COMPUTER-AIDED DESIGN & APPLICATIONS, 2018
VOL. 15, NO. 4, 488–500
https://doi.org/10.1080/16864360.2017.1419637

A tool for obtaining transparency and traceability in heterogeneous design automation environments

Tim Hjertberg, Roland Stolt and Fredrik Elgh

Jönköping University, Sweden

ABSTRACT

Today, CAD-system are used for much more than just geometric modeling. They are complemented by various software and information sources forming a complete environment for handling all life-cycle aspects of the product. In such systems, the CAD-system works as a central hub. The software and information sources may be of various types making the system highly heterogenous. This presents problems with transparency and traceability in the system making long term management difficult. In this paper, a novel tool is presented to keep track of the dependencies between the various parts of such systems providing an overview and making it possible to predict the effect of proposed changes and facilitating long term management. The tool is tested in a highly heterogeneous environment at a manufacturer of aerospace components, with the result that the traceability is expected to increase at the expense of that time must be spent on defining dependencies and meta-information as the system is evolving.

KEYWORDS

Design automation; dependency management; customization; traceability; transparency

1. Introduction

Re-use of knowledge has been discussed extensively in engineering design [1,3,6]. Several directions have emerged with the purpose of capturing, structuring and applying the engineering knowledge with a varying degree of automation. Design Automation (DA) and Knowledge Based Engineering (KBE) are examples of such directions. The motivation for employing systems for knowledge reuse can for example be to efficiently create design variants or to respond quickly and accurately to requests for quotations. Knowledge reuse is also employed in ETO business where the purpose is quality assurance to make sure that nothing is overlooked in the design process. The motivation can also be saving time or making extensive early stage design explorations.

Systems of this type are often based on connecting the CAD systems with information sources and tools such as mathematics software, FEA code and rule-based tools [16]. Today, CAD systems provide support and built in functionality for building such environments. They may integrate for example tools such as FEA and functionality for defining KBE systems. They also allow users to connect tools and sources of information via the API:s of the integrated software. Thus, the CAD system becomes a central hub surrounded by supporting tools and information sources. These types of interconnected systems where the CAD-system acts as a central hub tend to comprise many systems and information sources making it highly heterogeneous. Careful planning is required when setting up and distributing this type of environments, not least in the global perspective [25].

There are some tools available for planning the architecture such as MOKA [29]. However, the environment may involve many, often heterogeneous parts that evolve over time. There are also integration problems of a very specialized nature, such as the one between engineering and styling in the automotive industry [28]. Special approaches are needed to overcome these difficulties. It may therefore become difficult for those not involved in the system development to get an overview of it. To maintain it over time, transparency and traceability becomes important [9]. How the knowledge is represented and maintained is important for increasing the company's responsiveness to changing conditions [8,11,24]. There are currently no of the shelf systems directed towards the maintenance of such systems. PDM/PLM systems are successfully used for versioning and information structuring, but there is little functionality in afterwards tracing how the system operated in run-time, such as which versions of the system parts were used during a particular run. In addition, there is a need of support for the updating and maintaining the system such as understanding

CONTACT Tim Hjertberg tim.hjertberg@gmail.com; Roland Stolt roland.stolt@ju.se; Fredrik Elgh fredrik.elgh@ju.se

how the parts of the system are connected, not only on a part to part level, but between actual sections of code and sections in the information sources at a high level of granularity. Further, there is also an interest in predicting what effects a particular change will have on the system. The above described gap of knowledge can be formulated as a research question:

> *How can long term traceability and transparency in heterogeneous DA-systems be achieved?*

To address this question, a study has been made at an aerospace company where a prototype tool called "Dependency Manager" (DM) was developed and evaluated. Its primary purpose is tracing the dependencies between documents in the company's early stage multi-disciplinary design analysis system to enhance the transparency of the system and thereby facilitating the maintenance of the system. In is the dependencies between the documents of the DA-system itself such as written design recommendations, scripts and excel sheets that have been targeted, not the documents that are produced to specify the products.

DM is based on automated or manual tagging of sections in textual documents or programming code. This is done either automatically or interactively. The extended markup language (XML) is used to generate a file to store the dependencies between the different sections of code. The resulting file can be visualized using a software called Gephi, (www.gephi.org). In this way, dependencies in a multidisciplinary system for extensive design space explorations can be traced and visualized retroactively provided that the documents and scripts have been tagged and that the relations between the tagged documents have been documented in the XML file. A demonstrator of such tool has been built and evaluated at the aerospace company with the result that it is efficient in providing an overview which is useful both in getting to understand the connections in the system, but would also aid in the maintenance process in predicting the effect of proposed changes.

The remainder of this paper is organized as follows: First, a review of the research with a focus on dependency management in DA and KBE-systems is made in the literature chapter. This is followed by presenting the outcomes of a case study in which the dependency management system is tried on an actual system for extensive multi-objective design analyses at an aerospace company. Beneficial aspects of the dependency management are discussed, an example of a visualization technique is presented, and use scenarios are exemplified. Finally, the results of the evaluation are presented followed by discussions and conclusions.

2. Related work

This paper is focused on the heterogeneous design system itself, i.e. the tools and information sources that is needed to create the design specification. In literature, references that address traceability between items of the design specification (CAD-models, drawing ect.) are found as well as combinations of the two. The described systems for tracing of dependencies have varying scope and purpose and describe the product from different viewpoints and in different levels of abstraction. Dependencies often exists between the documents which can refer each other in many ways depending on the format. Dependencies can act on specific parts of documents, creating a complex dependency structure. In a PhD thesis [7] an interesting example from the computer science domain is found. It shows that IR (Information Retrieval) can be enhanced by various techniques counteracting the shortcomings of IR alone. It also shows that the visualization of the dependencies does help software engineers to understand, maintain, and manage the system.

Further, documents are often subjected to change during the engineering processes and it is important that they are consistent with each other [2]. The management of documents in such heterogeneous environments have frequently been pointed out as important in order to maintain consistency in document clusters and thereby keeping systems and documents valid [14]. Monticolo et al. [19] addresses this problem, focused on the engineering design process and expert models connected to CAD and CAE models. They describe the problem in a concurrent engineering perspective where information such as parameters, expert rules, and mathematical relations are shared by several users in different disciplines. They further state that tools existing today are not capable of managing encapsulated knowledge and cannot ensure that information is consistent through different heterogeneous expert models. A model, called the KCModel (Knowledge Configuration Model), is proposed with the aim to allow for acquisition, traceability, re-use, and consistency of explicit knowledge used in configuration. The solution for consistency is based on checking every knowledge instance used in a knowledge configuration with all other configurations. Scheffczyk et al. [26] proposes the use of strict explicit formal consistency rules in order to obtain consistency in heterogeneous repositories. They present a tool which can be used to automatically achieve consistency or to pinpoint inconsistencies in document structures. By setting priorities to the rules, an impact assessment can be extracted from the inconsistency analysis. Naqvi et al. [22] are focusing on integration problems of System-of-Systems (SoS) and describes

a problem which is commonly seen in development environments for engineering support tools. Systems and applications, initially not intended to work together in a larger context, are combined to create one system capable of performing the actions of the previously separate systems. The separate systems all had their individual goals which now has been merged to a common goal in the larger system. The authors present a language which is supposed to help with the integration of SoSs by using it to represent and analyze natural language-based texts. A tool called EMatrix NPL is used for the semantic analysis of text and is said to be 93% accurate for the English language. The language's intended use is for identifying relevant parts of large documentation collections, and to point out cross-document dependencies. Hutter et al. [15] presents a system called MAYA. The system is described as a tool which maintains formal developments. To interact with MAYA, the user translates specifications to a formal specification language. The specifications contain theories in which, when the specification is translated to the formal language, proof obligations are defined to indicate relations to other theories. External theory provers, such as the one presented in [5] can be connected to the software in order to operate the proof obligations.

Most of the research that deals with the consistency of document clusters are presenting methods of how to achieve consistency by enforcing a set of rules on the content of the documents. Egyed [8] presents a method for automatically detecting and tracking inconsistencies in software design models. Engineers must define consistency rules which are used by the system to automatically detect violations of the rules. The violations are presented to the user who should evaluate if the inconsistencies are relevant to deal with or not. Xiong et al. [32] introduces a language called Beanbag for the purpose of creating automated fixing procedures in software development environments. The language is based on languages for writing consistency relations but is also adapted for the adding of semantics which is used to provide a description of the fixing procedure. Mäder et al. [20] addresses the problem of traceability decay. They present a method for maintaining traceability relations and focuses on re-establishing the traceability after changes have been made to the relations between elements in UML structures during software development. They present a prototype system [21] which recognizes different types of changes in the relations between the elements. Spanoudakis et al. [27] have developed a model and a prototype system, used to generate traceability relations. To do this, traceability rules must be defined manually. These rules are represented in the XML format. From the rules, the prototype system produce four types of traceability relations. A very similar model can be seen in [23].

As can be concluded from this review, a lot of research have been done to the field. Methods and tools exists, which helps software developers or other practitioners to keep their document and system environments consistent and updated. Tools exists which automatically keep track of relations between documents or make changes to code to re-obtain consistency. However, to build the environments required for the tools to work, a lot of manual work must be done prior to obtaining automatic consistency checks. Most of the tools are developed with focus on large scale software development, specific problems or system entities, and are supposed to be used by pure software developers. In the engineering design field, a lot of smaller software tool development projects are performed, without the intention to be part of a larger system in the future, relatable to the System-of-Systems concept. The individual software tools are often developed by the design engineers themselves who perhaps are not very rigorous in doing documentation work, and not by software developers. Relations between functional sections where semantic interpretation of informal statements must be performed is not supported by tools found in this review. Unlike software development projects, the development of such systems is ongoing during the whole life-cycle of the systems. This puts new types of demands on traceability and transparency.

No solution with the ability to explore the content inside of the various types of documents has been found in this literature study. However, there is clearly a need of keeping track of relations between sections in one document type to sections in another document type, and doing this with a low amount of set-up effort. Transparency is something that has been pointed out by tool developing engineers to be an important factor both for gaining trust in the system and for enabling maintenance and thereby increasing longevity [13, 31]. No support tool for software maintenance have been found which considers transparency as an important factor.

3. Requirements for dependency management

Commonalities can be found in research which presents technical solutions for DA systems. The systems are often based on several different commercial software, several programming platforms, and in-house developed software which works together in different ways to achieve a common objective. This structure must communicate with knowledge bases, and PLM systems, as well as follow company practice and regulations as depicted in Figure 1.

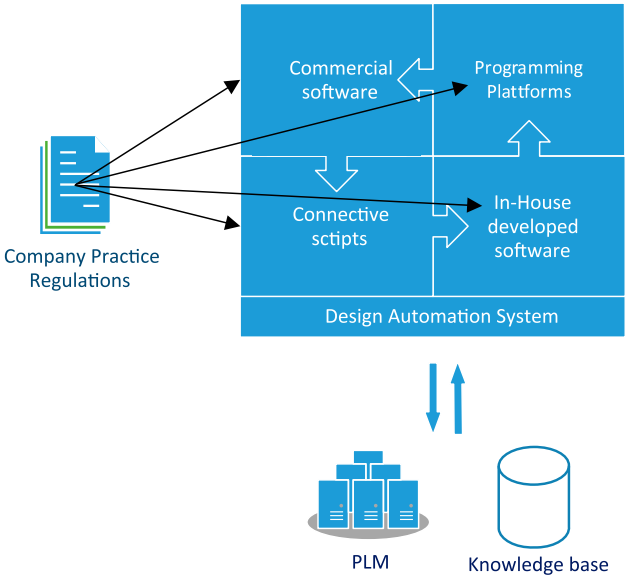

Figure 1. A common structure of DA systems, restricted by the surrounding environment.

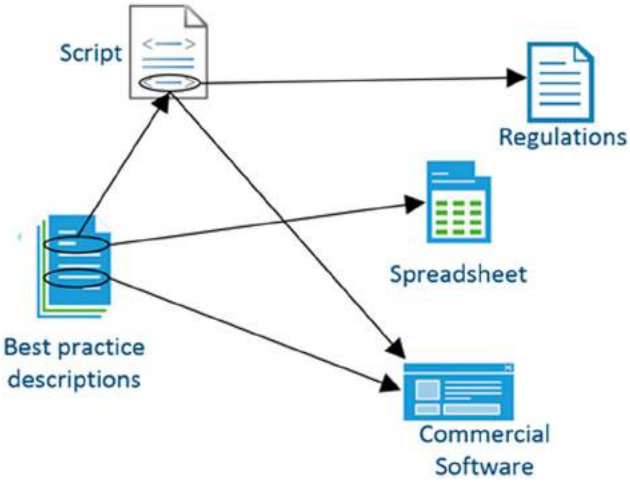

Figure 2. Exemplified low granularity view of DA system. Arrows illustrate dependencies.

3.1. Capturing dependencies

Depending on the type of DA-system, there will be different types of constituents in the system. However, there will almost certainly be some kinds of interactions between them as seen in figure 2, showing dependencies between documents and sections of documents. Depending on the type of dependencies, they can be captured in different ways. In this paper dependencies are divided in two groups i.e. structural, and passive. The structural are those that are directly related to the functions of the program whereas the passive is a result of the structure of the coding itself.

These terms are in analogy with [17] about functional relationships. Dependencies can be captured manually or automatically depending on how they are formalized in the system or documentation. If the dependencies occur in a standardized format, these could be found by an algorithm and be automatically captured. Dependencies which are not described in a predictable way or if it for some reason is not worth to build the structure needed for automatic capture, they can be captured manually. Programming languages usually describes several types of dependencies which easily can be captured automatically. These could be relations between subroutines, functions, classes, and libraries. Dependencies which are typically hard to capture automatically are the passive dependencies. These are often described in natural, non-formal language and might have to be captured manually. Cross-platform dependencies can also be hard to capture automatically since communication between two platforms can occur in several different ways. One must ensure that all ways of communication is covered in the algorithm to ensure that all dependencies are captured and that they are captured in the correct way.

3.2. Granularity levels

Dependencies can be captured in different levels of granularity depending on needs in specific cases. A fine granularity level enables visualization of the system structure in different views. Depending on the purpose of using the system, or what person makes use of it, it might be desirable to have this possibility. Setting up the system dependency structure in fine granularity enables different stakeholders to filter the view to suit their discipline or wanted level of abstraction. An example of granularity levels can be seen in figure 3 where the children of

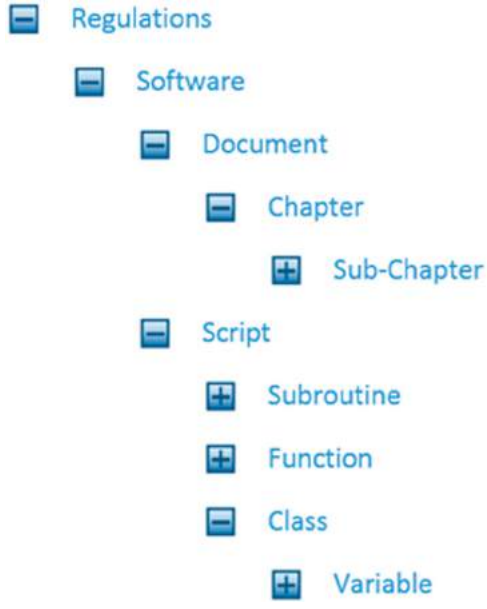

Figure 3. Example of granularity levels.

a parent represents a finer grained representation of the parent.

To exemplify the concept from another discipline, consider a production unit. The production unit consists of a number of manufacturing cells. The manufacturing cells contain a number of different machines which can make use of a set of tools. The tools themselves have a number of properties and data connected to them. Considering the production unit as the observed system, describing it with the manufacturing cells as the smallest entity would be a course grained description whereas a description with tool properties as the smallest entity would be fine grained. The level of granularity should be adapted to the receiving person.

3.3. Meta data

By adding meta data to captured dependencies, or while capturing dependencies, the efficiency of the utilization of the stored dependency structure can potentially be increased. Information about the person who captured a specific dependency enables the possibility to contact this person for consultation when a change is planned for a considered dependency. Descriptions of the purpose of the dependency and how the affected system entities interact technically, enables engineers to be quickly informed and saves them from going through code or documentation. If there are any specific demands which are required to keep the dependency valid, this could be added here. Examples include scripts that needs to be associated with a specific version of a commercial software to work, or that a variable need to be kept within a certain range.

3.4. Visualization

The captured dependencies can be used to visualize the system structure in different ways to obtain overviews of the system. Informative views can be obtained by configuring the dependencies using the meta data and the granularity levels. Utilization of filtering and clustering techniques provides possibilities to create discipline specific views by removing irrelevant parts or by putting focus on relevant parts. By using the granularity levels, views which require prior knowledge about the system can be obtained. This enables the creation of visualizations adapted for the individual stakeholder.

3.5. Transparency/accessibility

The dependency structure can be used to obtain transparency of the system environment. By providing direct access to system components such as scripts or descriptive documents through the utilized visualization approach, the engineers would not have to search for the files, and could also be guided to the correct place inside the considered system entity without manual navigation or interaction with PLM or version control systems. Interfacing functionality could also provide previews and editing capability of system entities without having to open them in their native development environment.

3.6. Impact assessment and change propagation

During maintenance of system entities, it can be hard to assess the effect of a change, to other system entities. Through the dependency structure, the engineers can get estimations of the impact of a change depending on what types of relations it has to other system entities, or how many dependencies the entity considered for change have to other parts of the system. The finer the granularity in which the system is described, the higher the accuracy of the impact assessment will be.

When a change is made, its effect will propagate through the system via the dependency structure. Depending on the nature of the change, it might affect components of the system, outside of the changed component. Further change might have to be done to affected components to regain consistency. This behavior can thereby keep propagating through the system. By investigating meta data captured in the dependency structure, engineers can determine if change also must be made on interfacing components.

4. Case study

In this chapter, the implementation of a demonstrator in a real industrial setting is described. It was done to show how the above requirements can be met. It is presented together with a suggestion of an approach, based on the modeling and management of dependencies between functional sections inside and across different types of system components, which is aimed to aid implementation and management of DA tools.

4.1. The environment for the implementation

The aerospace company is a global player in development, production, service and maintenance of components for aircraft, rocket and gas turbine engines with high technology content. The company provides products that are completely custom engineered in an international market with high competition. The products are integrated in complex systems working in extreme environments for long time periods of time with both customer and legal demands for complete documentation

and traceability. The company takes full responsibility for the functionality of their products during its operation including service, maintenance and updates. Fulfilling these harsh requirements is a challenge but at the same time an opportunity to stay competitive. Automation of design and production preparation by knowledge based engineering (KBE) has been used at the company for more than a decade to enable quick adaptation to change in customer specifications and evaluation of different design solutions. To aid the concept development phase, a multidisciplinary analysis system containing KBE applications is currently being developed by the company. The purpose of the system is to provide knowledge of how changes of the design parameters affects the performance on conceptual designs. This knowledge is obtained by performing analyses in several different disciplines simultaneously. Figure 4 shows the general model behind the multidisciplinary analysis system.

The CAD models seen in figure 4 are all variants of the same conceptual design. They have been generated by varying a number of parameters such as lengths and angles in a pre-planned way. Their variation is determined by design of experiments (DOE). Multidisciplinary analyses are thereafter made on each variant to determine for example its structural stiffness, risk of buckling, producibility and aerodynamic performance. The results of these analyses are compiled and visualized so that the development team can explore the concept, building knowledge and identifying possible design refinements.

The system is of a heterogeneous nature and consists of several different commercial software, controlled and held together by in-house developed software and scripts written in several different programming languages. The engineering work at the company follows method descriptions called Design Practices (DP) together with other documents and knowledge sources. These form a starting point for building the system. The DP:s do not provide details such as how the individual parameters of a FEA should be set. Instead, this information is encoded in the scripts and code developed to complete the tasks. There is also explanatory information found in the comments of code. Connecting the DP:s, the scripts and the code is seen as challenging but important in order to obtain a high traceability in the system. Over time, the DP:s as well as the program code will be subjected to changes which will create problems in keeping the connections valid. Aspects such as knowledge traceability is therefore expected to have a great impact on the success of the implementation. In this study, items related to the manufacturability evaluation has been used for testing the how the DM system can trace the dependencies.

4.2. Identifying the target condition

The research project that this paper was written as a part of, employed Design Research Methodology (DRM) [4]. A key principle of DRM is formulating Success Criteria (SC) and enablers (EN). In short, SC:s are envisioned states to be obtained after the research project has ended. Examples of SC:s are: "shorter development time" and "increased number of innovations". The EN:s are conditions that are assumed to have to be fulfilled to achieve the SC:s. An example of an EN is that the company acquires a tool to visualize dependencies between scripts and codes. Then that tool is an enabler for achieving one or several SC.

In the project, workshops and interviews with companies participating in the project resulted in a set of success criteria (SC). For each SC, a set of EN:s was formulated. As seen in table 1, each SC is supported by several of the EN:s. The final SC:s and corresponding EN:s are presented in Table 1.

4.3. Application description

The target is to achieve increased system transparency and facilitated knowledge traceability in the Producibility Assessment System (PAS). It performs producibility analyses by automatically reviewing the conceptual

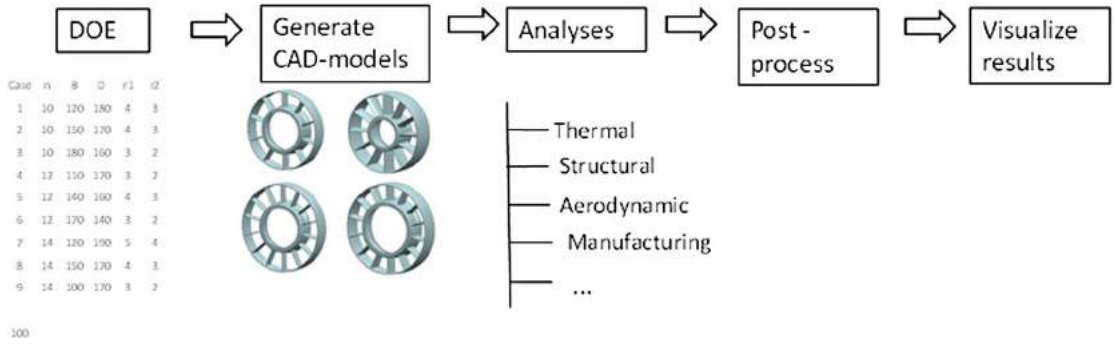

Figure 4. The general model behind the company's multidisciplinary analysis system [30].

Table 1. SC:s and EN:s for successful implementation, derived from company workshops.

Success criteria (SC)	Enablers (EN)
Increased user acceptance	• Results visualization • Domain adapted knowledge
Increased confidence in system	• Results visualization
Reduced time spent to access and understand stored knowledge	• Domain adapted knowledge • Traceability between system, code, and knowledge • Connection between code and normative description
Reduced time making documentation	• Traceability between system, code, and knowledge • Traceability of requirements • Connection between code and normative descriptions
Faster changes to system (Updates or expansion)	• Synchronization of models • Traceability between system, code, and knowledge • Traceability of requirements • Connection between code and normative descriptions

components geometrical features in relation to available manufacturing processes [12]. The PAS is built on several commercial software such as Siemens NX, Microsoft Excel. These are complemented by code and script written in four different programming languages: Python, VB, VBA, and Siemens NX Knowledge Fusion (KF). These have a varying degree of connections to normative descriptions written in non-formal natural language in DP:s, most often written in Microsoft Word. The system is used to perform producibility assessments of a structural component, included in the frame of a jet engine. A simplified version of the component is shown in figure 5. It consists of a number of sheet-metal, cast or forged parts that are welded together using robotic welding.

In the case study presented here, has been limited to part of the PAS that predict the weldability in terms of accessibility of the robot held weld gun and the suitability of the geometries. This part of the system consists of a macro enabled excel file containing VBA programming of the logic to evaluate the manufacturability. It also encompass KF-scripts for the manipulation and reading of the CAD-models a Python program to orchestrate the analysis process.

When the company wants to analyze a design suggestion, all necessary files are copied from a vault to a client computer. It forms a local catalogue structure on the client.

The catalogues of the actual structure contain around 50 files. In addition to the CAD files it is mostly Excel,

Figure 5. Welding assembly sequence of structural jet engine component.

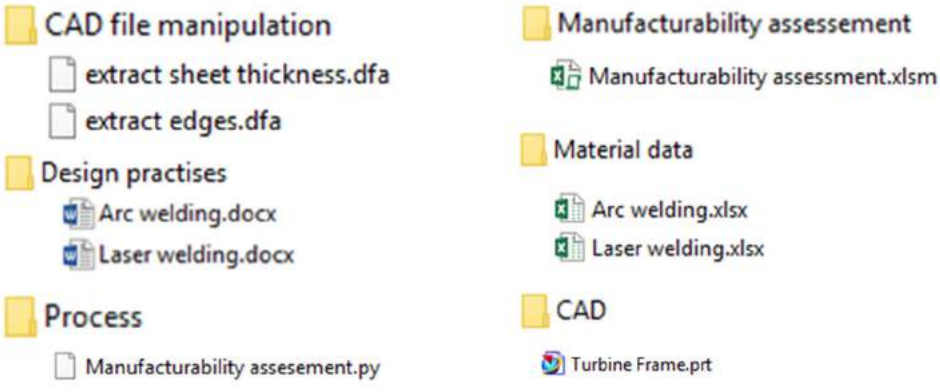

Figure 6. PAS catalogue structure copied to a client.

Word, vba, dfa and python documents. In figure 6, a catalog structure with a limited number of files is shown to visualize the structure.

The execution is governed by a Python script (Process/manufacturability assement.py) It starts a sequence of events starting with the extraction of relevant geometrical elements from CAD models (CAD/Turbine Frame.prt). The extraction is done by KF scripts in the dfa -files (CAD file manipulation/..). The data is transferred to manufacturability assement.xlsm which is a macro enabled excel file containing the logic that test the manufacturability conditions.

Most of these documents have dependencies between that needs to be traced to quickly get an overview. This is useful when trying to understand how the system works or when doing maintenance work.

4.3.1. Capturing dependencies & granularity levels

In some cases, it is necessary to trace dependencies between sections of code. To handle this "tags" are added in the code. A tag is a comment starting with a key word followed by a number such as "#Tag_001" These tags are added when the code is written and are then traced by DM.

In the test-case with the PAS, most dependencies were captured in a semi-automated way. The semi-automated capture method was based on the tags inside the programmed components. The tags contain all information needed to be kept track of each section of code. In the case, it was decided to include the following information in the tags: name of the component, creator of the component, a description of the components purpose, when it was created, when it was last modified, what type of component it is (e.g. a script, a word document, a commercial software). To enable automated reconstruction of the dependency structure from only the information contained in the tags they also included information on which other components of the system it makes use of. A python script was written and used to find and read the tags in all the system components. The script also reconstructed the information in an XML file which later was used to create input files for the visualization tool. To demonstrate the possibility of obtaining fully automated capture of system dependencies, another python script was written. This script included an algorithm which was based on the declarative programming language Knowledge Fusion (KF). KF is used to automate actions in the CAD software Siemens NX. The python script was written to identify all KF based system components and then to recognize when KF scripts called other parts of the script or other KF scripts. In the same way as the script for the semi-automated capture method, the dependencies were reconstructed in XML format and then merged with the dependencies captured semi-automatically. This to enable their inclusion in the same input file for the visualization tool. When the dependency structure was set up on the PAS, the finest granularities consisted of chapters in natural language documents, and subroutines/functions/classes in scripts. This resulted in 81 structural dependencies and 2 passive dependencies. 5 dependencies were caught automatically and 78 were caught semi-automatically. 63 of the 78 are directly connected to how the used programming languages calls or executes other entities of the system. Capture of these dependencies have the potential of being fully automated in the same way as the capture of dependencies between the Knowledge Fusion scripts. This means that 82% of the dependencies in this system has potential in being captured fully automatically with simple algorithms. This is without including possible automatic capture of cross-platform dependencies or attempts to standardize natural language descriptions.

4.3.2. Meta-data

The meta data was in the case study collected in two ways. It was done simultaneously as the capturing of the dependencies, with routines built in to the scripts for automatic and semi-automatic dependency capture. What can be captured with the fully automated method is limited to the information existing "naturally": That is information which would have been there regardless of trying to capture information about the component or not such the name, date of creation and the creator of the component. This is taken from the operating system file information.

In the case of automated capture, the collected meta data was limited to names of KF functions inside KF scripts, the name of the KF script, and the path to the location of the KF script. As for the semi-automated method, any desired information which did not exist naturally in the components, were introduced in the tags used for the dependency collection. This include the type of component and its purpose. As mentioned previously, the information included in the case was: name of the component, creator of the component, a description of the components purpose, when it was created, when it was last modified, and what type of component it is. Information which occurred naturally in the system components were not included in the tags and were thus collected automatically with the same routines as used by the fully automated collection method.

4.3.3. Visualization

In the test case, two different ways of visualization were tested. First, in a regular tree structure like the one in

figure 3 is used. The tree-graph is a natural way of presenting the dependency structure.

However, when the system grows and more dependencies are introduced, it can be hard to keep a clear overview at fine granularity levels. Filtering techniques can be used to improve the ease of use. For the second visualization approach an open source software for network exploration, Gephi [18] was used in order to build graphs. The graphs show system entities as nodes and dependencies as lines between nodes. Several different layout algorithms can be applied to the graphs to produce clear views of the structure. Filtering and clustering techniques can also be applied in Gephi to further improve the usability of the visualized dependencies. A python script was used to generate input files representing the dependency structure of the PAS system as described in section 4.3 for visualization in Gephi. The generated input files were imported into Gephi and resulted in the plots shown in figure 7. Each of the dots in the figure represents one tag in the scrips and the documents. The lines show the connections between them. The arrow shows in which direction the information goes. The color scale from green to red indicates how many interactions a system entity has with other entities. The texts at each node refer to the name of the called routine in each script as seen in the enlargement of the two nodes in figure 7. Meta data can be displayed in the graphs and they can be filtered and searched in order to provide suitable views as for example when changes to the system is being planned or when an overview is needed to introduce a new user to the system.

The graph is a useful tool to get a prediction of the consequences of changing. It can be seen in which other scripts a tagged section is used. Likewise, should there be an isolated dot, a conclusion can be drawn that it is not in use and can be deleted.

4.3.4. Transparency

Transparency was in this case study introduced by providing access to the system entities registered in the dependency structure. Two different technical solutions for achieving this were tested. The user of the system was given the possibility to open the system entity, in its native environment, directly from the visualization tool. Text based entities, such as code or natural language documents, can be displayed in the visualization tool in order to enable quick previews. If a dependency acts on a specific part inside such documents, this specific part is located and displayed to the user in the visualization tool.

4.3.5. Evaluation

To clarify how the proposed solution was perceived to support the SC:s (table 1), three professionals at the aerospace company where interviewed in November 2016 for around two hours. The professionals all have several years of experience of managing product development in the aerospace industry and have a good overview of the product development process at the company. The interviewees were all men around 50 years old. One them is the manager of the R&T department that work with the development of pre-order concepts. The second is responsible for the development of the engineering methods used at the R&T department. The third is the lead engineer of the new engine technology. First, the system was demonstrated to all three of them simultaneously. It was done by showing an example of dependencies

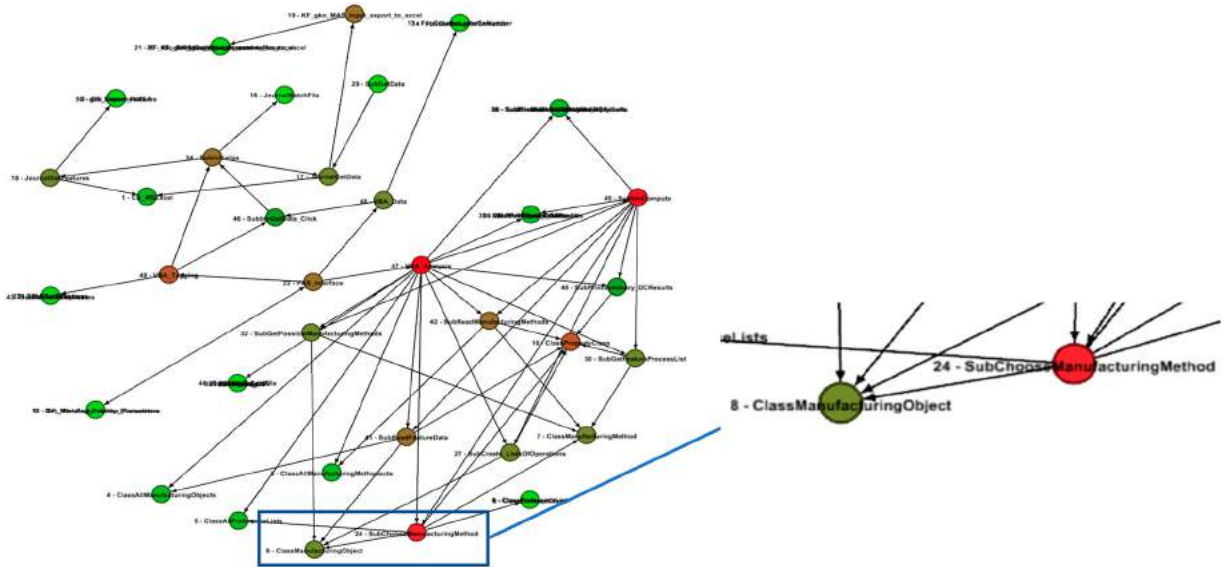

Figure 7. Gephi visualizations of the PAS system.

Table 2. Questions for evaluating DM.

1. Traceability and transparency	2. User perspective
A How will the traceability between design practice and programming code be affected?	A Will it be possible to grasp the dependencies in a short time using the system?
B Will the presented system contribute to a better overview of the information?	B What different types of users can be identified and what are their needs? Can the system as it has been designed meet these needs?
C Will it be possible to keep the relationships updated over long time (several years)?	C How will the time it takes for documentation be affected?
3. The effect on the product development process	D What will happen when a design practice is changed? Will it be possible to update the scripts identified to be affected through the system?
A. Will the system contribute to any of the following:	4. Customer business offer
• decreased variation?	A Will the system in any way contribute to increased customer value? If so, how?
• shortened development time?	
• more efficient use of resources?	
• increased trust in the results?	

tracing between script-files using both tree-graphs and Gephi-graphs. Second, the procedure of managing the relations using the XML-file was demonstrated. Finally, the tagging of code in code blocks inside scripts was demonstrated.

After the demonstration, questions related to the SC:s were individually asked to the interviewees. These questions are shown in table 2 and involve traceability and transparency, user perspective, the effect of the product development process, and the company's business offer to the customer. The answers to the questions given by the professionals are summarized under the headings below.

There was no major disagreement among the interviewees or contradictions in the answers. This was partly due to that the interviewees in some cases highlighted different aspects of the questions. The three following sections summarize the answers for each of the questions and point out were the interviewees agreed, disagreed or when they highlighted different aspects of the question.

4.3.6. Traceability and transparency

A. All agreed that the traceability of dependences will increase at the cost of the time needed to define and maintain the dependencies. One pointed out the there is a risk that it is easy to forget to do the tagging. Another highlighted the complexity with the different stakeholders of the system. It is only the ones involved in the system development that will define the tags.

B. There will be a better overview of the information. One pointed out that it will be possible to keep the set of documents clean i.e. trace obsolete documents for deletion.

C. The answers are not in unison. One said he did not know how much time that would be spent on maintenance. Another pointed out that there must be a firm sanctioning from the management to justify spending time on the maintenance. A third pointed out the risk that the tagging is forgotten and that the system developers put too much trust in the tagging and neglect the commenting of the code.

Further, one of them identified that it will be possible to identify code that is not in use for possible deletion. Two of the interviewees identified a challenge in communicating the need for maintenance. The relations need to be updated as they get obsolete otherwise DM will not gain any acceptance in the organization.

4.3.7. User perspective

A. DM will help the understanding of the structure and dependencies of the system, but under the condition that there still is good commenting and explanations. two of the interviewees agreed. One said it is possible that the effect will be increased overview, but wants to make additional testing on actual documents. One of the interviewees said that, in addition to telling the user that there is a dependency, the system should also help in visualizing the effect of changing something in system. As it is now, the code still needs to be commented and it is the person making the change who must understand the code first to successfully change it. Perhaps tracing the dependencies can facilitate running the system implicitly, i.e. entering the desired result and the getting what parameter settings on the in data that this corresponds to. Setting up the system for backward chaining is thought to be facilitated by DM.

B. Several different types of staff will use the system. Firstly, there are the system developers who also work with maintenance and continuous improvements of the system. Secondly, there are users like project leaders, design engineers and analysis engineers involved. Using the system, the dependencies can be overviewed.

C. The interviewees do not agree on how the system will affect the time spent on documenting. One says it is possible that it will decrease. Another say the time certainly will increase but that the work must be done since traceability is a requirement in the aerospace industry. One point out that it will be hard to add tags into documents already in the PDM-system as that will require new revisions. It would be best to start in research projects where there is no version handling yet.

D. For the updating of existing documents, versioning of the scripts is identified as a potential problem by two of the interviewees. One of them exemplified this by stating that it is sometimes necessary to re-run simulations using the same versions as in the first run. An example is than a deviation has been discovered in the production parts. The third had no opinion on this question.

4.3.8. The effect on product development

A. By variation is meant that the quality of the analyses will vary other time. This variation is expected to decrease since there will be a better control of scrips, programs documents and so on that are used, the interviewees agreed. The development time will decrease, two of the interviewees agreed. One stated that it is an indirect effect. This is because there will be fewer errors in the system.

As for the more efficient use of resources, one did not know. One of them said that efficiency can be obtained by reduced redundancy. The third said that it is related to the confidence in the results. If the quality is high and the users have confidence in the system, the efficiency will be high since the risk of mistakes decreases. The development time will be shortened but more resources are put on setup and maintenance of the system.

Resources are used more efficient since the development process is more focused. There will be increased trust in results since there will be a better integration between documentation and programming code and scripts.

4.3.9. Customer business offer

A. The external customers will likely not notice any difference, two of the interviewees said. One said that it can affect the external customer relations in a positive manner if errors are discovered in the ready products. In such cases, it is vital to quickly be able to show the source of the error. Knowing exactly how the system was set-up and what dependencies existed at a particular point in time can contribute to quickly finding the source of error, which is positive from an external customer relations point of view. Two of the interviewees saw an increased value for internal customers in that there will be fewer errors. If it can be shown that this has led to decreased costs in development, it has a high internal value, one of them stated.

4.3.10. Summary of interviews

The interviews gave some insights in how the SC:s presented are expected to be supported. The SC:s from table 1 are seen below:

- Increased user acceptance
- Increased confidence in system
- Reduced time spent to access and understand stored knowledge
- Reduced time making documentation
- Faster changes to system (Updates or expansion)

The user acceptance is not included in the interview questions since it is too early to evaluate. The interview indicates that the confidence in the results is expected to increase since it is possible to keep better track of what is run. The time spent to access the stored information will decrease. However, the system offers no support in understanding the accessed information. Further, the system does not influence the time spent making the documentation. Additional documentation will be necessary in that the tags and their meta-data will have to be documented. Updates or expansions will be supported to a degree in that the involved documents are readily traced.

The SC that currently is poorly supported is the help in understanding the stored information. Understanding it is a key to knowing how it should be changed. The system needs to support this, possibly by amending the meta-data information on each dependency explaining the reason for the existence of each dependency.

5. Discussion and conclusion

The objective of this work was to introduce a tool for achieving traceability and transparency in heterogeneous DA systems through dependency tracing and management. It sought to answer the question: "How can long term traceability and transparency in heterogeneous DA-systems be achieved?" It will be possible to map the dependencies between the documents, scripts and codes used in the DA-systems. A good overview is provided revealing if part of the system or environment is not used or if obsolete parts are still in use although they should not be. Should for example a script been superseded by an improved version, there is a risk that it is still in use in another part of the system. These problems can quickly be identified and corrected using the tool.

Dependency management was introduced on a sub-document level, allowing cross-document type dependency capturing. Manual labor was reduced by introducing automatic capture of certain dependency types. In the test case, 82% of the total amount of dependencies had potential for automatic capture. Two important parts of the approach is the consideration of granularity levels and the capturing of meta data. These can be used to create clear and explanatory overviews of the system in which the flow of information and knowledge can easily be traced through the system structure without having to laboriously go through documents and code.

The tool uses a XML file to keep a record of the dependencies in the DA-system. This will enable the keeping of a legacy of the development of the dependencies of the DA-system. It will require that the versioning of the dependent documents is included in the XML file meta-data. By regularly generating XML files and managing them in the company PDM systems, the system as it was when the XML-file was created can be reconstructed. It can be run under the same circumstances enabling the detection of errors. This is especially important in aerospace industry with its strict requirements to trace the sources of an error should there be an aviation accident.

From the evaluation of the tool, it was clear that the tracing of dependencies is important when preforming maintenance. However, no support is given in understanding how it is possibilities to change. The person who is making the change must read the comments in the code to understand how it works before being able to change it.

A conclusion based on reviewed literature, industrial input, and the case study presented in this article, is that there is a need for approaches which provides traceability and transparency to the considered type of environments, and that dependency management, visualization and dependency characteristic seem to have potential in achieving this.

6. Future work

Including the reason for the existence of a dependency will make it possible to more quickly grasp how the interlinked documents can be changed has been identified as important and should be included in the future work. Further, a more thorough evaluation of the presented system in the industrial environment of the case company is needed. The visualization of the dependencies and keeping track of the meta-data are also planned to be elaborated further.

Acknowledgements

The authors would like to express gratitude towards the participating companies in the study as well as The Swedish Knowledge Foundation (www.kks.se) for funding the project.

ORCID

Tim Hjertberg http://orcid.org/0000-0002-1608-4523
Roland Stolt http://orcid.org/0000-0001-6278-2499
Fredrik Elgh http://orcid.org/0000-0002-3677-8311

References

[1] André, S.; Elgh, F.; Johansson, J.; Stolt, R.: The design platform – a coherent platform description of heterogeneous design assets for suppliers of highly customised systems, *Journal of Engineering Design*, 2017 (Print).

[2] Becker, S.M.; Haase, T.; Westfechtel, B.: Model-based a-posteriori integration of engineering tools for incremental development processes, *Software and Systems Modeling*, 4(2), 2005, 123–140. http://doi.org/10.1007/s10270-004-0071-0

[3] Bimba, A.T.; Idris, N.; Al-Hunaiyyan, A.; Mahmud, R. B.; Abdelaziz, A.; Khan, S.; Chang, V.: Towards knowledge modeling and manipulation technologies: A survey, *International Journal of Information Management,* 36(6), 2016, 857–871. http://doi.org/10.1016/j.ijinfomgt.2016.05.022

[4] Blessing, L.T.M.; Chakrabarti, A.: *DRM, a design research methodology*, Springer, London, 2009

[5] Bundy, A.: Automated theorem provers: a practical tool for the working mathematician, *Annals of Mathematics and Artificial Intelligence*, 61(1), 2011, 3. http://doi.org/10.1007/s10472-011-9248-8

[6] Chapman, C.B.; Pinfold, M.: The application of a knowledge based engineering approach to the rapid design and analysis of an automotive structure, *Advances in Engineering Software*, 32(12), 2001, 903–912. http://doi.org/10.1016/S0965-9978(01)00041-2

[7] Chen, X.: Automated Documentation to Code Traceability Link Recovery and Visualization, in *The University of Auckland*. 2012, The University of Auckland: Auckland, New Zealand. p. 252.

[8] Egyed, A.: Automatically detecting and tracking inconsistencies in software design models, *IEEE Transactions on Software Engineering*, 37(2), 2011, 188–203. http://doi.org/10.1109/TSE.2010.38

[9] Elgh, F.: Modeling and management of product knowledge in an engineer-to-order business model. in *ICED 11–18th International Conference on Engineering Design - Impacting Society Through Engineering Design*. 2011.

[10] Elgh, F.: Automated engineer-to-order systems – a task-oriented approach to enable traceability of design rationale, *Int. J. Agile Systems and Management*, 7(3–4), 2014, 324–347. http://doi.org/10.1504/IJASM.2014.065358

[11] Elgh, F.: Supporting management and maintenance of manufacturing knowledge in design automation systems, *Advanced Engineering Informatics*, 22(4), 2008, 445–456. http://doi.org/10.1016/j.aei.2008.05.004

[12] Heikkinen, T.; Stolt, R.; Elgh, F.; Andersson, P.; Automated Producibility Assessment Enabling Set-Based Concurrent Engineering, in *Transdisciplinary Engineering : Crossing Boundaries TE2016*. 2016. Curitiba, Brazil: IOS Press.

[13] Hjertberg, T.; Stolt, R.; Poorkiany, M.; Johansson, J.; Elgh, F.: Implementation and management of design systems for highly customized products-state of practice and future research. in *Advances in Transdisciplinary Engineering*. 2015.

[14] Hutter, D.: Semantic Management of Heterogeneous Documents, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2009, 1–14.

[15] Hutter, D.; Autexier, S.: Formal Software Development in MAYA, in *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, D. Hutter and W. Stephan, Editors. 2005, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 407–432.

[16] Johansson, J.: How to build flexible design automation systems for manufacturability analysis of the draw bending of aluminum profiles, *Journal of Manufacturing Science and Engineering, Transactions of the ASME*, 133(6), 2011, http://doi.org/10.1115/1.4005355

[17] Malmqvist, J.; A classification of matrix-based methods for product modeling, in *DESIGN 2002, the 7th International Design Conference*. 2002: Cavtat, Croatia.

[18] Mathieu, B.; Sebastien, H.; Gephi, J. M.: An Open Source Software for Exploring and Manipulating Networks, International AAAI Conference on Web and Social Media; Third International AAAI Conference on Weblogs and Social Media, 2009.

[19] Monticolo, D.; Badin, J.; Gomes, S.; Bonjour, E.; Chamoret, D.: A meta-model for knowledge configuration management to support collaborative engineering, *Computers in Industry*, 66, 2015, 11–20. http://doi.org/10.1016/j.compind.2014.08.001

[20] Mäder, P.; Gotel, O.; Philippow, I.: Enabling automated traceability maintenance through the upkeep of traceability relations, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2009, 174–189.

[21] Mäder, P.; Gotel, O.; Philippow, I.: Semi-automated traceability maintenance: An architectural overview of tracemaintainer, in *2009 31st International Conference on Software Engineering - Companion Volume, ICSE 2009*. 2009.

[22] Naqvi, S.A.; Chitchyan, R.; Zschaler, S.; Rashid, A.; Südholt, M.: Cross-Document Dependency Analysis for System-of-System Integration, in *Foundations of Computer Software, Future Trends and Techniques for Development: 15th Monterey Workshop 2008, Budapest, Hungary, September 24–26*, 2008, Revised Selected Papers, C. Choppy and O. Sokolsky, Editors. 2010, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 201–226.

[23] Olsson, T.; Grundy, J.: Supporting traceability and inconsistency management between software artifacts, in *Proceedings of the 6th IASTED International Conference on Software Engineering and Applications, SEA 2002*. 2012.

[24] Poorkiany, M.; Johansson, J.; Elgh, F.: Capturing, structuring and accessing design rationale in integrated product design and manufacturing processes, *Advanced Engineering Informatics*, 30(3), 2016, 522–536. http://doi.org10.1016/j.aei.2016.06.004

[25] Salchner, M.; Hirz, M.; Stadler, S.; Ameye, J.: A new approach of a global knowledge-based engineering infrastructure, *Computer-Aided Design and Applications,* 14(3), 2017, 366–376. http://doi.org/10.1080/16864360.2016.1240457

[26] Scheffczyk, J.; Borghoff, U. M.; Rödig, P.; Schmitz, L.: Consistent document engineering: Formalizing type-safe consistency rules for heterogeneous repositories, in *Proceedings of the 2003 ACM Symposium on Document Engineering*. 2003.

[27] Spanoudakis, G.; Zisman, A.; Pérez-Miñana, E.; Krause, P.: Rule-based generation of requirements traceability relations, *Journal of Systems and Software*, 72(2), 2004, 105–127. http://doi.org/10.1016/S0164-1212(03)00242-5

[28] Stadler, S.; Hirz, M.: A knowledge-based framework for integration of computer aided styling and computer aided engineering, *Computer-Aided Design and Applications,* 13(4), 2016, 558–569. http://doi.org/10.1080/16864360.2015.1131552

[29] Stokes, M.: *Managing Engineering Knowledge: MOKA Methodology for Knowledge Based Engineering Applications*, John Wiley & Sons, 2001

[30] Stolt, R.; André, S.; Elgh, F.; Andersson, P.: Introducing welding manufacturability in a multidisciplinary platform for the evaluation of conceptual aircraft engine components, *International Journal of Product Lifecycle Management*, 10(2), 2017, 107–123. http://doi.org/10.1504/IJPLM.2017.085950

[31] Verhagen, W.J.C.; Bermell-Garcia, P.; Van Dijk, R. E. C.; Curran, R.: A critical review of Knowledge-Based Engineering: An identification of research challenges, *Advanced Engineering Informatics,* 26(1), 2012, 5–15. http://doi.org/10.1016/j.aei.2011.06.004

[32] Xiong, Y.; Hu, Z.; Zhao, H.; Song, H.; Takeichi, M.; Mei, H.: Supporting automatic model inconsistency fixing. in *ESEC-FSE'09 - Proceedings of the Joint 12th European Software Engineering Conference and 17th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2009.