A FRAMEWORK
FOR ASSESSING PROCESS REPRESENTATIONS

H. Dieter Rombach

Department of Computer Science
and
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742

E-mail address: dieter@cs.umd.edu
Phone: (301) 405-2707
Telefax: (301) 405-8707

Abstract

Explicit representations of software processes are needed to improve our ability to develop large software products in a predictable manner. Several languages for representing software processes have been proposed. In a graduate-level course at the University of Maryland, several process representation languages were surveyed, applied to a common process example, and assessed according to some framework. This paper describes the applied assessment framework. The framework suggests the presentation of empirical assessment results together with characterizations of the assessment objectives, the assessed language, and the assessment context.
1. Introduction

Explicit representations of software processes are needed to improve our ability to develop large software products in a predictable manner. Several languages for representing software processes have been proposed. In a graduate-level course at the University of Maryland*, several process representation languages were surveyed, applied to a common process example, and assessed according to some framework. This paper describes the applied assessment framework. The framework suggests the presentation of empirical assessment results together with characterizations of the assessment objectives, the assessed language, and the assessment context. We conclude by summarizing some of the assessment highlights. Although, the language assessments need to be viewed with caution, they clearly indicate that existing process representation languages can be categorized - similar to product representation languages - as requirements/specification, design or implementation languages.

2. Assessment Framework

Sound assessment of software engineering techniques, methods, and tools needs to take into account the objectives and context of the assessment. Whereas it is foolish to say

*object-oriented design is better than functional decomposition oriented design,*
it may be valid to say

*object-oriented design has been demonstrated to be better than functional decomposition oriented design in organization XXX to support ease of modification in the context of an iterative enhancement lifecycle model.*

Such an objective and context oriented framework has been proposed for the assessment of requirements and specification approaches [6].

In this section, we propose a similar framework for the assessment of process representation languages. Each language assessment is characterized according to four dimensions:

- **ASSESSMENT OBJECTIVES** (What process issues are being assessed for what purpose?)
- **ASSESSED LANGUAGE** (What language features are being assessed with the above objectives in mind?)
- **ASSESSMENT CONTEXT** (How - in what context - is the assessment being performed?)
- **ASSESSMENT RESULTS** (What are the assessment results wrt. each assessment objective?)

Each of these four dimensions includes a number of factors. An attempt has been made to include already existing frameworks to the extent possible (e.g., the ones presented by Dewayne Perry and Marc Kellner presented at the 4th and 6th International Software Process Workshops, respectively). Factors are numbered with roman/arabic numerals (e.g., I.1) and augmented with example categories. The 18 example capabilities proposed by Marc Kellner and Dieter Rombach for ISFW-6 are pointed out in angle brackets (e.g., [C1]).

---

* The graduate-level course "Modeling Software Processes: Concepts, Languages, and Experience" (CMSC 838R), was held in the Computer Science Department of the University of Maryland at College Park during the Spring semester of 1990.
The entire assessment framework uses the following 19 factors:

The **ASSESSMENT OBJECTIVES** dimension consists of the following three factors:

- **Issues**: What process issues/concerns are being addressed?

  **I.1 Representation**:
  - technical {interface of a process {functionality; behavior; ...};
    implementation of a process {static structure; dynamic structure; ...};
    interconnection {of processes; of processes and products; of processes and people [C18]; of processes and tools [C12]; ...} ...}
  - managerial {resources [C15];
    space issues {multiple objects of same type {product, process, resource} [C18]; ... };
    time issues {schedule [C15]; absolute time [C16]; versions [C16]; ...}; ...}
  - organizational [C8] {responsibility [C8]; physical communication [C9]; purpose, goal and rationale for process step; ...}; ...}

- **I.2 Enaction**:
  - instantiation; multiple instantiation of objects of same type [C18]; object and information management [C8]; resource assignment [C15]; long-term/nested transactions; varying degrees of persistence; tolerance of inconsistency; dynamic types (type hierarchies); dialogues; tool invocation [C12]; execution by humans [C18]; dynamic changes of process models [C17]; back-tracking; historical sequences; interaction between parallel activities; role-specific interpretation; triggering of actions; user intervention; simulation; ...

- **Purpose**: What purpose does the process representation serve?

  **I.3 Purpose**:
  - understanding; communication; planning; measurement [C5]; analysis {of consistency, completeness, risk, ...}; execution guidance; execution control; evolution; improvement [C17]; ...

The **ASSESSED LANGUAGE** dimension consists of the following five factors:

- **II.1 Basic component types**:
  - processes; products; agents; resources; ...

- **II.2 Models for specifying basic components**:
  - input-output; E-R; constraints; pre/post conditions; rules; state transition; object-orientation; algorithmic; communicating processes; ...

- **II.3 Operators for interconnecting basic components [C2]**:
  - control flow {sequence, alternation, iteration [C4]; parallelism [C18]}; data flow [C8]; control structure; data structure [C7]; ...

- **II.4 Operators for generating new components [C1]**:
  - decomposition/composition {process, product [C7], resource}; specialization/generalization {process, product, resource}; ...

- **II.5 Other features**:
  - decision points [C9]; non-determinism [C11]; multiple views; inheritance;

The **ASSESSMENT CONTEXT** dimension consists of the following nine factors:

- **Process of interest**: Which is the process being represented?

  **III.1 Object**:
  - requirements; design; coding; testing; management; performing a change; ...
• Process context: What are important characteristics of the process of interest to know?

III.2 Life-cycle context of process:
- waterfall; iterative enhancement; spiral model; prototyping; ...

III.3 Related processes:
- requirements process; design process; coding process; testing process; management processes; change process; planning process; ...

III.4 Nature of process:
- {creative [C10], mechanical};
- {constructive, analytic [C5]}; ...

III.5 Scale of process:
- {individual, group, city, state};
- {system, subsystem, component}; ...

• Product context: What types of products are being affected by the process of interest?

III.6 Application domain:
- real-time; commercial; scientific; ...

III.7 Product type:
- requirements document; design document; code; measurement data; schedule; ...

III.8 Product requirements:
- reliability; performance; maintainability; ...

• People context: What types of people are affected by the process of interest?

III.9 People:
- customer; user; project planner; developer {requirements eng., designer, coder, tester};
- analyst {tester, qa person, vbv person}; manager; ...

The ASSESSMENT RESULTS dimension consists of the following two factors:

• Attributes: To what degree have the following attributes been achieved with the language under assessment?

IV.1 Representation:
- formality {low/informal [C14]; average/semi-formal; high/formal};
- orthogonality of representing separate issues/concerns {not possible, hard/implicit; easy/explicitly supported};
- understandability {bad; average; good};
- conciseness/brevity {bad; average; good};
- modifiability {hard/by hand; average/via guidelines; easy/automatic support};
- reusability {hard/no support; average/some support (e.g., inheritance); easy/automatic support};
- visualizability {good, acceptable, bad, non-existent};
- tool support {good, acceptable, bad, non-existent}; ...

IV.2 Enaction:
- executability {not possible, hard/by-hand, easy/automated};
- analyzability {wrt.}
  - completeness {not possible, hard/not supported, easy/supported},
  - consistency {not possible, hard/not supported, easy/supported},
  - deadlocks {not possible, hard/not supported, easy/supported}, ...
- visualizability {good, acceptable, bad, non-existent};
- tool support {good, acceptable, bad, non-existent}; ...

This is a preliminary framework in the sense that (a) enaction issues are not treated at par with representation issues, but only included for completeness reasons (e.g., I.2 and IV.2), and (b) many dimensions are incomplete or not at the necessary level of detail (e.g., IV.1). The purpose of
this framework is to serve as a starting point for organizing process issues and assessing candidate languages and approaches.

Using the framework, an example assessment of some process representation language \(<X>\) can be formulated as follows:

The ability of language \(<X>\) to represent

(I.1 Representation: resource assignments)

in order to

(I.3 Purpose: communicate) it

has been assessed by applying language \(<X>\)

(II.1 to II.5 Features: processes, resources; resource assignment)

to represent

(III.1 Object: change process)

in the context of

(III.2 - III.9: Context: waterfall; individual, group)

with the following results:

(IV.1: formality: high; understandability: good; modifiability: easy).

In this context, we concentrate on the assessment of process representation languages rather than approaches. Consequently, representation issues are of primary interest. However, the proposed framework can be applied to the assessment of representation approaches equally well.

Finally, we want to mention that the above framework can also be used to select the most appropriate process representation language for a given project objective and context. The project objective and context can be characterized in terms of I and III, respectively. The desired language features can be characterized in terms of II (i.e., WHAT) and IV (i.e., HOW). Dimensions II and IV define the requirements for the 'ideal' process representation language. We match these requirements against profiles of existing process modeling languages and select the 'closest' match.

3. Assessment Example

In a graduate-level seminar at the University of Maryland, we introduced, discussed and assessed several software process representation languages. Each language was applied to a common 'real-world' example. Students provided their subjective judgements regarding the usefulness of each language. They were asked to rate each language with respect to each of the attributes listed under the 'assessment results' dimension of the framework.

3.1. Process Representation Languages

We investigated the following six process representation languages: APPL/A [8], Grapple [2], HFSP [4], MSL [3], MVP-L1[7], and Statemate [5].

3.2. Example Process

We have chosen to use the existing informal description of the software development process used at NASA/SEL for Fortran-based control software for unmanned spacecraft [1].

This development process is based on a waterfall life-cycle model. It consists of six phases: requirements analysis, preliminary design, detailed design, implementation (= coding and unit
testing), system testing, and acceptance testing. The life-cycle break-down represents an organizational project structure to assign schedules, resources, etc. During each phase more than one activity type takes place, although a given phase is named for the primary activity occurring in it. Development is completed when the system (= system code + system description + user's guide) enter operation/maintenance.

A more detailed description of each phase follows:

- **Requirements analysis**: This phase begins when the development team receives the requirements and specification document from some external organization (= system specification team). This document contains the draft functional specifications. The team consists of a small group of more senior developers. They identify places where the specifications are not complete, analyze the feasibility of the specifications (and algorithms) which are included, add specifications that are purely for software purposes (e.g., display and report specifications), identify all external interfaces, and identify existing code which can be reused on the current project. At the end of this phase, a meeting is held to review the correctness and completeness of the specifications. When the final draft of the requirements and specification document, and the requirements analysis summary report are completed, a formal review known as the System Requirement Review (SRR) is held. The SRR can either approve the results, approve them with changes recommended, or reject them. In general, these three options are available at any review held during the development process. What occurs if the products of a particular phase have severe problems will be described later.

- **Preliminary design**: During this phase the team is still small. After the SRR, the subsystems are determined, and which functions (from the specifications) will go into each subsystem. If alternative designs are to be considered, it is done here. All the interfaces are completely designed down to the subsystem level, and the design of each subsystem is completed to two levels below the subsystem level. The specifications are checked to make sure they are being met. A preliminary design document is kept, and at the end of this phase is a Preliminary Design Review (PDR).

- **Detailed design**: More staff is added during this phase. After PDR, the team continues to refine the design. Prologs, program design language (PDL) documents, all COMMON blocks (in FORTRAN), and interfaces are totally finished. Structured diagrams are done of the total design, and an implementation plan is completed. This phase is culminated by a Critical Design Review (CDR).

- **Implementation**: Staff size peaks in this phase. Subsystems tend to be developed in parallel, according to the specifications in the implementation plan. Each sub-phase of implementation is a build. New code is created, existing one is modified and reused, each module is unit tested, code read, and then put under configuration control. Unit testing is done by the same developer who developed the particular module. Code reading is done by another developer. A management plan is used to keep track of which modules (from the design) have been coded, code read, and tested. Thus management keeps track of how much of the implementation phase is still left to finish. At the end of each build, an integrated current version of the system is (integration-) tested to assure that it is meeting the specification. As these things are being done, the system description and user's guide for the system are also prepared. Test plans for the system testing phase are also drawn up.

- **System testing**: Because requirements changes continue to occur, some implementation overlaps with system testing. During system testing, the team does the tests specified in the system testing plan on the entire system, makes any corrections necessary, documents the results of each system test, and updates the drafts of the user's guide and system description.

- **Acceptance testing**: This phase begins after all system tests have been passed. The acceptance tests are written by a totally different organization (the one which provided the initial specifications). They are specification-based, and are typically developed before the end of system test. The acceptance testing team supervises the development team in carrying out the acceptance tests. Changes and corrections are made to the code if necessary, and the system documents (to be shipped to operation and maintenance) are finalized. The Operational
Readiness Review (ORR) is held to determine that the system is ready to go into operation.

If problems come up at one of the reviews, a formal process exists to handle this. The person identifying the problem submits a problem description to the team, including the impact the problem will have if it is not corrected. The team then considers whether to make the change, make no change, or compromise with the person submitting the report. If there is no agreement, a group known as the Configuration Control Board (CCB), made up of key divisional personnel, will determine the outcome. The result can be major changes, including schedule changes.

3.3. Preliminary Assessment Highlights

The following table summarizes the application of the assessment framework to our six candidate process representation languages. It is important to note that the entries in the ASSESSMENT OBJECTIVES and LANGUAGE FEATURES rows have been derived by studying publications and presentations regarding the respective languages; the entries in the ASSESSMENT CONTEXT rows DO NOT imply limitations of these languages but simply characterize the example problem (see section 3.2) used in this study; and the entries in the ASSESSMENT RESULTS rows are based on the students' subjective impressions.

The positive result of having used our assessment framework was that no language chosen to be the universally best language. Instead, students presented statements such as "language X and Y seem to be suited for the following objectives and from these two I would prefer language X". Students experienced that the effectiveness of a tool in general, and a process representation language in particular, depends on the context in which it is used, the objectives it is used for, and the degree to which its features are understood and used.

The students identified three classes of languages: (1) Appl/A and MSL as languages suitable for process execution, (2) MVP-L1, Statemate and HFSP as languages suitable for descriptive modeling, and (3) Grapple as a language suitable for planning.

As expected, there exists a high correlation between the intended purpose of a language (see 1.3) and its subjective assessment results (see IV.1). For example, all languages designed to support communication do better with respect to comprehensibility; all languages designed to support execution do better with respect to executability. In addition, languages with automated tool support were judged more favorably.

Further detailed results are not included in this write-up. The main reason is that they may have been biased by (i) the familiarity with a language, and (ii) the suitability of the chosen example. Four of the six languages were introduced by their authors (or in the case of Statemate someone intimately familiar with the language): Grapple by Karen Huff, MSL by Gail Kaiser, MVP by myself, and Statemate by Marc Kellner. The 'real-world' example chosen was much more suitable for languages oriented towards descriptive modeling and planning, rather than execution.
<table>
<thead>
<tr>
<th>DIMENSIONS</th>
<th>EXAMPLE REPRESENTATION LANGUAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>APPL/A</td>
</tr>
<tr>
<td>ASSESSMENT OBJECTIVES:</td>
<td></td>
</tr>
<tr>
<td>1.1 Representation</td>
<td>functionality, behavior, static structure, dynamic structure</td>
</tr>
<tr>
<td>1.2 Enaction</td>
<td></td>
</tr>
<tr>
<td>1.3 Purpose</td>
<td>execution control</td>
</tr>
<tr>
<td>LANGUAGE FEATURES:</td>
<td></td>
</tr>
<tr>
<td>2.1 Comp. types</td>
<td>Ada</td>
</tr>
<tr>
<td>2.2 Comp. specification</td>
<td>Ada + relations, triggers, predicates</td>
</tr>
<tr>
<td>2.3 Comp. interconnection</td>
<td></td>
</tr>
<tr>
<td>2.4 Comp. generation</td>
<td>Ada (composition)</td>
</tr>
<tr>
<td>2.5. Other features</td>
<td></td>
</tr>
<tr>
<td>ASSESSMENT CONTEXT:</td>
<td></td>
</tr>
<tr>
<td>3.1 Object of interest</td>
<td></td>
</tr>
<tr>
<td>3.2 Lifecycle context</td>
<td></td>
</tr>
<tr>
<td>3.3 Related processes</td>
<td></td>
</tr>
<tr>
<td>3.4 Nature of process</td>
<td></td>
</tr>
<tr>
<td>3.5 Scale of process</td>
<td></td>
</tr>
</tbody>
</table>
| 3.6 Application domain | | | | | | ??????
| 3.7 Product type | | | | | | ??????
| 3.8 Product requirements | | | | | | ??????
| 3.9 People | | | | | | manager, developer, quality assurance |
| ASSESSMENT RESULTS: | | | | | | |
| IV.1 Representation | | | | | | |
| - Formality | ++ | ++ | + | ++ | o+ | o+ |
| - Orthogonality | - | - | - | - | +/- | + |
| - Understandability | o | o | o/+ | +/- | o/+ | + |
| - Conciseness/brevity | o | | | | | |
| - Modifiability | - | - | - | - | - | o |
| - Reusability | - | - | +/- | o | o | + |
| - Visualizability | -- | o | -- | o | o | + |
| - Tool support | | | | | | |
| planning reuse | -/o | + | +/- | +/- | +/- | o/+ |
| analysis | -/o | - | +/- | +/- | +/- | o/+ |
| IV.2 Enaction | | | | | | |
4. Conclusions

Several software process modeling languages have been proposed. They serve different purposes, are based on different language paradigms, and are being explored and assessed in different environments. In order to develop a better understanding of the weaknesses and strengths of individual languages as well as process modeling in general, we need to establish mechanisms for exchanging experiences. This framework is intended as a first step towards a taxonomy for process modeling languages and a format for expressing assessment results in a comparable manner.

5. Acknowledgements

All students enrolled in the graduate-level course CMSC838R ("Modeling Software Processes: Concepts, Languages, and Experience") during the Spring semester 1990 contributed to the presented assessment results. Special contributions were made by the following students: Andrew Balinsky, Chris Lott, and Bill Martschenko. Special thanks go to our outside experts who volunteered to give guest lectures on their respective approaches: Karen Huff, Gail Kaiser, and Marc Kellner.

6. References


