

Review of “Specification and Transformation of
Programs” by Helmut Partsch, Springer-Verlag,
1990

Fritz Henglein
DIKU
University of Copenhagen
Universitetsparken 1
2100 Copenhagen Ø
Denmark
Internet: henglein@diku.dk

May 25, 1991

The aim of software engineering is the improvement of quality and cost of software by approaching the software development process in a principled, methodical fashion.

In transformational programming software is developed by applying transformation steps from a fixed or extensible base of provably correct rules to a formal specification until a correct and efficient target program is derived. This presupposes an initial stage in which a consistent and complete formal specification is engineered from informal requirements that are often incomplete, changing and possibly even inconsistent.

The book under review expounds the CIP (computer-aided, intuition-guided programming) approach to formal software development, addressing some of the technical issues, but not concerning itself with managerial aspects. It reflects the experience gained by the author and his (ex-)colleagues in the ongoing Dutch STOP project and in the CIP project conducted at the Technical University of Munich from the mid-Seventies until the mid-Eighties.

The book addresses the specificational and transformational issues of program development within the CIP approach. As such it complements the semantics-oriented descriptions of the language CIP-L (Springer-Verlag,

1985, Lecture Notes in Computer Science, Vol. 183) and the transformation system CIP-S (Springer-Verlag, 1987, Lecture Notes in Computer Science, Vol. 292). It can also be considered a successor to Bauer and Wössner's earlier book ("Algorithmic Language and Program Development", Springer-Verlag, 1982) as it expands on the transformational aspects of development of programs: it treats examples extensively, emphasizes the transformation process, and provides exercises at the end of chapters.

The book uses CIP-L, a wide-spectrum language based on (algebraic) abstract data types, as a linguistic vehicle for the core material. In the CIP approach a specification is transformed through several linguistic layers of CIP-L: from a descriptive specification (with non-operational set-theoretic and logical dictions) via an applicative specification (an operational functional program with recursion and abstract data types — abstract in the sense that they do not have to have an operational implementation at that point) to a procedural system-oriented program (with assignment, sequentialization constructs, and machine-oriented data types such as pointers and arrays presented as abstract data types). A (human) developer guides the transformational development based on his or her intuition. The clerical tasks of applying transformations, keeping track of a development and if necessary replaying it is performed by an automated system.

Chapter 1 introduces the main issues in program specification and transformation and gives an overview of the material, which is presented following the main conceptual steps of transformational program development. Chapter 2 is a detour into requirements engineering that may be skipped. Chapters 3–8 contain the core: the specification language and problem formalization (Chapter 3); basic transformation techniques and their theoretical foundations (Chapter 4); transformation of descriptive specifications into applicative ones (Chapter 5) and their modification (Chapter 6); transformation to procedural programs (Chapter 7) and implementation of abstract data types (Chapter 8). The final chapter, Chapter 9, contains four major example developments that demonstrate the use of the whole transformation process.

The book has been carefully structured and organized: the material is arranged logically along the linguistic levels of CIP-L; several running examples are used to demonstrate the individual transformation techniques; the indexes are good and helpful (so much so that they should have been listed in the table of contents); exercises conclude all core chapters. Despite its careful organization the book is not always easy reading: the exposition is at times rather dry and weighty; some examples carry so much detail that

it is difficult to follow what their point is; and some transformation rules are difficult to understand because of being stated needlessly general (e.g. rule “preservation of context information”, p. 207), complex side conditions (e.g. rule “looping-preventing folding”, p. 209), unclear identifier scoping, and complicated renaming and name clash conditions. The chapter on requirements engineering (Chapter 2) contains a line-up of requirements formalisms, but formalisms such as VDM or Z, DEVA, Extended ML, Object-Oriented Analysis *etc.* are noticeably absent. Similarly, in Section 1.5 a comparison to other programming methodologies such as object-oriented design, rapid prototyping, exploratory software development and others are missing (the treatment of logic programming and functional programming as programming *methodologies* in this section is somewhat peculiar). The transformation steps in the introductory example of Section 1.6 are not as intuitive and insightful as one might hope for, partly because of lack of motivation, partly because of the involved notation (including a minor error in the definition of predicate `nconf`, p. 16).

A few technical and terminological errors lurk in the core chapters. To wit, the use of the term “undecidable” (pp. 109, 111) is incorrect: neither Fermat’s nor Goldbach’s conjecture are (recursively) undecidable. Their answer is simply unknown. The specification of graph reachability (p. 223) is wrong since it asks for any set that is closed under taking its neighbors. In fact, reachability asks for the smallest such set. In exercise 3.4-11 (p. 146) a Hamiltonian path is erroneously called a Eulerian path. In several rules (e.g., pp. 169, 174, 207) a binary relation is required to be an equivalence relation of a transformation, but this is not sufficient for correctness: it has to be a congruence relation. There are some more technical problems with transformation rules, which is somewhat unfortunate given the emphasis on formal correctness. Overall, however, these are minor mistakes that do not affect the cohesion of the book.

The semantics of CIP-L is, in an intuitive sense, “bottom-up”: lower-level constructs such as procedures, assignment, while-loops, etc., are defined by their transformation to purely functional programs. Contrary to CIP philosophy I find this indirect and nonoperational semantics less conspicuous than a direct semantics, especially since it cuts out the very aspects that drive a transformational development to a large degree: operational (efficiency) concerns. (This is not really a problem since the notation on the imperative level is conventional and thus suggestive of its operational properties.) The syntactic locality of transformations combined with this indirect semantics affect how easy it is to understand what a program does

and, more importantly, how it does it. Since in a highly interactive programming methodology clarity and comprehensibility to the developer are most important, this may hinder ready practical application of the whole methodology.

The reliance on user guidance results in a large amount of detail that must be maintained by a developer. It not only makes developments potentially tedious, it also slows down the development process and forfeits the chance for early feedback on design decisions. In this sense the CIP approach is strikingly close to the conventional waterfall model of software development and one would expect that it is prone to inherit many of its problems in practice. Given the great responsibility for development the book does not provide enough methodological guidance for the developer. Generally, the control structure aspects are much more thoroughly treated than data structure design.

The prime emphasis in formal specification and transformation is on correctness: the transformations are to preserve the semantics of the specification; they may and should only concretize the design freedom available in the specification (expressing design freedom – referred to as liberality in the book – is a critical criterion for specification languages), but may not change the behavior. The driving secondary concern behind transformation of executable specification is efficiency: improvement of time and memory use and possibly other more systems-dependent resources (communication, registers, utilization of available hardware and software environment). Whereas correctness is explicitly captured and represented in the transformation rules with a stringent requirement that they be formally correct there is no formal support for the systems developer when it comes to improving the efficiency of an operational specification. While it is clearly too much to expect a usable, completely formal theory for this aspect, I would have liked to find more methodological support – structuring the transformations along efficiency concerns or evaluating transformation strategies according to their possible improvement of programs.

A related concern is that whereas the specification language is high-level the transformation process is low-level and relies on extensive user interaction. As long as no structured theories of program development emerge (similar to individual disciplines within mathematics) transformational programming may be a long way from improving the cost situation in software development.

To my knowledge, this book contains the most comprehensive collection of transformation examples and techniques in a uniform framework, not

only from within the CIP project, but from much of the transformational programming community at large. To anybody interested in learning about transformational programming, studying this book is hard to avoid. It is addressed to 3d or 4th year students; this may be too early, not because of the technical background required, but because of the students' likely lack of experience in appreciating the goals and difficulties of software engineering and evaluating the book against these. The book is also a valuable basis for professionals since its examples should permit evaluation of its software development approach in realistic settings.