HASH FUNCTIONS SUPPORTING MECHATRONIC DESIGN EVOLUTION

Fresemann, Carina;
Falbe, Max;
Stark, Rainer

Technische Universität Berlin

ABSTRACT

Both industry and science point out the need to integrate PLM and ALM since products evolve from mechatronic to smart products. This paper investigates data management tasks fulfilled when creating or improving design. Particularly, the differences and commonalities in design evolution management of the software and hardware disciplines are considered.
This paper introduces a beta version of a hash function based tool, applying the software management mechanism on mechanical revision management.

Keywords: Product Lifecycle Management (PLM), Application Lifecycle Management, Design management, Hash Function, Information management

Contact:
Fresemann, Carina
Technische Universität Berlin
Chair of Industrial Information Technology
Germany
carina.fresemann@tu-berlin.de

Cite this article: Fresemann, C., Falbe, M., Stark, R. (2021) 'Hash Functions Supporting Mechatronic Design Evolution', in *Proceedings of the International Conference on Engineering Design (ICED21),* Gothenburg, Sweden, 16-20 August 2021. DOI:10.1017/pds.2021.431

1 INTRODUCTION

Software and hardware discipline manage design evolutions in a different manner. This conflicts for mechatronic or cyber-physical systems (CPS) (Anderl (2014)) where software and hardware components need to match. This research explores solution elements for the management of design evolution supporting the need of both disciplines.
Traditional products were either software or hardware. The corresponding data management environments have been established accordingly in co-existence, namely Product Data Management (PDM) for mechanical products and Software Configuration Management (SCM) for software applications. PDM is defined as the consistent storage, provision and management of information about products and the associated development processes (VDI 2219). SCM is the discipline of managing the evolution of large and complex software systems (Conradi (1998)). The larger concepts of Application Lifecycle Management (ALM) (Kääriäinen et al (2009)) and Product Lifecycle Management (PLM) (Saaksvuori and Immonen (2008)) have been put in place introducing the goal of supporting the entire lifecycle from the early ideas until the deployment use or even re-use with processes and tools. This extension along the life cycle results in still parallel hardware and software concepts, treating aspects in comparable but not identical manner.
The need for an integration of PDM and SCM is brought-up by several authors (e.g. by Deuter and Rizzo (2016), Eigner, Koch and Muggeo (2017), Hehenberger et al (2016)). They all argue with the enhancement of products from mechatronic products including software and hardware to cyber-physical systems. An integration has to consider several dimensions, such as processes, engineering tasks and methods, relevant data and documents, data and model interconnectivity as well as a supportive tool environment (Hehenberger et al (2015)). The methodological approach as well as the tool support for design evolution still differs between hardware and software as described by Dahlqvist, Crnkovic and Alkslund (2004) and therefore hinders from close collaboration. This contribution aims at bringing both disciplines closer together by supporting the mechanical design evolution with currently lacking tool functionalities as requested by Hehenberger et al (2016). The proposed hash function based approach visualizes the hardware design evolution in the same manner as the software design evolution.
The remainder of this paper details already carried out work on how to bride hardware and software disciplines. A following section provides details on the way SCM tools support the version control process applying hash functions and how the version graph traces design evolution. Finally, the proposed prototype applying the hash function concept to mechanical data management is described.

2 STATE OF THE ART IN VERSION MANAGEMENT

This section first gives a literature-based overview in the area of PDM and SCM integration with a focus on version management. Furthermore, it presents solution approaches as well as remaining needs.

2.1 Literature Review

For this review, the key words PLM/ALM integration, SCM/PDM integration, version management, hash function and PDM have been searched in combinations in google scholar database. For an initial choice the title as well as the provided abstract served as decision base, double entries resulting from the different combinations were removed from the data set. In total 31 papers were chosen as relevant. They were grouped into IT architecture and integration, collaboration, process and methods, hash function related mechanisms and others, e.g. focussing one tool only.
Observing the PDM/SCM integration work performed in the early 2000s e.g. Crnkovic, Alkslund and Dahlqvist (2003) proposed different API based tool couplings and direct data exchange. Newer research for example by Nardone et al (2020) implements and tests OSLC based information exchange between PDM and SCM. Proposals aiming at improving the processes and methods arise from Model Based Systems Engineering (e.g. Hehenberger et al 2015 or Eigner, Koch and Muggeo (2017)). Dahlqvist, Crnkovic and Alkslund (2004) or Nguyen (2006) mention version management as one currently differing element between hardware and software in terms of SCM and PDM but also in terms of processes and methods. Currently the use of hash functions in the context of PDM, SCM and CAD proposes data integrity applications for models and their relations (e.g. by Yu, Au and Chiu (2016) or Lemes and

https://doi.org/10.1017/pds.2021.431 Published online by Cambridge University Press

Lemes (2019)). This contribution aims at extending these approaches. The following sections detail the literature findings with a focus on version management.

2.2 PDM/SCM integration

Hehenberger et al (2015) order the integration of PDM and SCM in three categories, see Figure 1. The first one is on IT level supporting the connection of IT tools and the information exchange between them, the second one mentioned is the understanding and semantics of sent data and the third one is on an organisational level ensuring the information reuse in the respective process surrounding.
Ebeling and Eigner (2017) or Nardone et al (2020) approach the integration from an IT perspective applying Open Servies for Lifecylce Integration (OSLC). The OSLC standard supports linking of design artefacts (Ryman, Le Hors and Speicher (2013)) and thus strongly supports traceability. However, the method and organizational perspective are not covered, Reichwein and Lopez (2019). Deuter and Rizzo (2016) state that due to the different underlying understanding and data models a pure data exchange will not solve the convergence gap. For example is the proposed solution of Eigner, Koch and Muggeo (2017, p.171) supporting the "PDM way" of design evolution, neglecting the merge aspects of SCM. Deuter (2020) proposes a data model in order to allow data manipulation from one tool into another, extending current OSLC limitations. Hehenberger et al (2015) present a data model on a meta level, where they introduce new configuration objects and their relations. Both, Deuter and Hehenberger, do not explicitly describe the influence of their data model on version management principles.

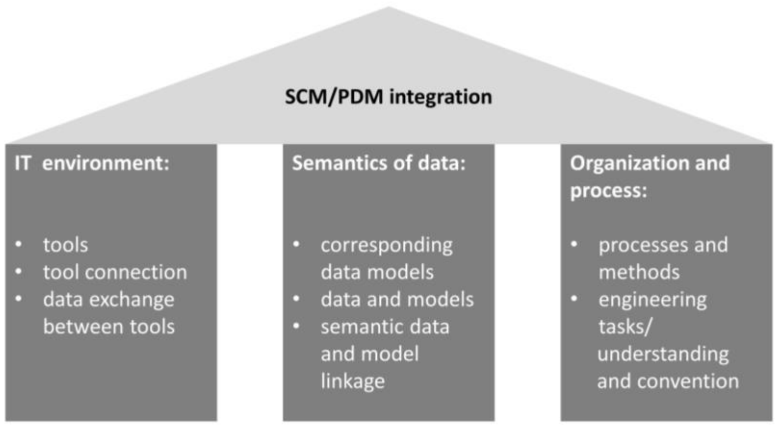

SCM/PDM integration
IT environment:
tools
tool connection
data exchange between tools
Semantics of data:
corresponding data models
data and models
semantic data and model linkage
Organization and process:
processes and methods
engineering tasks/ understanding and convention

Figure 1: Dimensions of SCM/PDM integration, according to Hehenberger et al 2015

2.3 Design Evolution

This sub-chapter compares the understanding and way of working for design evolutions as well as the underlying tool mechanisms in hardware and software discipline based on Crnkovic, Asklund and Dahlqvist (2003). The evolution of a mechanical design object is document focused and regularly improves it, usually treating so-called "minor changes". Correction of errors or improvement are typical driver for the design modification, which are often introduced officially after a first release and supersede their predecessors. Conradi and Westfechtel (1998) explains that software versions may be improvements, new variants of a software item and may be established for the purpose of collaboration. Building a software variant means for example combining software code from several former software configuration items, or probably removing certain content. Software versions are developed in private via branch and check-out by several persons in parallel (Bricogne et al (2012)). Usually one hardware design engineer only carries out the work a via check-out and check-in.
The design understanding and philosophy differs between software and hardware (Dahlqvist, Crnkovic and Alkslund (2004)). For example, where a software engineer establishes a new version a mechanical engineer creates a new object. The data model differs as well: software version control relates predecessor and successor whereas mechanical version control usually follows a sequential approach, which does not necessarily reflect the actual design content evolution (Bergsjö, Malmqvist and Ström (2006 b)). A software version is composed of the predecessor and successor items, their identification, a link plus an indication of the equality between both Conradi (1998), whereas the mechanical item lacks

https://doi.org/10.1017/pds.2021.431 Published online by Cambridge University Press

any differentiation hints Bricogne et al (2012). Several software versions may be applicable at a time and under configuration control indicating the product compilation Conradi (1998). The management of the product compilation is in the mechanical context regularly carried out on a level above the version. The Table 1 summarizes commonalities and differences for version management between the hardware and software way of working including the tool environment as described before. The "Delta" column concludes on the differences and commonalities.

Table 1. Commonalities and differences in the management of design evolution

No.	Software	Hardware	Delta
1	A new version may be established as improvement or new variant from one or several father artefacts.	A new version indicates improvements or corrections of one existing, already released artefact.	Method: Software versions contain broader changes including variants and are generated independent from the release or readiness state.
2	The version evolution is generated automatically and shown in a graph.	The version evolution is created manually and is represented in the BOM structure.	Tool: automatic versus manual evolution tracking.
3	Versions are concurrently organized.	Versions are sequentially organized.	Tool: The concurrent SCM data model provides the design evolution as add-on.
4	Branch and merge supports modifying versions in parallel.	Branch and merge concept does not exist.	Tool: branch and merge in SCM is based on a "commonality" check and account.
5	Several persons develop a version in parallel. ("copy-modify-merge" approach)	One person develops a version. ("unlock-modify-lock" approach)	Tool: the software approach is feasible based on SCM "diff and merge tools".
6	Several versions may be applicable in parallel.	Usually the latest version of a design artefact is considered applicable; in this case, applicability is inherited from the father object.	Method: understanding of version.
7	Configuration control applies to versions.	Configuration control usually applies to the father object.	Method. Tools: support both approaches.

Some approaches are allocating the differences mentioned in the table. El-khoury (2005) proposes to make product variants more explicit in SCM by using the part internal feature, function or class tree and apply a model based approach. Bricogne et al (2012) handle the branch and merge aspects indicated in lines 4 and 5 when he proposes a tool aiming at a geometric comparison and parallel work of several persons. Nguyen (2006) proposes in his data model to compare CAD model internal hierarchical structures for the branch and merge aspect.

Dahlqvist, Crnkovic and Alkslund (2004) request tools functionalities, data models and cultural behaviour to be align for a strong integration. Line 2 in Table 1 describes the advantages of the SCM based approach for the degin evolution visualization. Therefore, this paper aims at providing a comparable service of automation for mechanical design in addition to the aforementioned diff and merge approaches. The next section explains the hash function based functionalities in the SCM environment in detail.

3 FUNCTIONAL PRINCIPLE OF HASH ALGORITHMS IN SCM

This section describes the basics of hash functions and their application for version control in SCM. Hash functions are the underlying technology of bitcoin currency exchange, and are well known for their cryptographic abilities (Yu, Au and Chiu (2016)). Damgard (1989) describes hash functions as injective transformation of source data of arbitrary size into a standard length string, Figure 2 indicates arrows in one direction only. The resulting string h(M) is called hash code.

This work applies the secure hash algorithm (SHA) SHA-1 (Standard FIPS Pub 180-1), analogue to Git Chacon and Straub (2014). The SHA-1 transforms binary input of random length into 40 hexadecimal characters (0-9 and a-f). The hash function works in a way that one input file causes one output file only, or the other way around different input files will always[1] result in a different hash codes. This property is applied in SCM during each check-in or save activity; the hash code serves as internal data base identifier (Conradi 1998; Chacon and Straub, 2014). Like this, new versions are differentiated from existing ones and the database is supposed to be free from double entries. The use of hash functions for identification enables additionally to check the contents for integrity, since the hash code of a correct and of a corrupted file will be different. Furthermore, the unique and short hash codes support quick search functions.

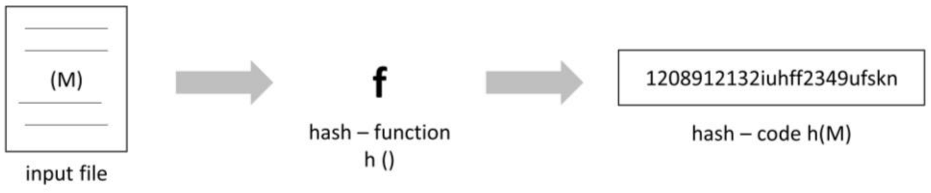


Figure 2: Hash function principle, according to Damgard 1989

The version graph is generated during save and snapshot activities in SCM next to the hash function application. Both concepts, the version graph and the hash code generation are applied to hardware design in a prototype described in the next section.

4 APPLYING A HASH ALGORITHM TO A CAD ENVIRONMENT

The technical realization of integration a hash function in Autocad environment is described in the following section. It is followed by a section on experimental execution and findings.

4.1 Experimental set-up

Figure 3 depicts an architectural overview of the implemented experimental set-up. Autodesk Fusion 2018 serves as CAD environment; it provides an API supporting C# programming language. The programmed API includes the user communication, the different version control and change mechanisms, the hash function and hash code generation as well as the storage. EXCEL serves as data storage for the hash codes and the links between CAD objects. The CAD files themselves are stored locally via explorer folders in this experimental setting.

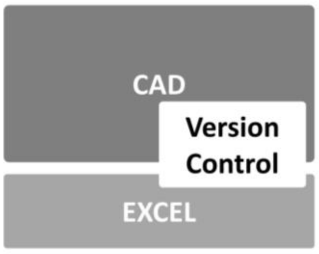


Figure 3: Version control integration in CAD environment

Figure 4 depicts a simplified sequence diagram detailing the implemented version control. When opening or saving a mechanical part in the CAD environment a user-interface opens asking for a decision about what kind of design evolution the designer foresees. Change/branch as well as change/merge activities correspond to "major changes" in the mechanical approach, the revision concept is considered equal in software and hardware. This distinction is supposed to be the intention and responsibility of the designer - in line with current principles in mechanical and software disciplines and even in a future scenario, where the tool might be able to fully and quickly analyse commonalities and differences. The graphical user interface (GUI) box in Figure 4 indicates the section relevant for the user interaction. The user has furthermore the option to describe the change and give an easy to understand name, afterwards the save process for the link between predecessor and successor is launched and the hash code is created.

1 Further reading on boundaries and exceptions e.g. in Preneel (1994).

The intention of SCM and PDM is to provide a framework where given rules in design teams are supported. Therefore, the programmed demonstrator checks the status information as conventionally applied in PDM and checks certain "common process rules". One example is that a released object might only be branched/changed but no longer revised. Here a fix-programmed rule suggests design evolution principles. It may be very useful to investigate and establish common design modification principles between the disciplines for such cases.
The support environment first generates the hash code from the given CAD file and stores this hash code according to the user input in the EXCEL file. The storage process thus indicates the branch and revision link. A graphical representation as already given in SCM has not been implemented.
The light grey depicted boxes in Figure 4 have been foreseen in the concept but have not been implemented. The check-in /check-out functionality is similar in SCM and PDM, thus an integration does not bring benefit for the demonstration purpose; the change/merge functionality needs to be developed based on the preliminary work of Bricogne et al (2012) and Nguyen (2006).

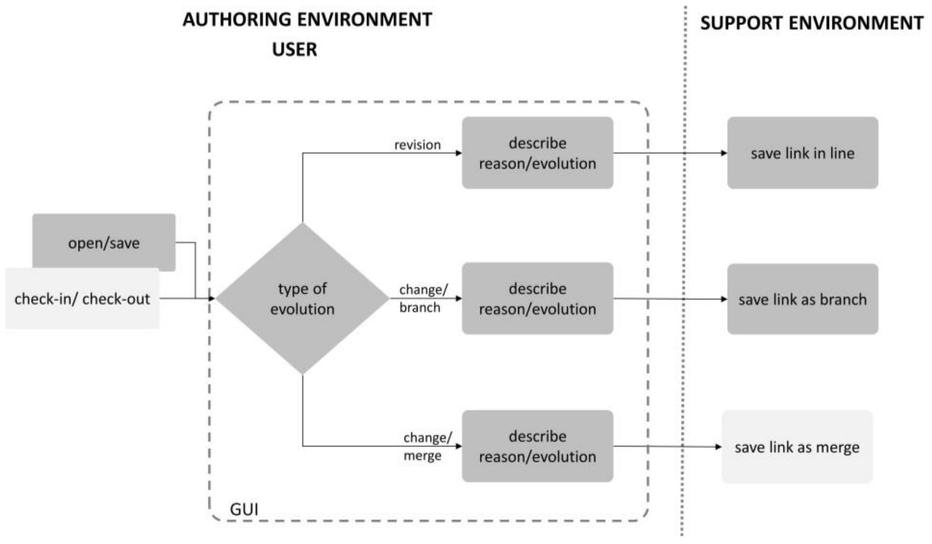
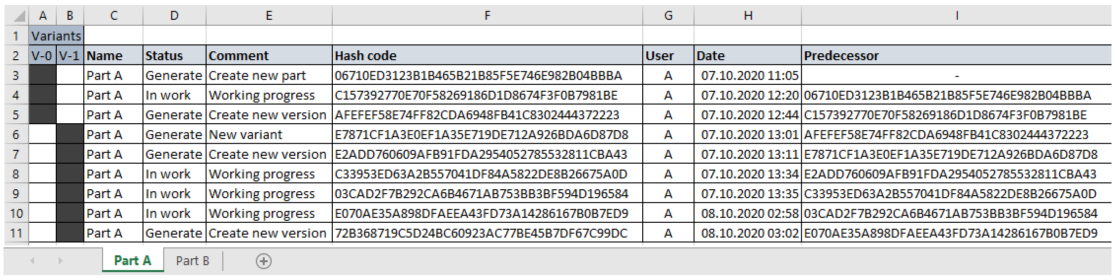


Figure 4: Overview on demonstrator functionality

The experiments are carried out with Autodesk proprietary part data (DWG) as well as the corresponding STEP files. DWG files are converted into a binary file, in order to make it accessible for the hash function. The binary files are not readable for a human being. Step files consist of a header including attributes such as a time stamp and the file name. The content within a DWG and Step file also contains the spatial orientation of the part, the colour coding as well as meta data such as a material choice.
The applied hash function SHA-1 (Standard FIPS Pub 180-1, S. 7–15) takes the entire information found in a file as input if not specified differently. Here the STEP header are excluded, colour and material information as well. The spatial orientation of the model is overwritten by a standard view orientation.

4.2 Results and findings

The EXCEL file representing the data base entries for design evolution is depicted in Figure 5.



	A	B	C	D	E	F	G	H	I
1	Variants								
2	V-0	V-1	Name	Status	Comment	Hash code	User	Date	Predecessor
3	■		Part A	Generate	Create new part	06710ED3123B1B465B21B85F5E746E982B04BBBA	A	07.10.2020 11:05	-
4	■		Part A	In work	Working progress	C157392770E70F58269186D1D8674F3F0B7981BE	A	07.10.2020 12:20	06710ED3123B1B465B21B85F5E746E982B04BBBA
5	■		Part A	Generate	Create new version	AFEFEF58E74FF82CDA6948FB41C8302444372223	A	07.10.2020 12:44	C157392770E70F58269186D1D8674F3F0B7981BE
6		■	Part A	Generate	New variant	E7871CF1A3E0EF1A35E719DE712A926BDA6D87D8	A	07.10.2020 13:01	AFEFEF58E74FF82CDA6948FB41C8302444372223
7		■	Part A	Generate	Create new version	E2ADD760609AFB91FDA2954052785532811CBA43	A	07.10.2020 13:11	E7871CF1A3E0EF1A35E719DE712A926BDA6D87D8
8		■	Part A	In work	Working progress	C33953ED63A2B557041DF84A5822DE8B26675A0D	A	07.10.2020 13:34	E2ADD760609AFB91FDA2954052785532811CBA43
9		■	Part A	In work	Working progress	03CAD2F7B292CA6B4671AB753BB3BF594D196584	A	07.10.2020 13:35	C33953ED63A2B557041DF84A5822DE8B26675A0D
10		■	Part A	In work	Working progress	E070AE35A898DFAEEA43FD73A14286167B0B7ED9	A	08.10.2020 02:58	03CAD2F7B292CA6B4671AB753BB3BF594D196584
11		■	Part A	Generate	Create new version	72B368719C5D24BC60923AC77BE45B7DF67C99DC	A	08.10.2020 03:02	E070AE35A898DFAEEA43FD73A14286167B0B7ED9

Figure 5: Screenshot of several consecutive design modifications

The hash code is depicted in column F "successor hash code", the column I indicates a "predecessor hash code" to each object. The automatically recorded relation between the hash code and its predecessor traces the design evolution. The first columns A and B show the variant evolution. Figure 5 comprises two variants V-0 and V-1, each consisting of several revisions -or improvements of the variant. The "comment" in column E indicates a hint why and what evolves, the user may detail it optionally. In the case, a designer defines the design evolution as "new object" the EXCEL database creates a new sheet, here named part B in Figure 5. The predecessor is stored for this case as well, unlike the behaviour of PDM systems.
A visualization as usually given in SCM needs to be introduced for the version graph, supporting a more comfortable user experience. The merge function in SCM as well as for STEP files starts with a text based approach comparing two files line by line; the user adds differently indicated lines according to design needs. An improvement compared to the proposed solution should visualize differences in a geometrical representation.
The DWG files are proprietary, therefore the relevant sequences detailing content such as colour coding etc. cannot be excluded from the hash generation. These irrelevant differences could not be properly managed for DWG files. Consequently, future applications need to consider the "relevant" hash content only. This relevant content needs to be defined in detail with respect to use and meta data of each file type or model. One way might be to understand the inner data model or structure of the models as executed here for STEP files. Another approach establishes a "replacement model", for example by taking screenshots from each side and applying picture recognition neural networks. Bricogne et al (2012) suggest using CAD vendor built-in geometry similarity search and concludes, that they do not fully support the establishment.

5 CONCLUSION

This contribution reviewed literature on convergence activities for SCM and PLM with a focus on design evolution concepts in software and hardware disciplines. It was shown, that research lately focusses on traceability between objects e.g. via OSLC integration. The convergence on semantic integration of data models as well as the methods for establishing cross-discipline design evolution understanding is still rare. An experimental set-up introduced a design evolution framework for CAD supported by a version graph indicating predecessor and successor including a change indication. Like this, a tool functionality is provided for mechanical design supporting software design evolution methods in a more integrated manner.
Further experimental work should be carried out implementing the here established version control mechanism to a programming shell, such as ECLIPSE. Like that, the application to different authoring environments could be shown. This approach then opens up the field for conclusions on the level of integration between authoring environment and support environment. A direct comparison to the approach of Bergsjö, Malmqvist and Ström (2006 a) managing software in a PDM is suggested. Furthermore, a professional database should replace the EXCEL setting in a way that both meta data and use data from mechanical and software environments can be persisted. Such an experimental setting supports the co-creation in one data management environment.
Further research covers the merge from two or more source files. Merging two mechanical source files is a first step including a precise understanding of what needs to be under hash function control. An easy to understand similarity and differences indication for example by colour coding a geometric or simulation model and adding annotations to it builds a solid foundation. This includes meta data (such as header, colour coding) and use data (e.g. the signal flow in simulation model) of the source files. An easy selection process for carrying over certain geometric features and skipping others is included - similar to the SCM discipline. These findings match and detail the work of Bricogne et al (2012).
Covering the entire data set of PDM makes it necessary to investigate on each data type (e.g. simulations, documents, requirements) and their respective different file formats.

REFERENCES

Anderl, R. (2014) "Industrie 4.0 - Advanced Engineering of Smart Products and Smart Production", Proceedings of International Seminar on High Technology, Vol. 19

Bergsjö, D., Malmqvist, J., Ström, M. (2006 a) "Implementing Support for Management of Mechatronic Product Data in PLM Systems: Two Case Studies.", *ASME International Mechanical Engineering Congress and Exposition*, Vol. 47675, pp. 1175–1183

Bergsjö, D., Malmqvist, J., Ström, M. (2006 b) "Architectures for mechatronic product data integration in PLM Systems", *In DS 36: Proceedings DESIGN 2006, the 9th International Design Conference, Dubrovnik, Croatia*. pp. 1065-1076

Bricogne, M., Rivest, L., Troussier, N., Eynard, B. (2012) "Towards PLM for Mechatronics System Design Using Concurrent Software Versioning Principles", *IFIP International Conference on Product Lifecycle Management*, pp. 339–348, Springer, Berlin, Heidelberg. http://doi.org/10.1145/280277.280280

Chacon, S., Straub, B., (2014) "Erste Schritte". *In: Pro git*, pp. 11–28, Springer Nature

Conradi, R., Westfechtel, B. (1998) "Version Models for Software Configuration Management", *ACM Computing Surveeys (CSUR)*, Vol. 30 No. 2, pp. 232-282

Crnkovic, I., Asklund, U., & Dahlqvist, A. P. (2003). "Implementing and integrating product data management and software configuration management." Artech House. ISBN: 9781580536851

Dahlqvist, A. P., Crnkovic, I., Asklund, U. (2004) "Quality Improvements by Integrating Development Processes", *In proceedings of: 11th Asia-Pacific Software Engineering Conference, pp. 64-72*.

Deuter, A., Rizzo, S. (2016) "A critical view on PLM/ALM convergence in practice and research" *Procedia Technology*, Vol. 26, pp. 405–412

Ebeling, R., Eigner, M. (2017) "OSLC based approach for product appearance structuring.", *Proceedings of the 21st International Conference on Engineering Design (ICED)*, Vol. 4: Design Methods and Tools, Vancouver, Canada

Eigner, M., Koch, W., Muggeo, C., (Eds.) (2017) "Modellbasierter Entwicklungsprozess cybertronischer Systeme- Der PLM-unterstützte Referenzentwicklungsprozess für Produkte und Produktionssysteme", Springer Berlin. http://doi.org/10.1007/978-3-662-55124-0

El-Khoury, J. (2005) "Model data management: towards a common solution for PDM/SCM systems" *In: Proceedings of the 12th international workshop on Software configuration management*, pp. 17-32

Hehenberger, P., Bricogne, M., Le Duigou, J., Eynard, B. (2015) "Meta-Model of PLM for Design of Systems of Systems", *IFIP-International Conference on Product Lifecycle Management*, pp. 301–310, Springer, Cham

Hehenberger, P. Vogel-Heuser, B., Bradley, Tomiyama, T., Achiche, S., D., Eynard, B., (2016) "Design, Modelling, Simulation and Integration of Cyber Physical Systems: Methods and Applications", *Computers in Industry*, Vol. 82, pp. 273–289. http://doi.org/10.1016/j.compind.2016.05.006

Kääriäinen, J., Välimäki, A. (2009) "Applying application lifecycle management for the development of complex systems: experiences from the automation industry." *In: O'Connor, R.V., Baddoo, N., Cuadrago Gallego, J., Rejas Muslera, R., Smolander, K., Messnarz, R., (Eds.) EuroSPI CCIS*, Vol. 42, pp. 149–160, Springer, Heidelberg

Lemeš S., Lemeš L. (2020) "Blockchain in Distributed CAD Environments". *In: Karabegović I. (eds) New Technologies, Development and Application II. NT 2019. Lecture Notes in Networks and Systems*, Vol. 76. Springer, Cham. https://doi.org/10.1007/978-3-030-18072-0_3

Nardone, R., Marrone, S., Gentile, U., Amato, A., Barberio, G., Benerecetti, M., De Guglielmo, R., Di Martino, B., Mazzocca, N., Peron, A., Pisani, G., Velardi, L., Vittorini, V. (2020) "An OSLC-based environment for system-level functional testing of ERTMS/ETCS controllers"; *Journal of Systems and Software*, Vol. 161. https://doi.org/10.1016/j.jss.2019.110478

Nguyen, T. N. (2006). "A unified model for product data management and software configuration management". *In: 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, pp. 269-272

Reichwein, A., Lopez, F. (2019) "Open Services for Lifecycle Collaboration (OSLC)-Extending REST APIs to Connect Data.", *ISWC Satellites*. pp. 329–330

Ryman, A. G., Le Hors, A., Speicher, S. (2013) "OSLC Resource Shape: A language for defining constraints on Linked Data" *LDOW*, Vol. 996.

Saaksvuori, A., Immonen, A. (2008) "Product Lifecycle Management." Springer, Berlin

Standard FIPS Pub 180-1, (1995) "Secure Hash Standard." Online available: nvlpubs.nist.gov, last access: 15.07.2020.

VDI 2219 (2016) "Information technology in product development - Introduction and usage of PDM systems"; VDI-Gesellschaft Produkt- und Prozessgestaltung

Yu, K.M., Au, K.M., Chiu, W.K. (2016) "Watermarking scheme for geometric data protection and detection on 3D CAD assembly model", *Computer-Aided Design and Applications*, Vol. 13, 2016 - Issue 6, pp. 845-854. https://doi.org/10.1080/16864360.2016.1168232