proceedings June 16-19, 1999

the 11th international conference on software engineering & knowledge engineering

printed by Knowledge Systems Institute Graduate School
PROCEEDINGS

SEKE’99
Eleventh International Conference on
Software Engineering and Knowledge Engineering

Sponsored by
Branden University, USA
Fraunhofer Institute for Experimental Software Engineering, Germany
Knowledge Systems Institute, USA
University of Kaiserslautern, Germany

Industrial Sponsors
Daimler Chrysler AG
Deutsche Telekom
Ericsson Finland
IBM Deutschland Informationssysteme
Insiders Teknon Wissensbasierte Systeme
Q-LABS
sd&m
Softlab GmbH
TECINNO GmbH

Technical Program, June 16-19, 1999
Kaiserslautern, Germany
SEKE’99 Conference Organization

General Chair
Dieter Rombach, University of Kaiserslautern and FhG IESE

Program Chair
Guenther Ruhe, FhG IESE

Steering Committee
Vic Basili, University of Maryland, USA
Bruce Buchanan, University of Pittsburgh, USA
Shi-Kuo Chang, University of Pittsburgh and Knowledge Systems Institute, USA
C. V. Ramamoorthy, University of California, USA

Tutorials, Panels and Workshops Chair
Ahmed Rafea, Cairo University

Workshop on Learning Software Organizations Chair
Frank Bomarius, FhG IESE

Exhibition Chair
Bidjan Tschatschian, German Research Center for Artificial Intelligence

Local Arrangements Chair
Barbara Dellen, University of Kaiserslautern

Finance Co-chairs
Judy Pan, Knowledge Systems Institute, USA
Kornelia Streb, IESE, Germany

Program Committee

Perry Alexander, USA
Paul Attie, USA
Paul Bailes, Australia
Alfs Berziss, USA
Pere Botella, Spain
Kai Chang, USA
Lewis Chau, China
William Cheng-Chung Chu, Taiwan
Reidar Conradi, Norway
Chrysanthos Dellarocas, USA
Yi Deng, USA
Andreas Dengel, Germany
Jrgen Ebert, Germany
Kokichi Futatsugi, Japan
Robert Glass, USA
Claude Godart, France
Christiane Gresse von Wangenheim, Brazil
W. David Hurley, USA
Natalia Juristo, Spain
Srdjan Kovacevic, USA
Bernd Krdmer, Germany
Xiaoqing Liu, USA
Michael Lowry, USA
Chung Horng Lung, Canada
Tim Menzies, USA
Roland Mittermeir, Austria
Sandor Morasca, Italy
Ana M. Moreno Sanchez-Capuchino, Spain
Markku Oivo, Finland
Ahmed Rafea, Egypt
Michael Richter, Germany
Steve Roach, USA
Dave Stuart Robertson, United Kingdom
Mansur Samadzadeh, USA
Eugene Santos, USA
Dorothy Sedlff, USA
Wie-Tek Tsai, USA
Jason Wang, USA
Naoki Yoenzaki, Japan
Marvin Zelkowitz, USA
Kang Zhang, Australia
An Approach to an Integrated Intelligent CASE Tool for Automatic Software Design

Hui Chen Hideki Takano Hassan Abolhassani Zenya Koono B. H. Far
Department of Information and Computer Sciences, Saitama University
255 Shimo-okubo, Urawa 338, Saitama, Japan
chen@cit.ics.saitama-u.ac.jp

Abstract

This paper introduces system design of an Integrated Intelligent CASE tool, that may be applicable from top level (data flow diagram) design down to coding. This tool consists of a Data Flow Diagram CASE (DFD CASE) tool and a Structured chart PAD CASE tool to integrate a data flow diagram and a structured chart for each stage of design. Design of a hierarchical detailing is first expressed on algorithm using DFD, and then the control is designed and expressed by PAD. By combined operation between DFD CASE tool and PAD CASE tool, design knowledge of a hierarchical detailing is shared to both DFD design and PAD design. Design knowledge is automatically acquired from respective documents and stored in the knowledge base. The acquired design knowledge may be reused in following designs. By reusing it, a similar software system may be designed automatically. This tool features universality, an essentially zero start-up cost for automatic design, and a substantial increase in software productivity after enough experiences have been accumulated.

1. Introduction

The software industry has been facing increasing demand for software, and automatic design has been regarded as the final solution[8]. Many automatic systems have been developed, but automatic design technology has not yet been used widely in the industry. Software Creation Project aims at establishing a rational and scientific basis for enabling automatic design for any kind of software. Researches have been made in a bottom up manner, and the implementation has been made considering the cost effectiveness.

As the result of studies, Intelligent CASE tool [1, 2, 3] was developed, and presented at SEKE '98. It may be applied to structured programming phase of software development, and decreases man-hours during detailed design and coding less than 20% of human design after enough design experiences are accumulated. The principle is reuse of structured chart PAD (Problem Analysis Diagram)[4] in fragments from Software Engineering view point, and automatic design based on skill level design knowledge from Knowledge Engineering view point. Integrated Intelligent CASE (IICASE) tool is now being developed in the project following the bottom up research and development strategy. It extends the principle of Intelligent CASE tool to cover top level data flow design phase. This paper introduces IICASE tool.

The most frequently used method for system development is a combination of Structured Analysis[6] and Structured Design. Though it is fitted for EDP where files are standardized, but it is not fitted for software design in general. Not only it invites a gap between DFD phase and flow chart phase, but also the automation invites extra efforts of automating DeMarco's DFD[6] in addition to that for flow chart or structured chart. In this system, a different approach from conventional design work process of data flow design and flow chart design has been taken.

Software design is a hierarchical detailing process of human concept. The conventional way is to make DFD design first, then structured programming follows. In this paper, we propose to design both DFD and structured chart (PAD) in each of hierarchical detailing. Design of a hierarchical detailing is first made on algorithm using DFD, and then the control is designed and expressed by PAD. By combined operation between DFD CASE tool and PAD CASE tool, design knowledge of a hierarchical detailing is shared to both DFD design and PAD design. PAD design may be omitted in the case of high level design, and similarly DFD design may be omitted in the case of low level
design.

Design process knowledge is a hierarchical detailing relationship and the acquisition is made automatically on DFD and PAD. The design knowledge is stored in Knowledge Base and may be reused in following designs. As the system experiences designs, the degree of automatic detailing becomes larger. This decreases the initial cost of preparing design knowledge prior to design and makes it possible to apply II CASE tool to any kind of software design.

This II CASE tool decreases development cost during maintenance phase and enables precise and reliable design of a system. It is fitted for component level design where details of the design must be clearly kept and be well-readable always, and later coming people may work easily.

Section 2 explains basic principles of software design, Section 3 introduces systems design, and Section 4 is a discussion.

2. Software design

2.1. The structure of design

In any kinds of software design, there appears a hierarchical structure. In the top managerial level, it is called hierarchical decomposition of object. The design of software system comes next, starting with the specification. It is a hierarchical decomposition such as systems, departments, sections, and so on, and during the design information is detailed hierarchically.

Figure 1[7] shows design of a clock program, an example of a program level design. From the left to the right, the specification for clock is converted to DFD’s, to flow charts and finally to source codes. In the left part of the Figure 1, step by step design of data flows are shown in a top down manner. Data and functions are decomposed or detailed hierarchically as the design progresses. DFD shows the decomposition algorithm best. In the center part of this figure, flow charts are shown. Each box is the same as the corresponding function box in the corresponding DFD. The only difference from DFD’s is to show control of sequence of executing function boxes and decision boxes, not shown in DFD but appearing when necessary. The hierarchical detailing of both DFD and flow chart coincides with each other.

2.2. Standardized design process

Based on the principle of the hierarchical software design above, the standardized way of design process becomes possible. Figure 2 shows the standardized design process. It consists of a decomposition of a function (Func.) to several lower level functions (Func1., 2, 3), the algorithm is the best shown by DFD, and the structured chart (PAD) shows the control sequence using mostly the same function box representations in DFD. The structured chart (PAD) of this level are de-
talled and finally they are converted to code. When
thus standardized, the same set of documents (DFD
and PAD) may be used throughout design. This may
be applied from top system level, then each functional
unit, and finally down to each program module. This
shows that a design of a system has been reduced to a
standard work procedure. By this standardized work
process, the same automatic design may be applied
from the top level design down to coding.

3. An Integrated Intelligent CASE system

3.1. Architecture

The architecture of IICASE tool is shown in Figure 3. It is composed of combination of a DFD CASE
tool, a Structured chart PAD CASE tool, a knowledge
acquisition unit, Knowledge Base, an automatic design
unit and an integration unit to integrate a data flow
diagram and a structured chart for a design.

A designer draws a data flow diagram using the DFD
CASE tool, starting from a natural language concept.
When the design completes, the corresponding PAD is
drawn automatically from the DFD, and some neces-
sary control function, such as decision and repetition,
is added when necessary. The diagrams are sent to
the expert system and the design knowledge is stored
in Knowledge Base. The integration unit checks the
consistency of DFD and PAD.

If a designer draws some PAD using the PAD CASE
tool, the original sketch of function box(es) for the cor-
responding DFD is prepared and a designer can easily
complete the DFD by inserting data boxes.

For a new design, a designer draws the initial design
sketch using the graphic symbols on the DFD CASE
tool display, then it generates the corresponding PAD,
and the designer adds some control needed. When syn-
chronous automatic design starts, a symbol not yet de-
tailed is fed to the expert system, which details it using
the previously stored design rules. DFD and PAD are
detailed respectively to the next level. The correspon-
dence between detailing of PAD and DFD is checked
automatically.

By monitoring the progress of the automatic design
on the display, the designer checks the design, adds
new design rules, selects an appropriate design rule if
needed, and modifies the design. For a part newly de-
signed by the designer, the design knowledge may be
acquired automatically. In this way, the most detailed
design charts ending in source code is displayed on the
CASE tool.

For a normal operation, the system works with both
DFD and PAD CASE tool, but at a high level design
(such as for job flow) only DFD is used, and at the last
end of detailed design sometimes only PAD is used.
For human intervention during automatic design, hu-
man operations must be made without affecting an-
other CASE tool. For these purposes, the system can
select a CASE tool to be on-line or off-line. Other
controls are related to automatic design and automatic
knowledge acquisition and their control for modes and
operations.
3.2. Knowledge acquisition

This system features automatic acquisition of elementary design process knowledge, named a design rule, or fractions of documents corresponding to a small step of progress of design. The principle is all the same as that of PAD[2].

![Diagram of Knowledge Acquisition]

**Figure 4. Acquisition of design knowledge**

Figure 4 shows major three documents of the system. Functional Structure Diagram (FSD) on the top shows the structure as its name implies, for enabling an integrated operation of DFD and PAD designs. The middle left of the figure shows a pair of DFD, where a lozenge denotes data and a square denotes a function. A unit DFD (D1-F1-D2) as a parent is detailed to another DFD (D11-F11-D12-F18-D13) as the child. Such a parent-to-child(ren) relationship is called a DFD design rule, and the DFD design process knowledge is expressed by the relationship. For the automatic acquisition from DFD documents, a dotted arrow line shows the correspondence.

The bottom left of this figure shows a PAD[2, 3]. In this figure, a parent concept symbol connected to several children symbols (child1,..., child4) located to the right corresponds to a PAD design rule. The PAD design process knowledge is expressed by a parent-to-child(ren) relationship. A dotted arrow line route shows a tree walk program to acquire the PAD design rule automatically.

3.3. Automatic design

Figure 5 shows input, automatic design operation (reusing the design) and output. If a function name is specified as shown in the top left figure of FSD, the system searches the name F in FSD KB and details F automatically as shown in the corresponding right side of the figure.

Automatic designs of DFD and PAD are shown in the middle and the bottom of Figure 5. All automatic designs are performed by reusing the design rules. If any corrections are needed, the designer modifies the designs.

This system also features automatic coding. Similarly to preceding designs, it reuses previous results of coding, namely design rules from a natural language symbol (parent) to source code symbols (children) relationship. The reuse of the relation of a natural language to source code eliminates the complex knowledge required for coding, and makes the system independent of the programming language used.

![Diagram of Automatic Design]

**Figure 5. Automatic design**

4. Discussion

The most important evaluation criteria of this kind of tool is the improvement of software productivity. It was evaluated quantitatively by using Learning Effect Engineering[5] in Industrial Engineering, and Figure 6 shows the evaluation for the case of Intelligent CASE tool[2]. As the system experiences many designs and stores many design rules, designs may be performed with a higher degree of automation than before. In Figure 6a the curve starts from the initial zero, and rapidly clears over 80% but it tends to 100% very slowly. Based on it, Figure 6b shows man-hours for a design taking consideration of supplementing new designs. When the experience is over 10, man-hours is decreased to less than 1/4 of pure human design.
Intelligent CASE tool supports structured programming process, and so does Integrated Intelligent CASE tool. By introducing a standardized design procedure using both DFD and PAD at each hierarchical decomposition, a seamless automatic design becomes possible from early phase of design down to coding.

As DFD shows decomposition algorithm and PAD shows the control flow, most of PAD is automatically prepared and the additional work needed is only to add decision and repetition.

IIICASE tool has another mission to reveal further detail of human design. As the system provides both data and control information, it becomes possible to include 'automatic programming' in the detailed level, and to extend further to 'requirement analysis' using function structure diagram. Both researches have been being made. These evolulional progresses will be reported in the future.

5. Conclusion

This paper reported an Integrated Intelligent CASE tool, that enables automatic design applicable from top level DFD design down to coding. This tool is combined of a DFD CASE tool and a Structured chart PAD CASE tool to integrate a data flow diagram and a structured chart for a design. It features following;

- automatic design from high level data flow design down to coding, reusing design knowledge,
- design knowledge is acquired automatically from design documents of a hierarchical detailing to both DFD design and PAD design,
- applicable irrespectively of programming language used for coding,
- zero start-up cost for automatic design,
- substantial saving of design man-hours, and
- a substantial increase in software productivity.

From a Software Engineering viewpoint, this system is a reuse of design documents in fragments for flexibility, but it is better explained from a Knowledge Engineering viewpoint.

Acknowledgement

This system is based on studies in the Software Creation Project. The authors thank many students for their contributions. The authors wish to express their sincere gratitude to Hitachi Chubu Software Ltd. for supporting this project.

References


