Recommendations for developing safety-related systems with graphical languages

Nick Berezowski, Markus Haid, and Marie-Elisabeth Hartmann

CCASS Hochschule Darmstadt, Darmstadt, Germany
{nick.berezowski,markus.haid}@h-da.de
http://www.ccass.h-da.de

Abstract. The present paper deals with the development of recommendations for the application of graphical programming languages in safety-related system developments.

The basis for this development is the analysis of existing safety-related systems and the way in which these systems implemented in text-based programming environments meet the applicable norms and standards.

Based on that a present research project analyzes how graphical programming environments can meet these requirements.

The core of the project is the development and validation of recommendations for graphical programming languages to meet these applicable norms and standards, including certification bodies, professional associations, manufacturers and users. The elaboration is limited to concepts and suggestions regarding software aspects.

Finally, these recommendations should be implemented and verified in the specific development environment LabVIEW.

Keywords: functional Safety · safety-related Systems · LabVIEWsafety.

1 Background

1.1 Graphical programming languages

Graphical programming languages show a way in which algorithms and system behavior of programs can be implemented by graphical elements and their arrangement [36]. This paper presents these relationships mainly using the programming language G of the development environment LabVIEW. It is a programming environment of the manufacturer National Instruments, which started more than 30 years ago. The language G describes a flow-controlled model [7], similar to the usual graphic modeling using block diagrams in UML. LabVIEW programs are called Virtual Instruments, or VIs for short. These essentially consist of two components, the front panel containing the user interface and the

block diagram with the graphical program code [8]. This doesn't translate into other programming languages, but the graphical code is compiled directly into machine language [9].
G combines the strengths of the theoretical data flow model with the practice-oriented principles of structured and modular programming, revealing key features of traditional higher-level programming languages [37]. These are simple and compound data types, static typing with strict type checking, hierarchical and polymorphic operations, branches, case distinctions, sequences and loops [37].

1.2 Safety-related systems

Safety-related systems consist of two different parts. On the one hand, a functional hardware control to be monitored and, on the other, safety functions that monitor the correct functioning of the overall system and initiate risk-reducing measures [11]. Safety-instrumented functions as a link between software and hardware represent software code that can enhance the functional safety of hardware components by performing a risk assessment, for example, using the risk graph method according to DIN EN ISO 12100 or ISO 61508-5 [19]. High demands are placed on all hardware and software components [26]. A so far very successful concept for the realization of such safety-related monitoring systems represent programmable logic controllers which are used in the largest industrial sectors of process, automotive, aviation and medical industry. They work internal processes with a cyclic process-driven behavior and can be implemented in a user interface consisting of text-based and graphical-based elements. In addition, there are also purely text-based programming languages and corresponding development environments, such as C, C ++ or Pascal, with which safety-related systems can be developed [35].

1.3 Present procedure

The procedure for implementing applications in computer science is primarily not standardized. Developers can choose from a variety of schematic approaches, implementation methods, and programming languages. Assuming text-based programming, this ultimately consists of a large amount of lines of code that are difficult to use even in smaller projects with multiple programming library accesses [2]. Large projects, with mainly several developers, increasingly restrict the structural quality in terms of maintainability, modularity and extensibility, if no other application description or handling exist [2].
Textual descriptions such as text-based program code are thus a relatively poor working basis for an efficient handling of software applications. There is a lack of abstracting intermediate models that allow people with technical understanding to understand the various abstraction levels and perspectives of the software, thus allowing a gradual approximation to the application [2].

Other industrial sectors, such as mechanical engineering, have already adapted to the complexity of present and future machines [2]. Using CAD, the computer-aided design, two- and three-dimensional constructions could be constructed digitally for the first time in the 50s and 60s of the last century, allowing them to move, adapt and expand freely. Initial programs still worked with digital drawing tables and pen output, later the accuracy of output improved due to higher-resolution screens and printers [5]. As a result of this digital graphical modeling, great development time savings have been made, along with higher value solutions and lower development costs. Reasons for this can be found in the significantly better controllability of complexity in a digital graphical framework [2].

A similar approach can be applied to software development. For example, UML, the Unified Modeling Language, can be considered as the first step in graphical software modeling [2]. UML uses block diagrams to describe the architecture, the design and the implementation of a software. Through this visual modeling, it is possible to produce universally valid structural and behavioral models of the software. The result is a promotion of the intelligibility of the implemented structures solely through the graphical representation [4].

In view of the development in current machines, digitization, communication and flexibility of individual components are in the foreground. Static processes without adaptive manufacturing processes, resources and subscribers should be a thing of the past very soon [6]. Technologically, the industry speaks from the fourth industrial revolution. In order to network the entire corporate structure of companies, automation systems should be no longer controlled in individual processes but independently exchange information between different operating areas and coordinate entire work processes [21]. Possibilities for this can be found in the constant development of communication via the Internet, using microcomputers. This development is also called IoT, which is the acronym for the term Internet of Things. However, the implementation of systems mentioned for industrial environment applications requires the use of abstracting but still structuring programming languages [26].

However, many Internet of Things applications and many Industrie 4.0 applications and implementations must also meet safety-critical requirements, so that application standards for precisely these programming languages have become indispensable in the safety-critical environment such as medical technology, automation systems, the automotive industry and aerospace engineering. The research of future electrical, electronic and programmable electronic automation systems generally depends on the IEC 61508, which is described as the basic standard, which is presented as the basis of all others. [1]. It covers requirements for the entire safety lifecycle, from the concept phase through system development and production start-up to the decommissioning of safety-related products

[11].

The standard distinguishes between hardware and software conception, as well as the management of a project, with all its components. An integral part of the concept and development phases is the software architecture for safety-critical systems. A development environment should be used to adequately support this process by having pre-certified structures or providing guidelines [11].

Management regulations are mostly used for quality assurance, project overview and traceability regarding tests and security features of systems and project constellations [11]. Software rules, including programming language rules, define how to properly handle and comply with policy values in order to create only secure source code, and which will cause harmless consequences only [11]. In addition to various methods for determining the safety integrity and risk analysis of component groups, hardware regulations also specify maximum permissible limits for developed systems [11]. Depending on the field of application, such a system requires different limits and problem approaches, but all of them follow the same basic designs [11]. This is the reason for the norm and guideline development of different industries according to their own requirements in accordance with the legal regulations of European committees. These are european standards, so-called EN, which set the legal requirements. These technical standards may be inaccurate, incomplete, interpretable or even obsolete, but at the legal level represent the state of the art [39].

Programming languages established in the safety-critical environment are the text-based languages C in the area of embedded systems and IL list for programmable logic controllers [35]. It's about these text-based languages in a strongly and weakly typed environment that is aligned to the norms [11]. Strongly and weakly typed programming languages compile a large number of tests that define the use of data types, variables, and other syntax, such as data access or procedure calls, to ensure maximum system safety [11].
Programming guidelines, such as the MISRA-C, additionally support developers in complying with the specifications expected of certification bodies [14]. Other environments are not allowed or used, including graphical implementation.

For the development of functionally safe software code in graphical programming languages like G, various individual criteria can be set up, which are divided into four consecutive categories.
The first step is to check the syntax in a programming language. Each programming language has its own language set and possibilities to connect different elements or commands. If these language limitations are not met, it would not be possible to compile into the machine language, making verification the basic element of any programming language. Most development environments for C or C ++ can query for syntax compliance before a program is transferred. In graphical languages, such as G, this query can be done at every syntactic change.

It therefore has equivalence to text-based languages, such as C [40].

The second step involves compliance with guidelines that represent the proven structures for safety-related systems in various industries. Similar to the first step it is a static code analysis. Advanced rules, such as those defined in policies such as MISRA-C, can be used to review structural constraints. Such programming languages are limited to a smaller language scope [40]. This is limited to a verifiable level of complexity that precludes, for example, the use of pointers to produce only clear traceable code. Such a set of rules in the form of recommendations for a policy is part of the work. For this purpose, the already proven MISRA-C Guideline was roughly compared with the LabVIEW Development Guideline provided by the software manufacturer National Instruments [30]. However, a direct comparison of the two is difficult, because the structure is based on a guide to designing and implementing a project in LabVIEW, not listed rules. VI Analyzer, a LabVIEW's tool, can validate the design suggestions that are made there, making it possible to analyze MISRA-C against LabVIEW compliance. In addition, the VI Analyzer offers the possibility to develop own tests for the static verification of the code [16]. It was noticeable that some rules are already included in the LabVIEW Development Guideline and VI Analyser tests [15]. Others do not need a definition, because LabVIEW has no other way than being compliant due to the building block principle. All yellow marked rules should be included in a new policy and should be checked for compliance [30]. Overall, the MISRA-C defines 141 rules, of which only 121 are required. By means of the color code, the rules to be implemented can thus be reduced to just under one third. The remaining 40 required rules still to be implemented are basically the pure definition of the correct LabVIEW application [30] [17].

The third step is basically to prove that the software code is running properly. This must be ensured by assigning software code to functional requirements, but also requires a test environment to check runtime errors and timing [40]. What software verification can look like from conceptual design to the testing of safety-related systems in graphical programming languages is currently being developed. Using the example of G, the tools NI RequirementsGateway and NI UnitTestFramework could be used.

The third step closes the link between static code verification, dynamic run-time testing and quality management to the fourth step regulatory compliance. This is only possible if design regulations, such as modularity or diversified design, depending on the applicable standards, can be checked. The tools mentioned in the previous paragraph are expected to provide a more general approach due to the iteratively building complexity [40]. A comparison of individual criteria of the standards for possible realization in graphical languages is currently being developed.

2 Future relevance

If you consider UML instead of a modeling language as a own programming language, it fulfills almost all requirements of graphical development. The vocabulary of the language would consist of graphical elements which, by their wording, describe the functionality [2]. Relations and relationships between functions and classes could easily be analyzed and surveyed by their slightly abstracting level of development [2]. However, this is just as the problem in the implementation. Due to the general abstracting approach of UML it is not suitable for the concrete implementation of complex systems even in the extended versions UML 2.0. There is a lack of a specific vocabulary or syntax [2]. Often, UML is also run as an additional layer in parallel with the text-based project, which means that at least two languages are spoken in a project. The problem of such system breaks often lies in the greater complexity in project organization. If any defects occur, this can partially or completely nullify the benefits of graphical modeling [2]. In 2004, Martina Maier already stated that a complete expressive graphical description language would free us from the breaking with languages and thus advance a big step towards to controllability of complexity in software development [3].

In 2015, a research initiative launched by the CAS in Darmstadt with the company National Instruments and other Alliance partners. This is where the term 'LabVIEWsafety' emerged, which should serve as a defining element in the use of graphical languages for programming in high-assurance system development [10].
Through cooperation with various alliance partners and certification authorities, the possibility of a safety-related system development in this and other graphic development environments is likely to emerge in the future.

Even at the beginning of the research initiative it was noticeable that a visualization of the program execution could considerably improve the understanding of complex processes in some points. By means of graphical notation, software can also be understood by non-software specialists, as it maps the function similar to a block diagram. Such a universal and solution-neutral approach is particularly helpful as a communication tool in the team or during prototyping with the customer. The data flow model used here can thus be used flexibly, starting with the design, the modeling and the simulation, over the implementation, up to the test and the validation of the system. Difficult concepts of traditional programming such as dynamic data structures or variables are largely eliminated. Program execution can be done in parallel, without much effort by using special instructions for developers. A structured programming approach is facilitated by clear interface definition [23].

3 Problem description

In the research and development of technical automation systems, there is a trend to design and develop more and more complex systems with a decrease of development times and more complex legal basic conditions. This progressive approach of companies is mainly due to increasing demand and competitive pressure from the continuous automation and autonomization of industrial fields and private environmental influences. An ever increasing subarea of such automation systems are the safety-related systems, which are set to a much higher legal framework than conventional systems [11]. Text-based programming languages are considered established in the context of implementing functional safety. Evidence that graphical languages can not live up to these conditions does not exist. Thus, a responsibility issue arises in relation to a possible further development of safety-related software development [26].

The task of the research thus also represents an comparison, based on the standardization, between these different languages in order to create clear structures, in terms of aptitudes in the safety-related environment, and to ensure a strict typing of the programming language [11].

Since a high degree of clarity in management and thus often long training periods are necessary to develop complex systems in a safety-conscious way, in text-based languages this often only allows a small group of developers. This limits the know-how required for task and solution finding as well as potentials for early error detection [25]. A graphical development environment can bring significant benefits here [25].
In their basic structure, conventional programming languages , whether text-based or graphical, have few prerequisites to control such complex structures in the Internet of Things in a clear way. Software code must always be checked for errors and should have a well-proven compiler. [23]. There is an enormous development effort to create a necessary modular environment for a safety-related system [24]. A similar development effort is evident in graphic programming languages, but according to previous analyzes better clarity and less potential for errors should arise [23].

In addition to the industrial development, programming languages continue to evolve. In 4th generation languages, a high level of abstraction can be achieved through modular design and easy-to-use tools, which can be quickly understood by inexperienced developers and checked for accuracy or extensibility [22]. The principle of a strong abstraction in complex working modules basically enables all people with technical understanding to have access to the programmatic development of programmable electronic systems, especially in the graphical framework [23]. Part of the research work will be the exploration and development of such a higher abstraction level for the safety-related environment in graphical programming languages using the example of LabVIEW.

For very specific safety-critical systems, such as those required for medical devices or individual process plants, reliable, fast and inexpensive design methods are still lacking. There are already products that have been largely realized with programmable logic controllers. However, these are very expensive in small quantities, since there is little combination with components of other manufacturer product ranges. There is a lack of a viable alternative for custom machine design in the industry that could be created through a modular iterative approach [27]. Findings from previous drafts in the text-based and graphical environment should serve to build higher levels of abstraction for graphical system modeling and implementation.

Safety functions provide a link between software and hardware. According to DIN EN ISO 12100, the central standard for risk assessment in the safety-related area, a safety functions is a function of a machine whose failure can directly increase the risk. They represent software code that can increase the functional safety of hardware components by means of a risk assessment. The extent to which the use of a safety function actually minimizes the risk can be determined by means of the risk graph method according to DIN EN ISO 12100 or according to ISO 61508-5 [19] [11]. Such certification for risk assessment for graphical program code does not exist yet, without evidence of inability.

A means of program verification of software code is a formal method. It helps to ensure compliance with legal standards by verifying the accuracy of the algorithms and not just subordinating them to testing. Examples of formal methods can be found in symbolic program execution, as well as the method of pre- and post-conditions according to Hoare [38]. The extent to which these methods can be used in graphic development has not been examined yet. It must be developed a program verification for graphical program code.

4 Objective

The project is dedicated to the topic of using graphical programming languages for safety-related system development for various reasons. On the one hand, the increasingly complex safety-critical system structures of systems to be automated create the need for further development of development environments and programming languages for the application of current, but also new structures for module-based clear systems in the IoT and Industry 4.0 [11]. The work does not pretend to solve current problems immediately, but plans new approaches for graphical system architectures to create comparisons to current approaches. This may be useful for demonstrating operational reliability, as sub-architectures from earlier safety-related constructs could be used.
On the other hand, some approaches of the current industry and its standards are already outdated and thus increasingly difficult to reconcile with the latest developments in the direction of IoT. Some do not provide a graphical devel-

opment approach or can be difficult to apply to graphical languages such as LabVIEW. Although such guidelines and standards are updated in committees every few years, they still rely on the same approaches in current releases [11] [18] [14]. Graphical, well-structured programming tools, could take away an essential level of complexity and thus allow the creation of more complex projects, sometimes with the help of inexperienced developers. In addition, a graphical approach also helps to avoid program code errors, such as misspelling or forgetting to include libraries, as these are simply not possible [24].
By demonstrating and restricting various approaches in a new graphical programming language policy, using LabVIEW as an example, and certifying it, graphic software and hardware manufacturers can legally secure themselves in relation to the various provisions in the standards and provide an additional incentive for researchers and developers of safety-critical systems for the selection of graphical development environments for implementation [27]. A complete new guideline development also creates the opportunity for a better orientation towards future-oriented technology and architectural designs. The aim of this research is to develop recommendations for such a directive, as actual implementation is only possible through close cooperation between manufacturers, certification bodies and the various industrial sectors.

5 Summary

Guidelines, such as the MISRA-C and C ++, consist of a list of rules for safe and consistent programming in the programming languages. The purpose is the simple verifiability of certification bodies. Projects created under this policy thus comply with all legal regulations of the software, by strictly typing the programming algorithms [14]. In order to guarantee a scientific gain of knowledge, the guidelines of Design Science Research (DSR) will be used as a methodological framework for the processing of the presented research questions [32]. In general, already established, fundamental theories and practices are applied, adapted, abstracted or combined in order to generate concepts for solving existing application-related knowledge gaps [31] [33]. The research is not intended to develop a completely new technology and approach for safety-related systems, but also to examine the existing approaches with regard to their applicability in future machine structures, in order to make optimal use of the potentials of previous architectures for future visionary automation applications. The usability of the developed solution concept and the scientifically grounded approach to the preservation of this concept is thereby secured by the iterative research process provided by the DSR [33]. In addition, the already existing approaches are adapted to the efficient development of the overall architecture. The procedure for splitting the problem into subproblems in order to be able to break down the complexity is called Method Engineering [34].

A next step is a further examination of the existing standards from the given

areas and a comparative analysis between graphical and text-based code, as well as various graphical programming languages. In addition to the software architecture, special emphasis must be placed on the hardware, which must be fundamentally divided into different artifacts. Further steps include recommendations for developing current and exploration of new standards and guidelines for graphical programming languages, which could include the creation of proprietary software architectures, security libraries, qualification tools and code analysis tools.

Basically, the total cost of creating safety-related software is divided into two subcategories that are interdependent. These are the joint creation of guidelines by graphical software and hardware manufacturers and our research institute, which will later help guarantee easy certification with graphical languages created safety-related software. The basic prerequisite for this is, first of all, the certification of the development environment in order to prove that all the necessary requirements are met. Beginnings of the analysis can be found in chapter Present procedure.
In order to presuppose on the legal level that all specifications are adhered to in the current development environment, a pre-certification of existing functionalities is an option. Special emphasis should be placed on the basic level of such languages, which usually contain all components of more complex functionalities [15]. Thus, referring to the research aspect, it is derived from a presentation of these towards certification bodies, for concrete analysis.
The observation and reworking of concrete methods and methods for assessing the adaptation of graphical programming structures is an essential part of the data collection to be created in the first steps. For example, expert interviews with persons involved in the certification selection process represent a further survey method in order to gain an overview of the requirements of individual devices with regard to the certification bodies. Document and content analyzes of the industry-specific standards to be investigated in the course of research projects are to be used to find solutions for a wide range of safety-related, electronically programmable devices.
High standards of clarity and traceability are also set for the programming technology. Some of these are already very detailed in the LabVIEW Development Guideline on the example of LabVIEW [15]. However, there are some limitations to interrupts and recursions, exclusion criteria for using dynamic variables, and static verification methods needed in the safety-related part of programming [14].

There is a lack of framework conditions for the use of graphical programming languages for the development of safety-related electronically programmable systems. This creates a need to evolve safety-critical code into the graphical environment [28]. So far, there are neither legal nor systemic requirements to fulfill this goal.

References

1. Braband, J.: Funktionale Sicherheit. https://link.springer.com/content/pdf/10.1007%2F978-3-540-31707-4_14.pdf. Last accessed 13.10.2018
2. Maier, M.: UML: Stärken und Schwächen der grafischen Modellierung. https://www.tecchannel.de/a/uml-staerken-und-schwaechen-der-grafischen-modellierung. Last accessed 10.10.2018
3. Maier, M.: UML: Stärken und Schwächen der grafischen Modellierung. https://www.tecchannel.de/a/uml-staerken-und-schwaechen-der-grafischen-modellierung.pp. 3, Chapter 3, last paragraph. Last accessed 10.10.2018
4. Petre, M.: UML in practice. https://ieeexplore.ieee.org/document/6606618. Last accessed 15.10.2018
5. MiSUMi. Die Geschichte des CAD. https://de.misumi-ec.com/de/customer-service/blog-beitragsleser/computer-aided-design-teil-1-die-anfaenge-der-konstruktionszeichnungen. Last accessed 11.10.2018
6. Weyrich M., Ebert, C.: Reference Architectures for the Internet of Things. https://www.researchgate.net/publication/288855901_Reference_Architectures_for_the_Internet_of_Things. Last accessed 15.11.2018
7. National Instruments: International Directory of Company Histories. Vol. 22. St. James Press (1998). http://www.fundinguniverse.com/company-histories/national-instruments-corporation-history/. Last accessed 13.10.2018
8. National Instruments: Grundlagen zur LabVIEW-Umgebung http://www.ni.com/getting-started/labview-basics/d/environment. Last accessed 13.10.2018
9. National Instruments: Wie funktioniert der Compiler von NI LabVIEW?. http://www.ni.com/tutorial/11472/de/. Last accessed 13.10.2018
10. H.Z.: Auf dem Weg zu Labview Safety. www.etz.de, VDE Verlag, 10/2015. ftp://ftp.ni.com/pub/branches/germany/2015/artikel/11-november/09_Auf_dem_Weg_zu-LabVIEW_Safety_Ronald_Heinze_etz_10_2015.pdf. Last accessed 20.10.2018
11. Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme – Teil 1 bis Teil 10 (IEC 61508:2010)
12. Hilderman, V.: Understanding DO-178C Software Certification: Benefits Versus Costs. https://ieeexplore.ieee.org/document/6983815. 11.10.2018
13. Richtlinie 2006/42/EG des Europäischen Parlaments und des Rates vom 17.Mai 2006 über Maschinen und zur Änderung der Richtlinie 95/16/EG (Neufassung). http://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32006L0042&from=EN. Last accessed 10.07.2015
14. MISRA C++:2008 Guidelines for the use of the C++ language in critical systems. http://frey.notk.org/books/MISRA-Cpp-2008.pdf. Last accessed 07.07.2015
15. National Instruments: LabVIEW Development Guideline http://www.ni.com/pdf/manuals/321393d.pdf. Last accessed 08.07.2015
16. National Instruments: USER GUIDE LabVIEW VI Analyzer Toolkit http://www.ni.com/pdf/manuals/373631d.pdf. Last accessed 30.12.2018
17. MISRA Safety Analysis. http://www.misra.org.uk/Activities/MISRASafetyAnalysis/tabid/92/Default.aspx. Last accessed 08.07.2015

18. Klein, G.: Funktionale Sicherheit nach DIN EN 61511. http://www.tuev-sued-stiftung.de/uploads/images/133974201635838428032 4/funktionalesicherheit-61511.pdf. Last accessed 04.09.2015
19. Sicherheit von Maschinen - Allgemeine Gestaltungsleitsätze. https://www.beuth.de/de/norm/din-en-iso-12100/128264334. Last accessed 12.10.2018
20. Wöhner, M.: Sicher programmieren auf Basis der IEC 61131-3. https://www.computer-automation.de/steuerungsebene/safety-security/artikel/88378/. Last accessed 31.09.2015
21. Was ist Industrie 4.0?. https://www.plattform-i40.de/I40/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html, Last accessed 14.10.2018
22. Yamamoto, S., Kawasaki, R., Nagaoka M.: VGUIDE: 4GL application platform for large distributed information systems. https://ieeexplore.ieee.org/document/545880; Last Accessed15.10.2018
23. Schmid, M.: Kreativität entfesseln. Embedded – Softwareentwicklung für Cyber – Physical Systems. iX Developer – Embedded Software (2/2014)
24. Schmid, M.: Cyber – Physical Systems ganz konkret. Eine neue Generation smarter, dezentraler und vernetzter Embedded Systems erfordert neue Denkweisen und Entwicklungsmethoden. ELEKTRONIKPRAXIS Embedded Software Engineering Report (Nr. 7, 2014)
25. Schmid, M.: Entwicklungsbeschleuniger: Zeit als neue Währung (Teil 1). Ausführbare Rechenmodelle in einem heterogenen Aktor – Framework unterstützen unsere Denkweise und beschleunigen die Entwicklung von Timing in Embedded – Software. ELEKTRONIKPRAXIS Embedded Software Engineering Report (Februar 2015)
26. Rahman, J.: Sensors need to evolve to make Industry 4.0 workable. Control Engineering Europe (04.11.2014). http://www.controlengeurope.com/article/87387/Sensors-need-to-evolve-to-make-Industry-4-0workable.aspx. Last accessed 16.04.2018
27. Rahman, J.: Fünf Kerntechnologien treiben das Internet of Things voran. Markt & Technik (18.12.2014). http://www.elektroniknet.de/markt-technik/industrie-40-iot/fuenf-kerntechnologien-treiben-das-internet-of-things-voran-115696.html. Lastt accessed 16.04.2018
28. Rahman, J.: Beyond IoT: 2015 is the year of Industry 4.0. Electronics Weekly (06.01.2015). https://www.electronicsweekly.com/blogs/viewpoints/beyond-iot-2015-year-industry-4-0-2015-01/. Last accessed 16.04.2018
29. Berezowski, N.: High-Assurance System Development with LabVIEW. Masterthesis zur Erlangung des akademischen Grades Master of Science an der Hochschule Darmstadt (2015)
30. Berezowski, N.: High-Assurance System Development with LabVIEW. 18. GMA/ITG-Fachtagung Sensoren und Messsysteme 2016. https://www.ama-science.org/proceedings/details/2426. Last accessed 12.10.2018
31. Hevner, A. R.; March, S. T., Park, J.: Design Science in Information Systems Research. MIS Quarterly Vol. 28 No. 1, pp. 75-105 (March 2004)
32. Venable, J. R.: A framework for design science research activities. Proceedings of the 2006 Information Resource Management Association Conference (CD), Washington, DC, USA, 21-24 May 2006, Idea Group Publishing, Hershey, Pennsylvania, USA

33. Vaishnavi, V., Kuechler, W., Petter, S.: Design Science Research in Information Systems.
http://www.desrist.org/design-research-in-information-systems/. Last accessed 10.10.2018
34. Agh, H., Ramsin, R.: A pattern – based model – driven approach for situational method engineering. Information and Software Technology, 78, pp. 95 – 120
35. Huelke, M.: Sicherheitsbezogene Anwendungssoftware von Maschinen. IFA Report 2/2016, pp. 13-15
36. Lehrerfortbildung Baden-Wortemberg. Visuelle Programmierung.
https://lehrerfortbildung-bw.de/u_matnatech/informatik/gym/bp2016/fb1/2_ algorithmen/1_hintergrund/2_hintergrund/1_visuell/. Last accessed 18.10.2018
37. Andrade, H. A., Kovner, S.: Software Synthesis from Dataflow Models for G and LabVIEW. http://users.ece.utexas.edu/˜bevans/professional/ asilomar98/hugo-scott.pdf. Last access 18.10.2018
38. Halang, W. A., Konakovsky, R. M.: Einige formale Methoden zur Programmverifikation. Sicherheitsgerichtete Echtzeitsysteme pp 269-306
39. Wilrich, T.: Die rechtliche Bedeutung technischer Normen als Sicherheitsmaßstab. pp. 3-29, Beuth Recht
40. Lalo, M.:Embedded Software Verification for IEC 61508 and ISO 26262.
https://de.mathworks.com/videos/embedded-software-verification-for-iec-61508-and-iso-26262-81727.html?elqsid=1548418121128&potential_use=Education. Last accessed 3.10.2018