System Reverse Engineering to Requirements and Tests

Qi Zhang
Fakultät für Informatik
Universität der Bundeswehr München
Neubiberg, Germany
qi.zhang@unibw.de

Andreas Karcher
Fakultät für Informatik
Universität der Bundeswehr München
Neubiberg, Germany
andreas.karcher@unibw.de

Abstract—The long operational phase of products in the aviation industry demands constant maintenance and adaptation of its systems in order to fulfill to the demands of customers and the market and prevent obsolescence. For this purpose, a display system – including all documentation and tools - will be transferred from development to a maintenance department to be supported there over a long duration. To be able to modify the transferred system, maintenance first needs to achieve an understanding of the system. Due to the long development phase, requirements of embedded systems in this domain are most often historically evolved, and only in rare cases formally documented. Typical weaknesses of informal requirements, such as incompleteness or inconsistency, pervade the subsequent levels of the system's life cycle ranging from design to testing. Insufficiently documented system behavior can be increased according to the methods of reverse engineering by analysis of the requirements and design. This article intends to analyze the state of documentation of an existing aviation system in order to create a solution for completing and improving legacy requirements and tests.

Keywords-Avionic; Embedded Multifunction Displays; Maintenance; Requirements; Reverse Engineering.

I. INTRODUCTION

The development of embedded avionic systems is performed by several suppliers, who derive detailed requirements and test cases from high level specifications of an original equipment manufacturer (OEM) in order to develop the commissioned components. All requirements are expected to be well-defined, comprehensible and testable [1] in order to allow integration and interconnection by the OEM according to „systems thinking" [2]. The system comprehension is impaired by conditions such as lack of traceability, inconsistency or incompleteness, especially within and between text-based requirements – company-internal and -external. The deficient documentation leads to difficulties when modifying requirements and test cases during the maintenance phase. Reverse engineering can assist in some ways to the proper maintenance of these legacy systems [8]. The article at hand reflects reverse engineering in the context of software maintenance and further development in the aviation industry. Starting point is a description of the general challenge maintaining legacy avionic display systems. After analyzing the current state of the art for requirement and reverse engineering, the industrial application is presented. Finally, a domain specific concept for improving requirements and tests by reverse engineering is provided.

II. RELATED RESEARCH

Research into requirement engineering has, in the past, been driven by a trend away from documentation centralization towards model centralization [3]. To improve consistency and completeness of requirements, languages and models have been developed [4] [5] which are compatible with extensible markup language (XML). These are governed by standards specifically tuned for loss-less exchange of requirement data between OEM and suppliers [6]. The focus of these methods lies exclusively on the improvement of requirement quality. Whereas the application protocol AP 233 of the ISO standard STEP 10303 [7] offers models for formalization of requirement data, from which areas like testing can also benefit. The underlying idea of this standard is the development of a tool-independent format for long-term archiving and loss-less exchange of data within a complex system. The standard is non-specialized due to its wide range of applicability and requires being adapted to the system in question before its application.

The technologies and methods mentioned above refer to the development phase. In practice, there is usually a finished product given for which the quality is to be improved. Research in the area of reverse engineering is specialized on the analysis of existing products [8]. Reverse engineering is the process of analyzing an existing system in order to identify its components and their interaction, illustrating it in a different or more abstracted form [9]. Although reverse engineering would be applicable to the entire life cycle – starting with existing source code, re-creation of the design and mapping of inherited requirements – the main focus in this area up until a few years ago was on „implementation and design artifacts" [10]. For this reason, Knodel, Koschke, and Mende [10] require the expansion of reverse engineering procedures onto all levels, such as testing and requirements. Similar results are drawn by

 ISBN: 978-1-61208-184-7

Canfora and Di Penta [11] in their article „New Frontiers of Reverse Engineering". There, a road map for reverse engineering is presented, where it is made clear that reverse engineering can go further than recovering design artifacts: requirements are also an important output that can be produced by reverse engineering.

In the following, a concept based on the state of current technologies for improvement of information acquisition between existing requirements and test cases is presented, using the example of the maintenance of symbols on a helicopter's multifunctional display (MFD).

III. INDUSTRIAL APPLICATION

MFDs are used to display flight data and warnings in the helicopter's cockpit. The embedded software has got to be maintained, and the displayed symbols have got to be changed according to customer expectations where necessary. In the context of this application the MFD was developed by a supplier for air traffic control systems, who derived own specifications and test cases from given format specifications (FOS) of the OEM in order to develop the hardware, software and symbol design.

Since the informal FOS are hard to manage, e.g., data regarding size and position is incomplete, the suppliers were free to determine the method and tool with which to create the displayed symbols. The supplier derived formal software requirement specifications (SRS) to describe the logic of MFD input signals. Each combination of inputs signals offers an output signal triggering the desired symbol to be displayed on the MFD. The related test cases are used to verify input signals (as described by the SRS) and the resulting output symbols (as described by the FOS). The overall requirements and test cases can be correlated with each other but not with the FOS. The symbols' design was realized using a human-machine interface (HMI) modeling and display graphics tool, which stores the size and position of the symbols in human non-readable meta files (Fig. 1). This tool was also used to create pictures for visualization of the symbols in the FOS.

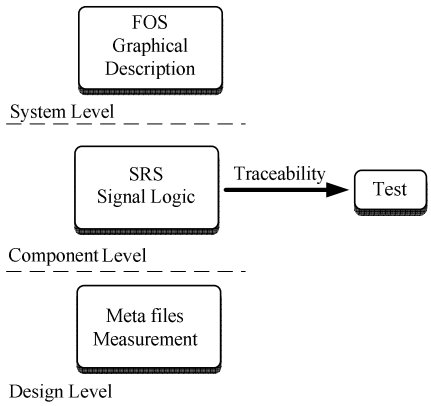

Figure 1. Current traceability of symbol documentation

For this reason, any symbol modification requires to be first implemented in the HMI graphics tool before it can be adapted to the requirements or presented to the customer. The following Figure 2 displays the process for a symbol modification.

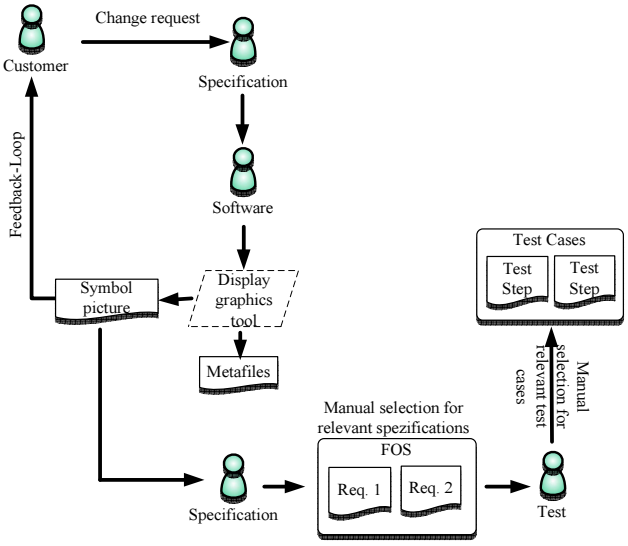

Figure 2. Current process for graphical changes in the MFD

Figure 2 shows that no data is provided from development regarding the criteria on which test cases were developed. Symbols are only tested as output in connection with the signal logic, not as a separate unit. This means that test cases such as „is the symbol ambiguous?" or „can critical conditions like 'white text on white background' occur?" do not exist. For this reason, symbol tests are incomplete.

This missing data is to be rectified during maintenance in order to improve the quality of inherited requirements and test cases of the MFD.

IV. PROPOSED IMPLEMENTATION CONCEPT

With the symbol requirements from development phase being incomplete, a collection of all symbol data in a symbol library is proposed. The idea of maintaining the symbols in a library/database is derived from geographical information systems and mine mapping [12]. The symbol library is to store, in an appropriate structure, data such as a unique attribution, conditional and positional information and its traceability to all relevant requirements and test cases. This builds the foundation for the collection and interconnection of all data from different documents. For processing symbol data like visualization data exchange is done via XML schema. XML offers, besides its platform independence [5], advantages such as translation of the symbol data from and into a database or a scalable vector graphics (SVG)-based graphical user interface (GUI). This is meant to facilitate the access to relevant documents in the case of a modification. Additionally, the collection and representation of data are supportive to the creation and modification of test cases for symbol verification.

 ISBN: 978-1-61208-184-7

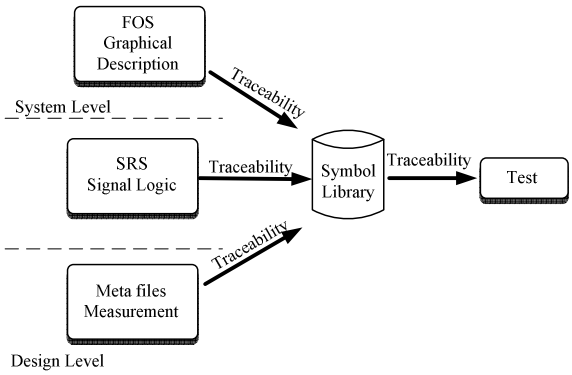

Figure 3. Interconnection of symbol data by a library

With the possibility to display symbols via SVG even before their implementation, the necessity of copying symbol pictures from the display graphics tool to the FOS is eliminated. Furthermore, modifications of symbols can be presented to the customer even in early development stages, without implementation on the design tool. The symbol description in SVG is created by an extensible style-sheet language transformation (XSLT) of the XML data. The XML schema provides the verification of the symbols as an independent artifact. Queries such as „do duplicate symbols exist", or „is the condition 'white text on white background' possible" or „is the new symbol schema-conform" are easily implemented in the database. These queries can serve as criteria for the creation of new test cases. The following section lays out the creation and simulated application of the solution prototype in the industrial environment.

V. EVALUATION IN THE INDUSTRIAL CONTEXT

For better evaluation of the current symbol requirements, an XML schema was created based on current documentation. The structuring and classification of the symbols is performed by assignment of defined conditions, as described in the FOS and in meta files. Figure 4 shows the implementation of the XML schema by example of the symbol „FIRE", which is presented by white writing on a red rectangle. Universal attributes, such as colors and fonts, are defined globally so that they need to be changed only in one place in the case of a modification. The content of the FOS and of the meta files are not consistently structured, so that an automatic query of the data is difficult to realize. For the implementation of the symbol „FIRE", the data was entered manually. Figures 5 and 6 show the XSLT of the prototype based on the SVG and the visualization of the symbol via SVG. It turned out that the current data processing of the development documents from which the XML schema has been created does not represent an adequate structure for a database transition or an XSLT to SVG (Fig. 5). For example, a rectangle is defined in the design tool by the coordinates of two corner points (compare „Rectangle" in Fig. 4), while the same element is defined in SVG by the coordinates of the top left corner, width and height (compare „rect" Fig. 5).

```
<Symbol>
    <Warning>
        <Title>
            <Name>FIRE</Name>
            <Pen_Color>white</Pen_Color>
            <Pen_ColorID>0</Pen_ColorID>
            <Set_Font>1</Set_Font>
            <Set_Linestyle>1</Set_Linestyle>
            <Coord>
                <x_Pos>2</x_Pos>
                <y_Pos>8</y_Pos>
            </Coord>
        </Title>
        <Background>
            <Fill_Color>red</Fill_Color>
            <Fill_ColorID>0</Fill_ColorID>
            <Type>
                <Rectangle>
                    <x1_Pos>0</x1_Pos>
                    <y1_Pos>0</y1_Pos>
                    <x2_Pos>150</x2_Pos>
                    <y2_Pos>50</y2_Pos>
                </Rectangle>
            </Type>
        </Background>
    </Warning>
</Symbol>
```

Figure 4. current data structure by example of „FIRE"

```
<!-- variables for background -->
<xsl:variable name="back_color" select="Symbol/Warning/Background/Fill_Color"/>
<xsl:variable name="back_x1_pos" select="Symbol/Warning/Background/Type/Rectangle/x1_Pos"/>
<xsl:variable name="back_y1_pos" select="Symbol/Warning/Background/Type/Rectangle/y1_Pos"/>
<xsl:variable name="back_x2_pos" select="Symbol/Warning/Background/Type/Rectangle/x2_Pos"/>
<xsl:variable name="back_y2_pos" select="Symbol/Warning/Background/Type/Rectangle/y2_Pos"/>

<!-- variables for text -->
<xsl:variable name="text_color" select="Symbol/Warning/Title/Pen_Color"/>
<xsl:variable name="text_x_pos" select="Symbol/Warning/Title/Coord/x_Pos"/>
<xsl:variable name="text_y_pos" select="Symbol/Warning/Title/Coord/y_Pos"/>
<xsl:variable name="text_size" select="Symbol/Warning/Title/Set_Linestyle"/>
<xsl:variable name="text_font" select="Symbol/Warning/Title/Set_Font"/>

<!-- svg output -->
<svg width="{$back_x2_pos}px" height="{$back_y2_pos}px" xmlns="http://www.w3.org/2000/svg">
<rect x="{$back_x1_pos}" y="{$back_y1_pos}" height="{$back_y2_pos - $back_y1_pos}"
width="{$back_x2_pos - $back_x1_pos}" style="fill:{$back_color}"/>
<text x="{$text_x_pos}" y="{$text_y_pos}" style="font-size:{$text_size}px; font-family:
{$text_font}; fill:{$text_color}">
<xsl:value-of select="Symbol/SysStat_Alarm_Symbols/Generic_Symbol/Warning/Title/Name"/>
</text>
</svg>
```

FIRE

Figure 5. XSLT based on SVG by example of „FIRE"

In long-living products, dependence on tools always comes with the risk of obsolescence. For this reason, the schema is currently adjusted to the standardized SVG symbol description in order to provide wider and more flexible tool compatibility for data processing. As the schema is still under development, the process sequence of a symbol modification was simulated with an idealized implementation concept. Starting point of the simulation is a graphical symbol modification which is included into the symbol library by a specification engineer. The symbol can now be displayed in a GUI simulation and used in a presentation for internal or external customers by using SVG format. This prevents potential misunderstandings or misinterpretations, which increases the efficiency of the modification process. The symbol pictures generated with SVG can also be implemented into the FOS. All documents affected by the modification (requirement and testing) can be displayed automatically due to their linking within the library; the error-prone manual selection can be omitted.

 ISBN: 978-1-61208-184-7

SQL queries are used to eliminate redundancy or other interference with existing symbols. These queries serve to validate the modification and create new test cases.

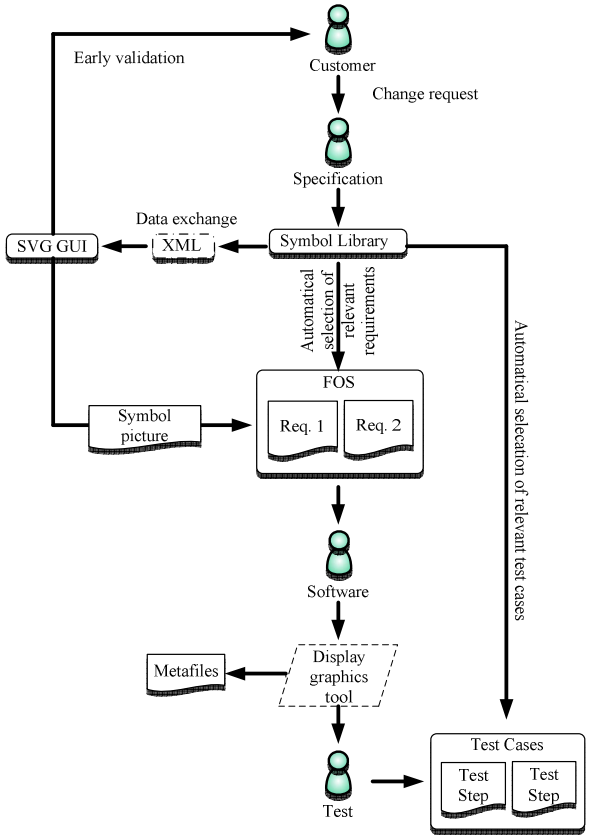

Figure 6. Simulated modification process based on the implementation concept

This implementation concept offers benefits for the specification and test stakeholders by partial automation of several process sections. The simulation of the implementation concept (Fig. 6) makes the actual process of modification more efficient by eliminating the feedback loop (Fig. 2) and automatically selecting all relevant documents. This way, the process of modification is more resistant to errors that may occur in manual research. The whole process is compliant to the „V-Model XT" standard. The resulting benefits for testing offers potential to develop and justify new symbol test cases derived from the database queries. Some further advantages are higher traceability of requirements and test cases, transparency and higher quality of symbol requirements, and improved verification and modification of symbols due to the formal definitions.

VI. CONCLUSION AND PROSPECTS

The implementation concept offers advantages for both requirements and tests of MFDs. Next the adaptation of XML schema is planned providing compliance to SVG. After this, a selection of MFD symbols is used to create a prototype for the implementation of real modification processes, in order to measure the results. The prototypes will be used after evaluation in the upcoming transfer of Supplier Software. Since the concept only provides a completion of symbol information of current specifications a subsequent goal will be the formalization of existing FOS and test cases to facilitate the exchange of their data with the library. This way the modification process from requirement up to testing can be (partially) automated. Developing a configuration management tool for the library is also required. This implementation concept intends, by example of the MFD, to expand current methods of reverse engineering to the levels of requirements and tests.

REFERENCES

[1] K. Bender, Embedded Systems - qualitätsorientierte Entwicklung: Qualitätssicherung bei Embedded Software: Springer Berlin Heidelberg; Auflage: 1, 2005.

[2] W. F. Daenzer and F. Huber, Systems Engineering: Methodik und Praxis, 11th ed.: Zürich Verl. Industrielle Organisation, 2002.

[3] T. Weilkiens, Systems Engineering mit SysML / UML. Modellierung, Analyse, Design: Dpunkt Verlag; Auflage: 1., 2006.

[4] M. Broy, V. Esperstedt, F. Houdek, K. Pohl, and H. Wußmann, "Leitfaden für modellbasiertes Requirements-Engineering und -Management softwareintensiver Eingebetteter Systeme — REMsES —," ed. Essen: REMsES-Konsortium, 2009.

[5] T. Wien, E. Carlson, T. Stålhane, and F. Reichenbach, "Reducing development costs in industrial safety projects with CESAR," Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference, pp. 1-4, 13-16 Sept. 2010.

[6] M. Adedjouma, H. Dubois, and F. Terrier, "Requirements Exchange: From Specification Documents to Models," Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference, pp. 350 - 354, 27-29 April 2011.

[7] D. Kretz, J. Militzer, T. Neumann, and T. Teich, "Developing an integrated solution for generative process planning based on ISO standard 10303 " Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference, pp. 61 - 65, 27-29 May 2011.

[8] H.-J. Choi and S. A. Fahmi, "Software Reverse Engineering to Requirements," Convergence Information Technology, 2007. International Conference, pp. 2199 - 2204, 21-23 Nov. 2007.

[9] E. J. Chikofsky and J. H. Cross, II "Reverse engineering and design recovery: a taxonomy " Software, IEEE vol. 7, pp. 13 - 17, Jan. 1990.

[10] J. Knodel, R. Koschke, and T. Mende, "Reuse in Reverse Engineering," Fraunhofer IESE Kaiserslautern, 2006.

[11] G. Canfora and M. Di Penta, "New Frontiers of Reverse Engineering " Future of Software Engineering, 2007. FOSE '07 pp. 326 - 341, 23-25 May 2007.

[12] S. Li and H. Liu, "COM-based symbol library extension for mine mapping," Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004 IEEE International vol. 6, pp. 4150 - 4152, 20-24 Sept. 2004

 ISBN: 978-1-61208-184-7