws	Safety manual / plan required; Separate safety lifecycle for software development; Degree of required rigour; Degree of independence in functional safety assessment; Degree of independence in V&V
(Application) Type	Nature of the application	Application domain; Application criticality	Degree of safety criticality
(Application) Quality	Application / Product quality characteristics	Required product quality	Degree of diversity (or diverse software); Defence in depth design and programming
(Requirements) Standard	Standard of application / product requirements	General quality of input and output requirements	Detailed requirements especially for outputs and safety properties in almost all software safety standards
(Application) Reuse	Extent to which existing proven software is reused	Required reuse; Extent of utilisation of externally sourced components	This is typically not a separate requirement in safety context, but strict requirements are defined in most safety standards to manage external components (may be called COTS, RUPS, PDS, SOUP etc.)
(Business) Magnitude of potential loss	Impact of negative events	Loss of human life	Magnitude of potential loss
(Business) External Dependencies	Dependencies outside of the business	Dependency on outside suppliers	Tool confidence level (or similar); Degree of COTS/RUSP qualification

(Personnel) Culture	The culture that exists among the personnel	Team culture	Safety culture

4 Elaboration of situational factors in safety critical software engineering

What we are assuming is that "there are a reduced set of situational factors that dominate safety critical software development". Correspondingly, where the generic situational factors reference model highlights the need to consider the application degree of risk, it does not go so far as to elaborate on different degrees of risk depending on the risk classification of the safety critical software.

Since some safety critical software is more critical than other safety critical software, there is often a distinction drawn in various safety critical domains which has the impact of imposing more stringent safety oriented constraints for higher degrees of risk. For example, medical device software that is classified as safety classification A does not require that detailed designs are developed and verified for interfaces between software units whereas medical device software safety classification C does impose such constraints (according to IEC 62304 [7]). We can therefore see the benefit of identifying the dominant factors affecting safety critical software development and where appropriate extending some of those factors with the additional level of detail that is common in safety critical software. For example, the degree of risk associated with the application may be extended to take account of the various different levels of risk.

Process evaluation is also a common feature of safety critical software development and in certain domains there is a basic requirement to pass an independent external audit in order to legally supply software to the sector (as is the case in the medical device sector). However, process assessment can be adapted to satisfy the needs of process audit, since all of the regulatory requirements of an audit can be embedded in a process capability framework – such as is the case with MDevSPICE [8] and Nuclear SPICE [9]. With process assessments, various different types of process assessment can be undertaken, ranging from an internal, first party, informal process assessment to an assessment led by an independent, certified third party. Since it is envisaged that the safety critical situational factors reference model described in this work may be utilised for the purpose of identifying the key situational concerns in advance of a process assessment, the process assessment type is also included as a factor in the generic safety critical situational factors reference list (Table 2). Note that a total of 12 generic safety critical situational factors (Table 2) have been elaborated from the 11 generic situational factors identified in Table 1.

Our first step in adapting and applying the generic software development situational factors model to the safety critical domain is at a generic level, "generic safety critical software"1. We try to identify common factors in numerous domain-

[1] Such "generic safety critical software" may not exist, because most industry sectors use their own standards. Note also that terminology may vary in standards, for example "safety-related software" or "software important for safety".

specific safety standards. Later in Section 5 we apply this generic set to two domains: medical devices and nuclear power. Our approach also allows a comparison between sector-specific profiles.

Many sector-specific safety standards and models have a long history and their own development community. In this paper, IEC 61508:2010 [10] is selected as the main source and reference for generic safety critical situational factors. More specifically, IEC 61508:2010 Part 3 is used, because it is a specific standard for safety related software development. In some sectors, this standard is reasonably well adopted and is the main starting point for sector specific additional requirements and adjustments. Good examples are the process industry (standard IEC 61511), automotive (standard ISO 26262) and railways (for example standards EN 50126, 50127, 50128). Medical device, space, avionics and nuclear sectors are somewhat distanced from IEC 61508, and use their own concepts. The nuclear sector goes further again and has separate standards for different safety classes (IEC 62138 and IEC 60880).

IEC 61508 is the main generic standard for functional safety. Software is only one element, the entire system (including hardware) must be considered. This is also the case in the nuclear sector, where IEC 61513 is the highest system-level standard (and it includes software). If system and software requirements are the same, then a system requirement is more valid. In the medical device sector however, software can be an independent of a physical (i.e. mechanical or electrical) medical device.

Our result from the first step is presented in Table 2. It is a shortlist of selected situational factors based on requirements in the generic safety standard IEC 61508:2010. This standard has a wide range of safety related requirements. For that reason and to make a comparison between sector profiles easier, we propose an ordinal scale for each of the selected factors. It is typically a 3-point or 4-point scale, see Table 2. We try to avoid a binary scale (for example No/Yes), because safety is rather a continuum than black or white. This is easily seen for example in safety integrity levels (SIL), which are in range 1 – 4.

In some cases, IEC 61508 does not have a direct requirement for some highly relevant factor. This may be true because no consensus is achieved as to how some requirement should be formulated. Diversity can be seen as one such factor. The other reason may be that a requirement or topic is not in the scope of IEC 61508 and is assumed to be valid only implicitly or indirectly. One such important topic is safety culture, which is a "soft" factor and may be implemented by organisational management rather than the development unit or project. Many such factors are in sector specific standards, and are therefore important to consider.

Table 2. Safety critical software development, definition of generic profile

Generic safety situational factor (adapted from Table 1)	**Source(s)**	**Range (ordinal scale if possible)**
Separate safety lifecycle for software development	IEC 61508-3, chapter 6, 7, 8	Not Required (NR), Recommended (R), Highly Recommended (HR)
Safety manual / plan	IEC 61508-1, Table A.3	Not Required (NR), Recommended (R), Highly Recommended (HR)
Degree of safety criticality	IEC 61508-1	SIL1…SIL4

Magnitude of potential loss, consequences	IEC 61508-1, 8.2.17	A, B, C, D
Degree of required rigour	IEC 61508-3, Annex C	R1, R2, R3
Tool confidence level (or similar)	IEC 61508-3, 7.4.4 IEC 61508-4, 3.2.11	T1, T2, T3 See ISO26262 Part 8 for further details
Degree of independence in functional safety assessment	IEC 61508-1, Tables 4 and 5	1: independent person, 2: independent department, 3: independent organisation
Degree of independence in V&V (IV&V)	Is specified in many domain specific safety standards, not directly in IEC 61508	1: independent person, 2: independent department, 3: independent organisation Example: ISO 26262 Part 2: Table 1, Table D.1 and 6.4.6.4.
Degree of COTS/RUSP qualification [2]	Is specified in many domain specific safety standards, not directly in IEC 61508	1: independent person, 2: independent department, 3: independent organisation Example: IEC 60880 chapter 15
Degree of diversity (or diverse software)	Is specified in many domain specific safety standards, not directly in IEC 61508	Not Required (NR), Recommended (R), Highly Recommended (HR) Example: IEC 60880, Annex G.5. See also ISO26262 Part 6 method Table 5.
Defence in depth design and programming	Is specified in many domain specific safety standards, not directly in IEC 61508	Not Required (NR), Recommended (R), Highly Recommended (HR) Example: IEC 60880 Chapter 13 (prevention of common cause failures)
Safety culture	Is specified in many domain specific safety standards, not directly in IEC 61508	Not Required (NR), Recommended (R), Highly Recommended (HR) Example: ISO 26262 Part 2, Annex B

As we can see in Table 2, all factor candidates are not well (or directly) defined in the generic functional safety standard IEC 61508. Some are still kept in the list, because they are mentioned in several domain standards (see some examples and references in the range column). It is also possible that the generic IEC 61508 standard is incomplete because of the consensus-driven standardisation process.

[2] COTS = Commercial off-the-self. RUSP = ready to use software product. In some standards, the abbreviation PDS (= pre-developed software) is used. Their meaning is equivalent in practice.

5 Adaptation of generic safety context factors in medical device and nuclear domains

5.1 Safety context definition in medical device domain

Table 3 is an adaptation of the generic safety situational factors (refer to Table 2) to the medical device domain. The medical device domain has long experience in safety standards (both in ISO, IEC and CENELEC) and regulatory body requirements (for example FDA in USA).

Table 3. Safety critical software development, adaptation of generic profile in medical device domain

Generic safety situational factor (see Table 2)	**Additional source(s) in medical device domain**	**Range in medical device domain (ordinal scale if possible)**
Separate safety lifecycle for software development	No lifecycle specified – but typically V-model seen as default IEC 62304 Annex C.4.2	Class A, B, C
Safety manual / plan	IEC 62304 Clause 5.1.1	IEC 62304 Clause 5.1.1 Note 1
Degree of safety criticality	IEC 62304 Clause 4.3	Class A, B, C
Magnitude of potential loss, consequences	IEC 62304 Clause 4.3	Class A, B, C
Degree of required rigour	IEC 62304 Clause 4.3	No scale. Class A, B, C can be used.
Tool confidence level (or similar)	Encourages use of IEC 61508 for tool advice	Proposed scale: Not Required (NR), Recommended (R), Highly Recommended (HR)[3]
Degree of independence in functional safety assessment	ISO 14971 Annex F.3	Proposed scale: NR, R, HR
Degree of independence in V&V (IV&V)	ISO 14971 Annex F.3	Proposed scale: NR, R, HR
Degree of COTS/RUSP qualification	IEC 62304 Clause 5.3.3 COTS is called SOUP in IEC 62304.	Proposed scale: NR, R, HR
Degree of diversity (or diverse software)	IEC 60601-1	Proposed scale: NR, R, HR
Defence in depth design and programming	IEC 60601-1	Proposed scale: NR, R, HR
Safety culture	ISO 14971 Clause 4.2	Proposed scale: NR, R, HR

[3] Medical device standards do not propose any scale for these factors. A scale from generic Table 2 is used here as an option.

Whereas a number of other domains adopt IEC 61508 for the design of Safety critical software the Medical industry does not adopt this safety standard and has instead defined their own safety classification levels within the medical device software process lifecycle standard IEC 62304.

The three main elements within the IEC 61508 standard are addressed differently within a combination of three medical device standards: (1) IEC 62304 [7]; (2) ISO 14971 [11] (the medical device risk management standard) and ISO 60601-1 [12] (the umbrella product level medical device standard).

The first of these areas that are covered within the IEC 61508 Risk Management lifecycle and lifecycle processes is covered in the medical device domain by IEC 62304 directly referencing the medical device standard for risk management (ISO 14971) as central to the IEC 62304 lifecycle process for medical device software. In fact, the risk management process in IEC 62304 references ISO 14971 and extends it only with additional software specific medical device elements that were not included in the more generic ISO 14971 standard.

The second of these 3 areas within IEC 61508 was the definition of Safety Integrity Levels (SILs). The medical device industry does not adopt SILs but instead uses the idea of software safety classes as defined in IEC 62304. Whereas, there are 4 SIL levels within IEC 61508 there are only 3 software safety classes of A, B and C within IEC 62304. Software safety class A means that no injury or damage to health is possible if the software system failed. Software safety class B means that non serious injury is possible if the software system failed. Software safety class C means that death or serious injury is possible if the software system failed. The main reason why the medical device domain uses these software safety classes as opposed to SILs is that SILs are based upon reliability which quantifies both the probability and the severity of harm caused by a software failure. This presents an issue within the medical device sector as the probability of failure of software is assumed to be 100%. Therefore, within IEC 62304 a more simplified approach is adopted as prior to assignment of software safety classes only the severity of the harm that will be caused by a software failure is taken into consideration. Once a software system has been assigned one of the 3 software safety classes, different processes are required for each of the different software safety classes as IEC 62304 specifies what is required for each of the safety classes (for each process). Whenever, a software safety class has been assigned to a software system it is thereafter desirable to make efforts to further reduce the probability of failure of the software (if possible).

The third of these 3 areas within IEC 61508 relates to recommending methods, tools etc. for software development and also provides information in relation to the independence of personnel responsible for performing different lifecycle activities. This is not handled by an individual standard within the medical device domain but rather a combination of standards and in fact IEC 62304 recommends IEC 61508 as a good source for software methods, tools etc. In terms of the medical device sector, information relating to the independence of personnel responsible for performing different lifecycle activities is covered in ISO 14971 as opposed to IEC 62304. ISO 14971 contains requirements for the independence of those performing for example verification and safety assessments.

5.2 Safety context definition in nuclear domain

Table 4 is an adaptation of generic safety situational factors (refer to Table 2) to the nuclear domain. The nuclear industry also has long experience in safety standards (mainly IEC) and regulatory body requirements. Global cooperation is extensive, important and well established, for example the International Atomic Energy Agency (IAEA) based in Vienna. The national level is most important for regulatory issues, because each country wants to define their own policy in nuclear energy and safety. The Common Position [13] is an example of cooperation between authorities in selected European countries.

A predominant feature in the nuclear domain is that the system life cycle and safety life cycle are considered separate. In practice, this means that functionality important to safety has independent systems from operational systems. Naturally, the same applies to software. Safety classes are numbered 1, 2 and 3, 1 denoting the highest safety class. Categories (A, B, C) – A being the highest – are assigned based on Instrumentation and Control (I&C) functions safety relevance.

IEC 60880 [14] covers the requirements for the software life cycle applicable in safety class 1. Additionally, it contains informal annexes on different special software qualification aspects such as defence against common cause failures, tools for software development and qualification, as well as requirements on pre-existing software. IEC 62138 contains graded requirements for software implementing category B and C functions [15]. IEC 60880 and IEC 62138 provide the principles and requirements for software safety classes. I&C functions of category A may be implemented in class 1 systems only, I&C functions of category B may be implemented in class 1 and 2 systems, I&C functions of category C may be implemented in class 1, 2, and 3 systems [16].

Table 4. Safety critical software development, adaptation of generic profile in nuclear domain

Generic safety situational factor (see Table 2)	**Additional source(s) in nuclear domain**	**Range in nuclear domain (ordinal scale if possible)**
Separate safety lifecycle for software development	IEC 60880 Clause 5.3; Annex A IEC 62138 Clause 4.3; 5; & 6	Systems performing category A functions; safety class 1 Systems performing category B or C functions; safety classes 2 and 3
Safety manual / plan	IEC 60880 Clause 5.5 IEC 62138 Clause 5.1.1 & 6.1.1	Software quality assurance plan Quality assurance plan (maybe part of System QA plan)
Degree of safety criticality	IEC 61226 [17] IEC 61513	Categories of functions A, B, and C for I&C functions important to safety Safety classes of systems 1, 2 and 3; unclassified
Magnitude of potential loss, consequences	N/A	
Degree of required rigour	IEC 61513 Clause 6.4.1.2	Safety classes 1 & 2
Tool confidence level (or similar)	IEC 60880 Clause 14; Annex H	none

Degree of independence in functional safety assessment	N/A	
Degree of independence in V&V (IV&V)	IEC 60880 Clause 8; 10	By process requirements, verification team separate from the development management
Degree of COTS/RUSP qualification	IEC 60880 Clause 15	none
Degree of diversity (or diverse software)	IEC 60880 Clause 13.4; Annex G	none
Defence in Depth design and programming	IEC 61513 Annex A.3; Annex C IEC 61226 Clause 5 IEC 60880 Clause 13	Safety classes and categories Safety classes Defence in Depth levels 1 - 5 in IAEA standard INSAG-10
Safety culture	Common Position Clause 1.6	none

Standards in the nuclear domain focus on quality assurance and the prevention of failures rather than analysing the possible consequences of failures. The IEC 61513 standard states:

The highest practicable integrity is generally deemed necessary for any system which prevents or mitigates the consequences of radioactive releases. A lower level of integrity may be acceptable for systems which support protection against there being releases, but do not directly prevent or mitigate them. Consequently, there is not an equivalent scheme to the reliability/risk reduction SIL levels proposed in IEC 61508 in common use in the nuclear sector. This deterministic approach has been found generally sufficient in the nuclear industry and has resulted in practice in the setting of very high targets of all protective functions. However, the nuclear sector does recognise the numerical approach, and methods of probabilistic safety analysis (PSA) may provide clearer targets for the reliability of CB systems [16].

Defence in depth is required for all safety activities. IEC 60880 provides requirements for defences against software design and coding faults which can lead to common cause failures (CCF) of functions classified as category A [15].

6 Discussion & Conclusion

Our conceptual experiment demonstrates – and maybe validates to some extent - that the generic situational factors model can be used as the main source to define dominant safety factors. Adaptation and mapping is however, not straightforward. Most of the selected factors required further elaboration for generic safety critical situational factor identification. Some additional factors are also needed (Table 2). The authors also highlight that while we possess considerable expertise in both safety critical software development and software development situational factors, the exercise to elaborate the generic safety critical situational factors (from the generic software development situational factors) presently lacks an independent validation.

One idea in this experimental research was to propose an ordinal scale for selected safety factors. It would allow for a “safety profile” to be identified, a high-level

common set of system/software specific normative requirements. When each factor value in an ordinal scale is aggregated further, it would be an overall indicator of safety in a given situation and for a given system/software.

Further adaptation of the selected factors from generic safety to domain-specific safety shows remarkable differences in results. Medical device software has much fewer requirements than nuclear domain. Major gaps also exist in the definition of the ordinal scale per each factor. At least, domain-specific standards may not even have such concepts. Maybe for historical reasons, different safety classifications are very popular. Unfortunately, they are quite different and not directly comparable. There may be benefits to adopting the generic IEC 61508 standard as a starting point and baseline in different domains, to improve comparability and cross-domain mapping of concepts and requirements.

Our research is still in early phase and remains highly conceptual. An in-depth validation is needed. Situational safety factors should be piloted and results should be compared between domains. Then it could be possible to improve comparability and better mapping between requirements in different standards.

A safety profile could be a first step in more detailed and well-established safety demonstration, such as safety case definition and assessment of system/software development processes. This is illustrated in figure 3. A safety profile can also be a separate result, some kind of quick analysis of system/software specific safety requirements and their achievement. Early identification of potential gaps could reduce risks in deliveries.

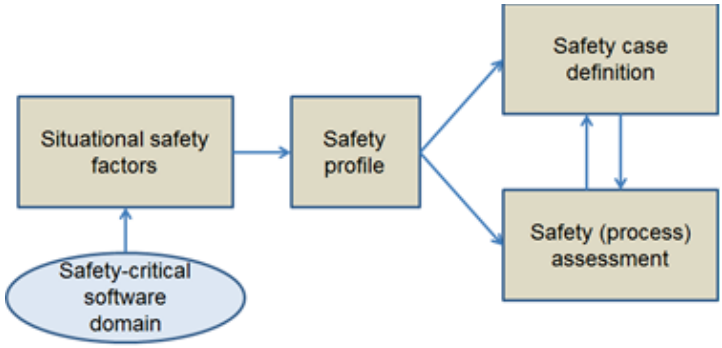

Fig. 3. Use of situational factors in defining a safety profile

The SPICE-based assessment approach is in use both in medical device and nuclear domains. Assessment methods are called with brand names such as MDevSPICE and Nuclear SPICE. The ISO/IEC 330xx family of standards is the main reference and starting point in both methods. Situational factors could extend the methods into earlier steps in supplier and system/software/platform selection.

The proposed safety situational factors contribute mainly in quality, for obvious reasons. They could be selected and elaborated further also by productivity and time criteria. It is a general trend in software and system markets that the overall success is much based on correct timing for markets. More agility is therefore also needed. In an ideal situation, quality and productivity factors would converge on a single point, and this is perhaps an outcome that can be achieved through the more aggressive adoption

of technology-enabled software development processes [18] that we are starting to see emerging in the general software engineering field.

Acknowledgments. This research is supported in part by the Science Foundation Ireland Research Centres Programme, through Lero - the Irish Software Research Centre (http://www.lero.ie) grant 10/CE/I1855 & 13/RC/20194; and in part by the Finnish national nuclear safety program SAFIR2018 (http://safir2018.vtt.fi/).

References

1. Clarke, P., O'Connor, R. V., Leavy, B.: A Complexity Theory viewpoint on the Software Development Process and Situational Context. In: Proceedings of the 2016 International Conference on Software and System Process (ICSSP 2016), IEEE, San Francisco, CA, USA (2016)
2. Clarke, P., O'Connor, R.V.: The Situational Factors that Affect the Software Development Process: Towards a Comprehensive Reference Framework. Journal of Information and Software Technology, 54 (5), 433-447 (2012)
3. Knight, J.C.: Safety critical systems: challenges and directions. In: Proceedings of the 24th International Conference on Software Engineering, pp. 547-550. IEEE, (2002)
4. Manson, S.M.: Simplifying Complexity: A Review of Complexity Theory. Geoforum, 32 (3), 405-414 (2001)
5. Clarke, P., O'Connor, R., Leavy, B., Yilmaz, M.: Exploring the Relationship between Software Process Adaptive Capability and Organisational Performance. IEEE Transactions on Software Engineering, 41 (12), 1169-1183 (2015)
6. Clarke, P., Lepmets, M., McCaffery, F., Finnegan, A., Dorling, A., Eagles, S.: Characteristics of a Medical Device Software Development Framework. In: Industrial Proceedings of EuroSPI 2014 conference, pp. 1-9. (2014)
7. IEC: IEC 62304 medical device software - software life-cycle processes. IEC, Geneva, Switzerland (2006)
8. Clarke, P., Lepmets, M., Dorling, A., McCaffery, F.: Safety Critical Software Process Assessment: How MDevSPICE Addresses the Challenge of Integrating Compliance and Capability. In: Proceedings of SPICE 2015 conference, pp. 13-18. Springer, Berlin, Germany (2015)
9. Varkoi, T., Nevalainen, R.: FiSMA report 2014-2. Advanced nuclear SPICE assessment process. Version 1.0, 2015-01-08. SAFIR2014. FiSMA, Espoo Finland (2015)
10. IEC: IEC 61508, functional safety of electrical/electronic/programmable electronic safety related systems. Parts 1 - 7. IEC, Geneva, Switzerland (2010)
11. ISO: ISO 14971 - medical devices - application of risk management to medical devices. ISO, Geneva, Switzerland (2009)
12. IEC: IEC 60601-1 - medical electrical equipment – part 1: General requirements for basic safety and essential performance. IEC, Geneva, Switzerland (2005)
13. BEL-V, BfS, CNSC et al.: Common position. Licensing of safety critical software for nuclear reactors. Common position of seven European nuclear regulators and authorised technical support organisations. Regulator Task Force on Safety Critical Software (TF SCS) (2013)
14. IEC: IEC 60880, nuclear power plants – instrumentation and control systems important to safety – software aspects for computer-based systems performing category A functions. IEC, Geneva, Switzerland (2006)

15. IEC: IEC 62138, nuclear power plants – I&C systems important to safety – software aspects for computer based systems performing category B and C functions. IEC, Geneva, Switzerland (2004)
16. IEC: IEC 61513, nuclear power plants – instrumentation and control for systems important to safety – general requirements for systems. IEC, Geneva, Switzerland (2001)
17. IEC: IEC 61226, nuclear power plants – instrumentation and control systems important for safety – classification of instrumentation and control functions. IEC, Geneva, Switzerland (2009)
18. Clarke, P., Elger, P., O'Connor, R.V.: Technology Enabled Continuous Software Development. In: Proceedings of the International Conference on Software Engineering (ICSE) Workshop on Continuous Software Evolution and Delivery (CSED), ACM / IEEE, New York, USA (2016)