Towards A CASE Tool for Jackson's JSP, JSD and Problem Frames

Nicholas Ourusoff
Teacher of the University, Department of Computer Science, University of Liverpool
65 Seamans Road
New London, NH 03257
USA
1-603-526-6195
nourusoff@yahoo.com

ABSTRACT

The development of a CASE tool that supports Jackson's methods – Problem Frames, JSD and JSP - is urged and justified. Jackson's methods are sound, and fundamental ideas about design provide unity to his thought; in short, his contributions deserve to be part of the software engineering curriculum. A CASE tool would help to achieve this end. Moreover, teachers need a CASE tool to teach software engineering effectively – a Jackson methods CASE tool would help demonstrate the benefits of sound design convincingly to students. Finally, a Jackson Methods CASE tool would further research.

1. Introduction

It has been argued (in [8]) that Jackson's methods and ideas have consistently been at the cutting edge of research and practice over more than three decades and deserve to be incorporated into the software engineering curriculum; that commercially successful CASE tools appear to sell methods rather than vice-versa; and that a CASE tool should therefore be developed for Jackson's methods and ideas.

This position paper again presents the case for developing a Jackson methods CASE tool, but this time justifies its development based on its usefulness to teachers and researchers. While several CASE tools could be developed, a rationale is given for developing a Jackson Methods CASE tool that covers the entire system development process - incorporating Problem Frames, JSD and JSP.

2. Jackson's methods and ideas

Jackson has contributed three methods to the field of software engineering: Jackson Structured Programming (JSP), a method for designing a class of simple programs; Jackson System Development (JSD), a method for specifying dynamic information systems; and Problem Frames, a method for analyzing software problems. A number of recurring fundamental ideas provide unity to Jackson's contributions to the field.

2.1. Soundness in software engineering method

We are skeptical whether large software systems can be proven to be correct, so, we won't equate soundness in software engineering with formal proof of correctness. Nor does soundness mean widely used or current "best practice". By sound, I mean that (i) a method's underlying conceptualization is elegant; (ii) that the method is logically persuasive; (iii) that the method conforms to Jackson's methodological principles1; and (iv) that the method works for its intended range of application. (Jackson's methodological principles, described in [1], pp. 368-370, are: (i) Easier decisions should be made before difficult decisions; (ii) Error-prone decisions should be deferred; (iii) Implicit decisions should be avoided; (iv) Error-prone decisions should be subjected at the earliest possible moment to confirmation or refutation; and (v) Whenever possible, decisions should be independent of each other (orthogonal).)

Both JSP and JSD work for their intended range of application; are logically persuasive because Jackson demonstrates in detail how they can be applied to a number of small but instructive problems; and conform to his methodological principles (see [8]). Below, I focus on the elegant conceptualization facet of the notion of soundness of a software engineering method.

2.1.1. Conceptualization in JSP

- The underlying conceptualization of JSP is that a program's structure may be *constructed* by *composing models* of its inputs and outputs using the control structures of structured programming.

2.1.2.Conceptualization in JSD

The underlying conceptualizations in JSD include:

- The structure of an information system is based on a model of its entities (real-world objects) and their actions (JSD is an object-based system development method).
- An information system is viewed as a network of communicating sequential processes.
- A JSD specification is in principle executable: Processes in JSD can be scheduled by exploiting program inversion.

2.1.3.Conceptualization in Problem Frames

The underlying conceptualizations of Problem Frames are:

- Software problems have a structure consisting of their *principle parts* and a *solution task* (see [8]).

The principle parts consist of a machine – what we wish the machine's behavior to be at its interface with the problem domains; the problem domains – facts about phenomena in the real world that are the context of the problem; and a requirement –what we wish to be true in the problem domain. The solution task is to build a machine that satisfies the requirement.

- A *discipline of description* is at the heart of information systems development.

Corresponding to the three parts of any software problem listed above, the developer must provide three descriptions: for specification, problem domain, and requirements. A new discipline of description is needed, based on the phenomenology of the application domains. The frame concern, based on requirement, specification and domain descriptions, is to use these descriptions to argue that the specification combined with the properties of domain phenomena entails the requirement. Precise designations, which serve to identify phenomena and are refutable, and definitions, formal extensions of designations, help in making a correctness argument.

- Problem frames provide a template with known solution methods for fitting problems.

If a problem fits a problem frame and a method exists to solve this class of problem, the developer can solve their problem correctly.

- Problem analysis relies on problem decomposition into sub-problems, followed by composition of the resulting machines.

Problem decomposition is based on a domain's type and properties. The resulting sub-problems are treated independently of other sub-problems – this is the basis of *effective separation of concerns*. Each sub-problem has its own machine (specification), problem domain(s), and requirement; each is a projection of the full problem.

Because Problem Frames has not been yet been validated through experience, we cannot say yet that it is sound – only that it is promising, because its concepts are elegant, its steps are logically and persuasively illustrated, and it conforms to Jackson's methodological principles.

2.2. Recurring ideas

JSD builds on ideas in JSP (program inversion, entities as a structuring principle, structure (tree) diagrams) and Problem Frames incorporates both: JSP is a method for solving the transformation problem frame, while JSD is a method for solving dynamic information problems. The unification of Jackson's methods should not be surprising: fundamental ideas recur in Jackson's thought, as shown in Table 1 below.

3. A Jackson methods CASE tool for teaching and research

One may object that a CASE tool for Problem Frames is premature – there are too many issues and ideas to work out. My answer is that we view this development as an ongoing research project. More importantly, teachers need a CASE tool that supports a promising software engineering method. It will help redress the present unsatisfactory state of affairs in academic software engineering.

3.1. Why a Jackson methods CASE tool is needed for teaching

The argument is:

1. Teachers need a CASE tool to teach program design, system development, and problem analysis.
2. A CASE tool for program design, information systems development and problem analysis does not exist.
3. Jackson's ideas and methods on program design, system development, and problem analysis deserve a place in the software engineering curriculum.
4. Therefore, a Jackson CASE tool should be developed: it will promote Jackson's ideas and methods in the software engineering curriculum and fill the need of teachers for a CASE tool.

We have made the case for 3. in [8]. Below, we argue for 1. and 2.

1. Teachers need a CASE tool to teach program design, system development, and problem analysis.

Programming is usually taught using a programming language to illustrate and motivate programming. Using a real programming language permits students to have the satisfaction of running programs and obtaining desired outputs. It also acquaints students with a programming language that is used in the real world and is relevant to a student's getting a job. Unfortunately, much of the focus in this approach to teaching an introductory course in programming must be on learning syntax and on debugging programs rather than on program design.

A more desirable approach to teaching programming is to focus on program design using pseudo-code. Proponents of this approach argue that there is much less emphasis on syntax and debugging, allowing one to focus on design. (See, for example, Shackelford, Russel. (1998). Introduction to Computing and Algorithms. Addison-Wesley.) Unfortunately, unless one can execute the pseudo code (for example, using an interpreter), students don't have the satisfaction of running their programs and seeing whether they produce the expected output. Without the feedback of a running program, the presence of errors goes undetected and detracts from the logical persuasiveness of a design.

The wish to run programs and see whether they produce the expected outputs is often misguided: obtaining the expected output does not mean that the program is correctly designed! [1] However, obtaining the expected output from a correctly designed program is both instructive and highly desirable. The same applies to system development and problem analysis – we need to show that correct results follow from sound design. In fact, we may take the following as a teaching *requirement*: Introductory software engineering courses need a CASE tool to show that correct output is a consequence of sound design.

2. *A CASE tool for program design, information systems design and problem analysis does not exist*

The typical introductory course in software engineering is likely to focus on traditional structured methods using Entity-Relationship and Data Flow modeling and/or object-oriented analysis and development using Rational Unified Process, UML and a CASE tool like Rational Rose (see [8]). But that's not what is needed! We need a Problem Frames approach, focusing on problem domain description, rather than on solution specification.

Nor are there any commercially successful CASE tools that support Jackson's methods. A proprietary JSP-COBOL Preprocessor developed by Jackson's consulting firm under his direction was used successfully in JSP and JSD projects for many years in the 70's and 80's, and there are some other proprietary tools based on JSD. The most promising CASE tool at present is the Jackson Workbench under development by Keyword Computer Services Limited (Contact: Jim Newport, Jim.Newport@BTInternet.com, +44 01494 870425. A free Web-based JSP tool, JSP-Editor is available (URL: http://www.isk.kth.se/kursinfo/6b4020/Diverse/JSPEditor ; Contact: Henrik Engstrom,University of Skovde, henrike@ida.his.se). It has been used to teach JSP (Contact: *rickduley@hotmail.com*).

Jackson Workbench has leveraged earlier JSP tools through reverse-engineering and supports JSP and JSD. I have described happily teaching introductory programming as software engineering using JSP and Jackson Workbench (in [7]). However, Jackson Workbench is under development as a secondary activity of a consulting firm. There is no definite plan for further development, nor does Jackson Workbench support Problem Frames.

Thus, the academic community must act itself to see that a Jackson Methods CASE tool for teaching and research is developed. We should explore obtaining active support of the research community in industry as well as collaboration with the developers of Jackson Workbench. The project needs to be under academic control, most likely as an Open Source Project or as one or more doctoral dissertations.

3.2. A Jackson methods CASE tool as a research project

Development of a CASE tool that focuses on Problem Frames can address research issues, such as:

1. Use of multiple formalisms

Which language should be used for what descriptions? Jackson suggests that a simple formal model - perhaps labeled state transition diagrams - should be used to interpret the specification in terms of the observable phenomena. First-order logic (FOL) is needed to express a set of designations and definitions as a basis for formulating a sound argument that the specification satisfies the requirement. Tree diagrams, class diagrams, and algorithmic pseudo-code may also be used.

Students find the task of developing state transition diagrams –thinking clearly about and distinguishing among events, states, and conditions – to be one of the difficult areas of software engineering. Developing a state transition diagramming tool that helps students reason about phenomena seems a worthwhile goal in this project.

2. Reasoning engine

A reasoning engine is needed to (i) derive a specification from the requirement ([2]); and to (ii) demonstrate that the specification together with the domain properties entails the requirement ([4]).

Inference engines are traditionally used in AI applications. Clearly, however, problem analysis involves reasoning about causal phenomena. Relations that hold true are called facts. Let's adopt Humean causality: A is said to cause B if B always succeeds A temporally (David Hume, *Enquiries concerning Human*

Understanding (1748)). If a system can infer that B does not follow A as expected, then a system can know that it is failing. This is part of the reliability concern ([4]).

3. Support for multiple views

The Problem Frames approach involves the decomposition of a problem into sub-problems and the composition of subproblem machines into a single runnable machine. Each sub-problem is an independent view, a projection of the whole problem. The CASE tool must support the integration of one view with another and the integration of the machine into the world based on shared phenomena.

Table 3 below lists features that may be present in a Jackson CASE tool.

3.4. Why a Jackson methods CASE tool should incorporate Problem Frames, JSD and JSP

One might be tempted to develop a CASE tool to support only Problem Frames, especially since Jackson Workbench is an existing CASE tool for JSP and JSD. Moreover, there are gaps and research issues in the Problem Frames approach, whereas JSP and JSD are well defined. Still, there are reasons for developing a CASE tool that encompasses not just Problem Frames, but all of Jackson's ideas and methods:

1. As described in section 2 above, the unity of Jackson's ideas argue for their inclusion in a single CASE tool. Moreover, since JSP is a sound method for the transformation problem frame and JSD is a sound method for the (dynamic) information display frame, it is natural to include these methods in any Jackson CASE tool that focuses on Problem Frames.
2. As described in section 3.1 above, teachers need a CASE tool in order to produce running code for program and system designs. We need to carry out the system development process from analysis to specification to implementation.
3. In [1], Jackson provides a method for deriving specifications from problem analysis. A CASE tool that encompasses the entire development process would permit further research and testing of this method.

Jackson Workbench is under development by a small consulting firm that uses JSP and JSD for much of its commercial activity. Development of Jackson Workbench is a secondary activity, contingent on the firm's profitability and the availability of (skilled human) resources. Moreover, I have cautioned that generally commercial CASE tools do not serve academic software engineering well: while they have influenced publishers and teachers to adopt the methods they embody under the guise of best practice, in fact their success is due to marketing: they are highly profitable and widely used - whether or not they are sound is really of secondary importance.

Development of a CASE tool for teaching and research should be done as a collaborative project by academics and researchers – perhaps as an Open Source project or as one or more doctoral dissertations. This does not preclude enlisting the support of sponsors from industry who have a strong research component – IBM, Sun, Microsoft, Oracle, and others. The developers of Jackson Workbench might also be persuaded to join in this collaboration.

4. Concluding remarks

I have argued for development of a CASE tool encapsulating Jackson's methods and ideas to be used for teaching and research. A CASE tool organized by academics and researchers would send a strong signal to publishers and vendors that the software engineering discipline is not for sale. It would provide a needed CASE tool for teachers of software engineering and further research. The project could become an important resource for teachers and researchers in software engineering by providing a Web site for discussion, case histories, and experience in teaching Problem Frames.

References

[1] Jackson, M.A. *System Development.* 1986. Prentice-Hall.

[2] Jackson, M.A. and Zave, Pamela; *Deriving Specifications from Requirements: An Example*; in *Proceedings of ICSE-17*; ACM Press, 1995.

[3] Jackson, M.A. *Software Requirements and Specification*. Addison-Wesley, 1996.

[4] Jackson, M.A. *Problem Frames*. Addison-Wesley Longman Publishing Co., Inc. , 2001.

[5] Jackson, M.A. *Aspects of System Description* in Programming Methodology, A McIver and C Morgan eds; (in publication).

[6] Newport, J. *JSD Version 2 And An Overview Of Developments In JSD.* IEEE Colloquium, *Jackson System Development (JSD) - The Original Object Oriented method?*, 1 February 1996

[7] Ourusoff, N. *Using Jackson Structured Programming (JSP) and Jackson Workbench to Teach Program Design.* In Proceedings of 2003 Informing Science and Information Technology Education, Pori, Finland, June 24-27, 2003. http://ecommerce.lebow.drexel.edu/eli/2003Proceedings/PaperPage106.html/

[8] Ourusoff, N. Reinvigorating the Software Engineering Curriculum with Jackson's Methods and Ideas. *ACM SIGCSE Bulletin, June 2004 (in press)*

Table 1: Recurring Ideas in Jackson's Methods

structure	Software engineering is concerned with design. and design is about structure, the relation of parts to the whole. In JSP, program structure reflects problem structure. In JSD, the system is structured in terms of abstract entities and actions, and then into a network of communicating processes. In Problem Frames, each elementary problem frame is structured into its principle parts and a solution task. Patterns are another facet of structure: JSP formulates rules and strategies for recognizing and processing various patterns in the inputs and outputs of transformation problems. Problem Frames are design patterns for problems.
modeling	The real world – not any formal paradigms - is the starting point for each of Jackson's methods. The simple programs of JSP, no less than the dynamic information problems of JSD, simulate the real world. Problem Frames focuses on descriptions of the application domain. (Jackson sometimes refers to these descriptions as "descriptive models", though generally, he uses model to refer to an analogical model.) Any problem involving dynamic information decomposes into sub-problems, one of which is an (analogical) model of the application domain.
composition/ decomposition	Effective separation of concerns is achieved by decomposition. In JSP, a program's structure is derived by composing its input and output models. Programs that appear to be beyond the range of JSP can often be decomposed into simple programs to which JSP can be applied. In JSD, a process resulting from a rough merge may be seen as a composition of the structures of the data streams being merged. In Problem Frames, problems are first decomposed into sub-problems - elementary problem frames – and analyzed separately; the resulting specifications are then composed – as parallel processes - into a final machine specification.

Table 2: Features supported in a Jackson methods CASE tool

diagram editors (context diagrams, problem diagrams, problem frames diagrams)	A principle of workpiece tools such as diagram editors is for the tool to do all of the drawing based on user feedback to prompts by the machine. (This principle is exploited beautifully in the Structure Editor of Jackson Workbench.) Each diagram must maintain the appropriate diagram symbols and annotations, as well as domain and interface markings.
description workbench	FOL and other languages of description will be supported. Considerable effort should be put into an interactive labeled state-machine diagram editor with markings indicating state-machine symbols, events and actions, and transitions.
reasoning engine	An FOL reasoning engine is needed to bridge the gap between requirement and specification using reasoning based on environment properties that can be relied on independently of the machine. A reasoning engine is needed to show that the specification combined with domain properties entails the requirement. To address reliability and other concerns, a reasoning engine is needed to reason about causal phenomena.
decomposition/ composition workbench	The CASE tool will prompt the user to decompose and fit a problem into sub-problems by selecting one or more problem frame template diagrams. The user should be able to switch between a problem and its sub-problems (multiple views). The CASE tool would have flexible templates to cater to frame flavors, model domains and variant frames and a facility to describe particular concerns (overrun, initialization, reliability, identities, completeness). The workbench should facilitate composition of sub-problem machines. In the special cases of dynamic information and transformation problem frames, the CASE tool should provide solutions using JSD and JSP respectively.