THE CONCEPT OF FUNCTIONS AND INFORMATION
CONVERSION IN SOFTWARE - DESIGN METHOD
ADAPTATION IN AN INDUSTRIAL CONTEXT

M. Weigt

Keywords: method, function, software, product data, automation

1. Introduction
Solving engineering design problems aims at creating technical systems to perform specified tasks. Re-use of existing design solutions can increase engineering design process efficiency and can contribute to technological maturity of these design solutions. Re-use requires identification of solutions and an evaluation of their applicability and adaptability with respect to given tasks. In this context, the concept of functions is essential to methodical engineering design procedure, since functions define intended purpose in a solution-neutral way, and the approach provides increased understanding of design solutions [Pahl et al 2005].

In innovative technical systems, electronic solutions in combination with software increasingly supplement mechanical solutions. Therefore, software component re-use offers considerable potential for optimisation of engineering design processes, and of products. Prerequisite for efficient re-use is availability of electronic product data. In the case of software components, this concerns for example specifications of intended working environments, and stability and performance aspects. Especially, comprehensive description of software functions is required, since these serve as principal criteria for task-oriented selection. Neither methodical nor IT support is currently sufficient for re-use or distribution by means of electronic marketplaces of software components.

To apply the concept of functions to software components, adaptation of this method is required to consider the nature of software as an abstract logical system. To this end, this paper applies an approach to design method development focusing information conversion to adapt the concept of functions to the specifics of software components. A software component repository prototypically implements the resulting method for functional specification. The industrial context for method adaptation and prototypical implementation was the field of automation technology, but the results are applicable in other domains as well.

2. Methodology

2.1 Analysis of method development from a process perspective
A *method* is a set of instructions, whose execution under given conditions sufficiently ensures the achievement of an intended objective [Mueller 1990]. A *process* is a set of activities that uses resources to transform inputs into outputs [ISO 9000:2000]. According to this, a method is a description of a set of activities, i.e. an abstraction of a process. Its purpose is to provide support for operators in processes. At the same time, a method is an (intangible) product, which is the result of a (method development) process. In this context, the term *meta-method* refers to methodical support of method development processes, i.e. a meta-method is an abstraction of a method development process.
This analysis results in the distinction of three different levels of abstraction in the context of method development. Their distinction facilitates the identification of conditions that are constitutive for method development, adaptation, implementation, and application (Figure 1).

At application level, operators face concrete tasks, e.g. the development of one concrete product. Such tasks constitute concrete application situations. Methods are supposed to be invariant for a certain class of tasks [Mueller 1990]. For this reason, method development usually considers generic application situations, e.g. product development tasks per se. Such generic application situations instantiate abstract problem situations.

Methodical support at application level is required to work twofold. On the one hand, methodical support aims at individuals performing operative problem solving activities within operative processes. On the other, it is required to support process management activities. Process management often employs decision gates, i.e. generic descriptions of information to which the results of operative activities are compared for the purpose of decision-making [ISO/TS 16949].

2.2 Framework for process-oriented method development based on information conversion

In this context, a process-oriented approach to method development is proposed, which focuses on concrete information at application level, on information properties at object-level, and on abstract information conversion aspects at meta-level. These information conversion aspects are organisational, technological and psychological aspects of information conversion. They are referred to as fluency, form and content, since they are interrelated mainly to communicating, representing and processing of information, respectively.

The approach applied in this paper structures the process of method development, adaptation and implementation according to the generic information conversing activities analysis, acquisition, consolidation, and definition. In the stage of method development, the principal focus of these activities is on information conversion. In the stage of method adaptation and implementation, it is on resources, shifting from qualitative aspects (resource capabilities) to quantitative aspects (resource capacity) (Figure 2).

The resulting meta-methodical framework proposes an iterative procedure for method development, consisting of a sequence of principal activities applied to information conversion, with the focus

---

3 This framework and its application are discussed in more detail in [Weigt 2005].
METHODS AND TOOLS IN DESIGN PRACTICE

3. Functional specification of software components

3.1 Problem analysis

Software is an algorithm formulated by measures of programming. It is therefore a specific type of method, whose supportive properties result from interaction with hardware and generally other software within a working environment. The automation technology applications focused in this paper are usable in variable working environments (personal computers or programmable logic controllers), as long as these meet specific standards. The approach does not consider embedded applications. Software is not a physical product, but an abstract logical system. It implements logical statements of the form \( \{P\} \langle S \rangle \{Q\} \): If a precondition \( P \) is true, by execution of the program \( S \) the postcondition \( Q \) becomes true. Physical effects and laws are only relevant for software components as environmental conditions. Analysis is not the mathematical foundation of software development, but instead combinatorics, mathematical logic and algebra.

A function is defined as the desired input-output relationship of a definite and limited system, whose purpose is to perform a task [Pahl et al 2005]. Since software is not a physical system, physical effects and corresponding mathematical interrelations do not provide appropriate foundation for its functional specification. In a working principle-oriented perspective, inputs and outputs of software applications are data processed by means of an algorithm. A functional perspective onto technical systems requires the interpretation of this data in the context of the given task, providing meaning for the data. Therefore, against the background of functional specification, software can be considered an

Figure 2. Meta-methodical framework

Focusing on information properties leads to an increased transparency of working principles of methods. This facilitates assessment of applicability of existing methods and their adaptation. It also improves method usability and acceptance, and effectiveness and efficiency of their practical application. The proposed meta-methodical framework can therefore serve as an appropriate foundation for adapting the concept on functions to software components.
Overall functions of software applications in automation technology result from the interconnection of so-called function blocks managed in libraries. These function blocks provide basic building blocks for control technologies [IEC 61131-3]. The resulting software is configurable, i.e. setting of certain parameters influences the functional behaviour of the building blocks and the entire software. Configurability specifically includes activation and de-activation of specific sub-functions.

The concept of working principles is well established in engineering design methodology. Principle properties of physical systems result from physical principles in combination with form design features [Pahl et al 2005]. Principle properties of software result from types of algorithms and types of data structures processed. Their combination provides a conceptual equivalent to the working principle of physical systems. Specification of function blocks according to [IEC 61131-3] is independent from tasks. Inputs and outputs state parameter names instead of what these inputs and outputs represent in the context of tasks. For this reason, these function blocks are more similar to the concept of working principles than to the concept of functions in engineering design methodology.

3.2 Method for the functional specification of software components

Consideration of fluency aspects of information conversion is not required in the context of this paper. It concerns a sub-process with specific process interfaces, which do not impose relevant constraints for its integration into the greater organisational and operational structure of the company.

At the cooperating company, the principal input information for functional specification is source code, including varying degree of textual commentary. Instruction list (IL) [IEC 61131-3] is the programming language used. IL is a procedural language similar to Assembler. Readability of IL code for human users is limited, especially when optimised for time-critical applications. This imposes a form-related corporate constraint to method development. Insufficient transparency is a form-related input information property that is important in the context of this paper. Increasing transparency requires input information consolidation with respect to form. Constructing program flow charts can achieve this. This graphical-textual representation increases readability and understanding of structure and purpose of the source code (Figure 3).

The following content-related task-specific constraints are essential for the adaptation of the concept of functions to software components:
- Software is not a physical system.
- No qualitative limitations exist to information conversion in software.
- The task of functional analysis and specification is typically of high complexity.
- The functional behaviour of software is highly configurable.

**Figure 3. Excerpt from IL code and corresponding program flow chart**

The following content-related task-specific constraints are essential for the adaptation of the concept of functions to software components:
Since Software is not a physical system, neither physical effects nor form design features are directly relevant for the functional behaviour of software components. Generic properties of inputs and outputs, which are influenced by means of the information conversion within the software, cannot be meaningfully provided independently from the context of the task. Therefore, instead of the context-independent concepts of physical effect-oriented special functions [Roth 2000] or input/output property-oriented generally valid functions [Pahl et al 2005], task-specific functions [Pahl et al 2005] can serve as a foundation for the functional specification of software components.

Task specific functions derive directly from the task. Their problem-oriented formulation uses technical common speech. This provides the benefit of not being limited to a specific set of technical verbs and/or physical terms. Technical common speech can therefore appropriately specify information conversion in software, which is subject to no qualitative limitations.

The concept of functions in engineering design methodology proposes hierarchical structuring as a means of making manageable complexity. To manage the high complexity of analysis and specification tasks, hierarchical structuring is an appropriate means in the case of software components as well. However, resource-oriented optimisation measures may lead to discrepancies between concrete code structure and abstract function structure.

Software components typically not only fulfil a set function. Their function can be highly configurable, and the software can advise about its functional behaviour. This makes advisable to distinguish between configuration-related control flow and processing-related data flow when defining the relations between the elements of a software function structure.

In the context of time-related synchronisation of program execution in distributed working environments, [IEC 61499-1] distinguishes between data flow and event flow. This standard provides a graphical notation that depicts function blocks with a separate execution control “head” and corresponding event inputs and outputs. Even though this event flow does not relate to functional configurability, this type of graphical representation can be suitably adapted to distinguish between configuration-related control flow and processing-related data flow in a software function structure (Figure 4).

Sub-functions are elements of a software function structure. Their inputs and outputs are data inputs (DI), data outputs (DO), control inputs (CI) and control outputs (CO). DI and DO constitute the data flow. It carries data that is algorithmically processed, e.g. process parameters. Inputs and outputs that are essential for functional specification are usually part of the data flow. Their identification enables functional specification based on their processing (e.g. “suppress low control deviations”) in addition to task-oriented functional specification (e.g. “stabilize controller”). The control flow supports structured specification of functional configurability. CI directly influence the functional behaviour of the software, while CO advise about it. Relations of the software function structure interconnect the inputs and outputs of overall function (OF) and sub-functions (SF), and of sub-functions among each other. Shell relations link sub-functions to the overall function. Core relations interconnect outputs and inputs of sub-functions.
The application context provides direction to determine the adequate degree of resolution of software function structures. In the case of development of new software applications, adequate degree of resolution depends on novelty of the task and on existence of solutions available for re-use. In the case of functional analysis and specification of existing solutions, the adequate degree of resolution depends on the degree of code optimisation: Resource-oriented boundary conditions motivate optimisation of time-critical applications, not the task per se. For this reason, with increasing degree of resolution, discrepancies arise between (resource-oriented) source code and (task-oriented) function structure. Beyond a certain degree of resolution, these discrepancies prevent re-use or adaptation for similar tasks with justified effort. Determination of this degree of resolution requires individual evaluation. As an example for a suggested degree of resolution, a function structure of a PID loop controller software is depicted in Figure 5.

![Figure 5. Function structure of a PID loop controller software](image)

This function structure facilitates a structured functional specification, with significantly reduced complexity of sub-functions as compared to the overall function. This function structure increases understanding of mode of operation and behaviour of the software. For example, interrelations between reaching higher and lower limits of the manipulated value and the integral action become transparent. Specifications of the respective sub-functions indicate halting of the integral action, and state the concrete conditions. Thus, this structured specification facilitates evaluation of applicability and adaptability of the software component to a given task.

Depending on software component complexity, their functional analysis and specification can require considerable effort. However, methodical development of new applications implicitly produces significant part of the necessary information: Task clarification includes a definition of the overall function. Specification of sub-functions and their interconnections reduces complexity of the development task, and is required for distributed development (function structure). Potential procedure within a program is defined (flow chart), as well as the program interfaces (inputs and outputs). Therefore, integration of the proposed method into methodical software development processes can significantly reduce required effort for functional analysis and specification.

### 4. Prototypical implementation

The proposed method for functional specification of software components is part of the prototypical software component repository Software Product Data Browser (SoftPDB). Basis of this web-based application is a hierarchical structure of software product characteristics. The hierarchy results from analysis of the product environment in the corporate and application context, throughout the product life phases, and from abstraction to principle and functional aspects. Among others, the hierarchy
includes descriptions of algorithms and data structures, specifications of intended working environments (hardware and other software), organisational aspects of software development and distribution, and technical conditions specific to the different product life phases. To identify potential solutions, elements and relations of function structures serve as main search criteria. With regard to potential ambiguities of task-specific function formulation using technical common speech, the repository features a semantic editor based on the WordNet lexical database [Fellbaum 1998]. Using this editor, users can provide definitions for terms used in designations and descriptions of overall functions and sub-functions, and their respective inputs and outputs (Figure 6).

![Figure 6. Specification of sub-functions of a PID loop controller using the semantic editor](image)

The semantic editor also enables the user to provide definitions for terms used in searches. Queries to the repository then do not compare terms, but definitions only. The semantic editor therefore facilitates effective use of synonyms and reduces the number of unsuitable matches. Following identification of potential solutions, comparing their properties to requirements of the task allows evaluating their applicability. This particularly concerns best-case and worst-case software execution times in combination with different CPU types, determined by means of code analysis and CPU benchmarking.

5. Summary and conclusion

Re-use of design solutions can increase engineering design process efficiency and can contribute to the solutions’ technological maturity. In the course of technical advancement, the importance of software components in technical systems increases in comparison to mechanical and electronic hardware. Methodical and IT support for standardised structured description of software products are currently not sufficient for internal re-use or electronic marketplaces. This paper presents a web-based software component repository that can serve as an electronic data sheet for software components. As such, it can contribute to the electronic generation and continuous transfer of product data. The underlying hierarchy of software product characteristics is life cycle-
oriented and follows established methods of systematisation and abstraction from engineering design methodology. The repository can therefore serve as a comprehensive foundation for structured documentation of software, its functions, working principles, and other particularities throughout the product life. Specifically, the hierarchy of software product characteristics can structure and guide the compilation of requirements lists for software development.

To facilitate identification of solutions and evaluation of their applicability and adaptability with respect to given tasks, the repository employs the concept of task-specific functions. This concept from engineering design methodology has been adapted to consider specifics of software as a logical system. A process-oriented approach to method development with a focus on information conversion provided support for this method adaptation. The resulting method for functional specification facilitates the structured analysis and description of software components, by making transparent the underpinning logic and making manageable the complexity of the specification task.

The method for functional specification and the software component repository are a result from cooperation with industry in the context of automation technology. However, application in other domains is possible, as long as the relevant constraints are sufficiently similar.

Acknowledgement
This paper includes results of the research project “Functional Specification in Automation Technology” (Transfer Unit 48). The project was carried out in cooperation with Siemens AG Automation and Drives and was funded by the German Research Foundation (DFG).

References

Dipl.-Ing. Markus Weigt
Universität Karlsruhe (TH)
Institute for Information Management in Engineering (IMI)
(formerly: Institute for Applied Computer Science in Mechanical Engineering (RPK))
76128 Karlsruhe, Germany
Tel.: +49 721 608-7958
Fax.: +49 721 661138
Email: markus.weigt@imi.uni-karlsruhe.de
URL: http://www.imi.uni-karlsruhe.de