An Analysis of Method Bridging in Euromethod Version 0

R.J. Wieringa\textsuperscript{1}

September 18, 1995

\textsuperscript{1}Faculty of Mathematics and Computer Science, Free University, De Boelelaan 1081a, 1081HV Amsterdam, the Netherlands. Email: roelw@cs.vu.nl. URL: http://www.cs.vu.nl/~roelw.
Abstract

In version 0 of Euromethod, method bridging is the procedure in which method suppliers describe their methods in a uniform terminology and format, so that different suppliers can use these descriptions to populate delivery plans with the deliverable profiles related to their method. Customers are then expected to be able to compare delivery plans written by different suppliers written in the same format, but using different methods. A description in Euromethod terminology of a deliverable as produced by a method, independent from any delivery plan, is called a method product profile. In this report, the method bridging procedure as well as the structure of method product profiles are analyzed critically.

It is argued that customers cannot and sometimes will not perform method bridging as prescribed by Euromethod version 0. Customers are either not interested in the methods and techniques used by the suppliers, or they are, but in that case they are likely to prescribe a method to the supplier rather than to put in the effort to understand a method they are not familiar with. An alternative approach is proposed, in which those customers that are interested in the methods and techniques used by the supplier, prescribe a particular method to the suppliers. Suppliers must have a toolkit of method components, that they can use to compose the method required by the customer.

Method product profiles presuppose a taxonomy of deliverable types according to which they are classified, as well as a taxonomy of system or project properties that can be described by the method product. These two taxonomies are analyzed critically and two alternative taxonomies are proposed. The consequences of these new taxonomies for method product profiles and for the toolkit approach are investigated.

Acknowledgement. This paper started from a discussion with Wim van Zijp about the utility of a toolkit approach.
## Contents

1 **Introduction** .................................................. 2

2 **Method bridging in Euromethod** ........................................ 5
   2.1 Description of the procedure ........................................... 5
   2.2 Structure of method product profiles ................................... 7

3 **Critical analysis** .................................................. 16
   3.1 The method bridging procedure ......................................... 16
   3.2 Method product profiles ................................................ 17

4 **A taxonomy of views and properties** .................................... 19
   4.1 System views .......................................................... 19
   4.2 A classification of system properties .................................. 22
   4.3 Comparison with other taxonomies ..................................... 22
   4.4 Comparison with the Euromethod taxonomy ................................ 24

5 **A taxonomy of deliverables** .......................................... 29
   5.1 Logic, history and rational reconstruction ............................. 29
   5.2 Product engineering versus process management ....................... 30
   5.3 Product engineering tasks ............................................. 31
   5.4 Product evolution tasks ............................................... 35
   5.5 Comparison with the Euromethod taxonomy of deliverables ............. 35

6 **Discussion** ................................................................ 41
   6.1 Assumptions ............................................................ 41
   6.2 Towards a toolkit for the development of computer-based systems .... 41

7 **Summary and conclusions** ............................................. 44


Chapter 1

Introduction

Euromethod is a method to regulate the interactions between suppliers and customers of information systems, where a supplier is an organization that develops information systems for customers. Euromethod is intended to bridge the differences between development methods used in the European Community and allows suppliers and customers to communicate even when they use different methods. The Euromethod project was initiated by the Commission of the European Community in 1989. It delivered version 0 of its product in 1994, called Euromethod version 0 [28]. This report contains an analysis of the method bridging part of version 0. Henceforth, whenever “Euromethod” is mentioned, “Euromethod version 0” is meant.

The core of Euromethod consists of two sets of guidelines:

- **Method bridging** guidelines, that can be used to describe the development methods of information system suppliers in a uniform terminology, so that customers can compare them.

- **Delivery planning** guidelines, by which one can define a sequence of customer-supplier transactions during the development process.

Euromethod focuses on the Information System (IS) adaptation process, which is a process in which an information system is updated to reflect changed customer requirements. The term “adaptation” deliberately ignores the difference between initial development and perfected maintenance of an information system, because in both these cases there is always some initial information system. An information system may go through several adaptations after initial construction, and different adaptations may be performed by different suppliers.

Each IS adaptation consists of a sequence of customer-supplier transactions (figure 1.1, taken from [44, chapter 16]). In the initial stages of this sequence, the customer issues a call for tender, to which suppliers can respond by offering a plan for performing the desired IS adaptation, called a **delivery plan**. Different suppliers may use different methods, so that different delivery plans use different methods; in order to allow the customer to compare the different delivery plans, the suppliers are required to describe their method in a uniform terminology and format, supplied by Euromethod, and to use this terminology and format in all references to their method in the delivery plan. The terminology and format to be used for the method descriptions is defined by the Euromethod Method Bridging Guide [27] and the Euromethod Concepts Manual 2: Deliverable Model [23]. These method descriptions are independent from any particular IS adaptation process, so that suppliers and customers can use them in different tendering processes. This report summarizes the method bridging approach of Euromethod, gives a critical analysis of this approach, proposes an alternative approach, and compares this with the Euromethod approach. Chapter 2 describes the procedure of method bridging in Euromethod in more detail and summarizes the terminology and format to be used in
Figure 1.1: Customer-supplier transactions in Euromethod.
describing supplier methods. Chapter 3 contains a critical analysis of the method bridging procedure and of the terminology and format prescribed by Euromethod. Chapter 4 contains a proposal for an alternative taxonomy of system properties that can be described by method products and chapter 5 does the same for a taxonomy of tasks and deliverables of system development. Chapter 6 discusses the consequences of these taxonomies for the description of method product profiles and for the toolkit approach. Chapter 7 summarizes the report and points out some topics for further research.
Chapter 2

Method bridging in Euromethod

2.1 Description of the procedure

The call for tender issued by a customer contains the first version of a delivery plan and, possibly, project management plans [25, pages 51-52].

- A complete delivery plan contains the following items of information [26, page 13]:

  - A description of the problem situation. This should include a description of the initial situation and of the desired final situation of the IS adaptation, as well as an assessment of the situational factors of the problem situation.

  - A description of the IS adaptation strategy. This includes the approach to changing the situational factors, the approach to development (incremental development, one-shot development, etc.) and the approach to project control (quality control, configuration management, etc.).

  - A description of the decision points. Each customer-supplier transaction constitutes a decision point in which the customer and supplier need to agree on the status of a deliverable. In particular, for each decision point, one or more required deliverable profiles can be defined, that describe the required content of a deliverable to be exchanged at this point. A required deliverable profile describes the deliverable independently from any method.

- The project management plans, if present, may contain descriptions of the approach to quality assurance, configuration management, etc.

If the customer omits some of this information in the call for tender, then the supplier may want to add the omitted information to his offer.

The procedure to be followed in method bridging in this process is illustrated in figure 2.1.

- Each method defines a number of products to be produced in an application of the method. For each method product, a supplier must produce a method product profile in the terminology and profile of Euromethod. Method product profiles are produced once for each method, and are independent from any IS adaptation.

- Each delivery plan defines a set of required deliverable profiles, that have roughly the same format as method product profiles. There are required deliverable profiles for each decision point (i.e. at each customer-supplier transaction during an IS adaptation). Required deliverable profiles are produced once for each delivery plan, and are independent from any method.
Figure 2.1: The role of method bridging in Euromethod.
When responding to a call for tender, a supplier matches its method product profiles with the required deliverable profiles in the call for tender. The results are called method deliverable profiles. This is possible because a uniform terminology and format is used to describe the method product profiles and the required deliverable profiles. Due to this uniformity, the customer can compare the delivery plan as populated by deliverable profiles from different methods and can use this to choose a supplier.

The division of work in this process is as follows: Suppliers define the method product profiles for their own methods, and customers define the initial, method-independent delivery plan for their IS adaptation. This delivery plan may be quite minimal but contains at least a description of the initial and desired final situations. If the customer has not defined a sequence of decision points in the call for tender, then the supplier can define the delivery plan by deriving the sequence of decision points from his own method, and populate these points with his method product profiles. If the customer has already defined a sequence of decision points with appropriate required delivery profiles in the call for tender, then it suffices for the supplier to populate the decision points of this plan with method deliverable profiles based on the method product profiles.

2.2 Structure of method product profiles

Examples of method products are entity-relationship diagrams, data flow diagrams, etc. Each of these is a method product type, of which in a particular IS adaptation, several instances can be produced. These instances are deliverables of the IS adaptation. Each deliverable of an IS adaptation is characterized by its deliverable type, its considered context, and its purpose [27, page 29].

- The deliverable type is one of the leaves of the taxonomic tree shown in figure 2.2. (The letter “e” in the small circles in the diagram means that the subclasses are exhaustive partitionings of their immediate superclass.)

- The considered context is one of the leaves of the taxonomic tree in figure 2.3 [23].

- The purpose of the deliverable is the reason why this method product is delivered. This may be to facilitate future decision making or to document decisions made, to represent conceptual structures, etc.

A deliverable with one deliverable type, considered context and purpose is called a logical component of a method product. One method product may contain several logical components.
Figure 2.2: Different types of deliverables.
Figure 2.3: Possible contexts of a deliverable.
A method product profile consists of a description of each of its logical components and a description of the required use of the logical components, as shown in figure 2.4. For each logical component, a characterization must be given of the captured properties that it represents. There are two kinds of captured properties, those captured in target domain deliverables and those captured in project domain deliverables. A property captured in a target domain deliverable contains knowledge about the information system that is being adapted; a property captured in a project domain deliverable contains knowledge about the IS adaptation being performed. Captured properties of the target domain are classified into functional and nonfunctional properties. The nonfunctional properties regarded by Euromethod are listed in figure 2.5 as instances of the class Nonfunctional properties. This list is based upon the ISO 9126 standard. The functional properties are classified further into different views as shown in figure 2.5. Figures 2.6 to 2.8 list the functional properties defined by the different views in such a way that the similarities between the views are brought out. The description of the target domain properties captured by a logical component of a method product consists of a list of these properties containing, for each property, an indication of the scope and formality of the description technique used in the logical component. The scope of the property is full or partial (description of all or part of the relevant domain), and the formality is formal, semiformal or informal.

Captured properties of the project domain are classified as belonging to a work area, where a work area is one of the leaves of the project domain deliverable subtree of figure 2.2. Figures 2.9 and 2.10 summarizes the captured properties of the project domain. The list of properties captured by a logical component may contain properties of the project domain, classified by work area, as well as properties of the target domain, classified according to view. Project domain properties are not annotated by scope or formality.
- List of logical components. For each logical component:
  - Deliverable type
  - Considered context
  - Purpose
  - Captured properties
  - Glossary
- Method product usage.
  - Typical development and quality states (e.g. "being produced", "submitted", "accepted")
  - Suitability for problem situations or systems (e.g. suitable for control-intensive systems or data-intensive systems)
  - Internal consistency rules of the method product (e.g. a relationship must connect at least two entity types)
  - External consistency rules with other method products (e.g. every business process must be achieved by one or several tasks)

Figure 2.4: Structure of a method product profile.
Captured properties

Captured properties of target domain

Captured properties of project domain

Nonfunctional properties

Functional properties

Properties defined in IS view

Properties defined in CS view

Properties defined in business information view

Properties defined in business process view

Properties defined in work practice view

Properties defined in CS data view

Properties defined in CS function view

Properties defined in CS architecture view

Figure 2.5: Different system views.
<table>
<thead>
<tr>
<th>Business information view</th>
<th>Computer system data view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information items</td>
<td>Retained data</td>
</tr>
<tr>
<td>Value ranges</td>
<td>Value ranges</td>
</tr>
<tr>
<td>Static dependencies</td>
<td>Static dependencies</td>
</tr>
<tr>
<td>Events that trigger a</td>
<td>Events that trigger a</td>
</tr>
<tr>
<td>change of the item</td>
<td>change in a data item</td>
</tr>
<tr>
<td>Events triggered by a</td>
<td>Events triggered by a</td>
</tr>
<tr>
<td>change</td>
<td>change</td>
</tr>
<tr>
<td>Dynamic dependencies</td>
<td>Dynamic dependencies</td>
</tr>
<tr>
<td>between changes</td>
<td>between changes</td>
</tr>
</tbody>
</table>

Figure 2.6: Captured properties of the business information view and the computer system data view.

<table>
<thead>
<tr>
<th>Business process view</th>
<th>Work practice view</th>
<th>Computer system function view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business events that trigger a process</td>
<td>Triggering organisational events of a task</td>
<td>Input messages (messages that trigger a function)</td>
</tr>
<tr>
<td>Preconditions of a process</td>
<td>Preconditions of a task</td>
<td>Preconditions of a function</td>
</tr>
<tr>
<td>Business events generated by a process</td>
<td>Organisational events generated by a task</td>
<td>Output messages (messages generated by a function)</td>
</tr>
<tr>
<td>Business process decomposition</td>
<td>Task decomposition</td>
<td>Function decomposition</td>
</tr>
<tr>
<td>Dynamic dependencies between processes (sequence, alternative, parallelism, iteration, synchronization, information dependencies)</td>
<td>Dynamic dependencies between tasks (sequence, alternative, parallelism, iteration, synchronization, information dependencies)</td>
<td>Dynamic dependencies between functions (sequence, alternative, parallelism, iteration, synchronization, data dependencies)</td>
</tr>
<tr>
<td>Business rules governing a process</td>
<td>Rules or procedures governing a task</td>
<td>Rules or procedures governing a function</td>
</tr>
<tr>
<td>Information used and generated by a process</td>
<td>Information used and generated by task</td>
<td>Data accessed (created, read, updated, deleted)</td>
</tr>
</tbody>
</table>

Figure 2.7: Captured properties of the business process view, the work practice view and the computer system view.
<table>
<thead>
<tr>
<th>Work practice view</th>
<th>Computer system architecture view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors (performers or managers of tasks)</td>
<td>Processing units (hardware, system software, application software)</td>
</tr>
<tr>
<td>Locations at which actors work</td>
<td>Locations where processing units reside</td>
</tr>
<tr>
<td>Organizational relationships between actors (e.g.</td>
<td>Relationships between processing units</td>
</tr>
<tr>
<td>aggregation into organizational units)</td>
<td>Data distribution over storage units</td>
</tr>
<tr>
<td></td>
<td>Function distribution over processing units</td>
</tr>
<tr>
<td>Communications between actors</td>
<td>Internal communications between processing units</td>
</tr>
<tr>
<td>Human-computer interface (static aspects such as layout,</td>
<td>External communications with other computer systems or other</td>
</tr>
<tr>
<td>and dynamic aspects such as triggering conditions)</td>
<td>information systems</td>
</tr>
<tr>
<td>Information access needs and rights of actors (the</td>
<td></td>
</tr>
<tr>
<td>external views of actors on the information resource)</td>
<td></td>
</tr>
<tr>
<td>Resources necessary for performing a task</td>
<td></td>
</tr>
<tr>
<td>Computerisation of a task</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.8: Captured properties of the work practice view and the computer system architecture view.

<table>
<thead>
<tr>
<th>Development plan properties</th>
<th>QA plan properties</th>
<th>CM plan properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target domain deliverables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>that the plan is related to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization (structure,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tasks and responsibilities,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interfaces)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network of tasks and</td>
<td></td>
<td></td>
</tr>
<tr>
<td>schedules</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detailed manpower and</td>
<td></td>
<td></td>
</tr>
<tr>
<td>budget</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer resources involved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supplier resources involved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input products to be used</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Required infrastructure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(hardware, software, Case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tools, etc.)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.9: Captured properties in project domain plans.
<table>
<thead>
<tr>
<th>Development report properties</th>
<th>QA report properties</th>
<th>CM report properties</th>
<th>Project status report properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target domain deliver-</td>
<td>Target domain deliver-</td>
<td>Target domain deliver-</td>
<td>Deviations from the de-</td>
</tr>
<tr>
<td>ables that the report is re-</td>
<td>ables that the report is re-</td>
<td>ables that the report is re-</td>
<td>livery plan</td>
</tr>
<tr>
<td>lated to</td>
<td>lated to</td>
<td>lated to</td>
<td></td>
</tr>
<tr>
<td>Progress statement</td>
<td>Report on QA procedures</td>
<td>The change requests to</td>
<td>New issues (risks, problems)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the related deliverables</td>
<td></td>
</tr>
<tr>
<td>Used manpower and cost</td>
<td>Protocol of the QA</td>
<td>The accepted change re-</td>
<td>Proposals for corrective</td>
</tr>
<tr>
<td>statement</td>
<td>procedures</td>
<td>quests</td>
<td>actions</td>
</tr>
<tr>
<td>Customer resources used</td>
<td></td>
<td></td>
<td>Proposals for changing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the delivery plan</td>
</tr>
<tr>
<td>Supplier resources used</td>
<td></td>
<td></td>
<td>Proposals for contract</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>change control</td>
</tr>
<tr>
<td>Actual input product</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>used</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deviations from develop-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ment plan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposal for corrective</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>actions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposal for changes in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>development plan</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.10: Captured properties in project domain reports. Entries on one line do not necessarily contain corresponding properties.
Chapter 3

Critical analysis

3.1 The method bridging procedure

The method bridging procedure of Euromethod assumes that the customer is willing and able to compare delivery plans that possibly contain different sequences of decision points or differ in the method products delivered at the decision points. This is a questionable assumption:

• If a customer is unfamiliar with a method, it is not very likely that he is willing to invest the effort to understand descriptions of the products of this method. If the customer is a small business, he may not even be interested in the method used at all; in some cases, only the quality and price of the end-result count. It is the responsibility of the suppliers to guarantee the desired quality and price and only when they fail, they may have to explain the methods they used to produce the result — but they may have to explain this to an arbitration committee of a professional organization, not to the customer.

• Larger customers may have their own in-house method or they may require the suppliers to use a particular method. In those cases, suppliers that cannot use this method, cannot participate in the tendering process.

• Assuming that the customer is willing to compare delivery plans populated with different methods, one can wonder whether the method product profiles, and the method deliverable profiles, contain sufficient information to be able to evaluate the delivery plans. If the customer does not have any knowledge of the method, then he will not learn the method from reading these profiles. But then it is hard for the customer to perform an informed evaluation of the delivery plans.

• Even assuming that the customer knows the methods used by all suppliers and is willing to compare delivery plans populated with products from these methods, then how should the customer evaluate these plans? Is SADT better than Ward/Mellor or JSD in a particular problem situation? How to compare Ward/Mellor, Hatley/Pirbhai, Statemate and OMT? Such a decision would require empirical data from similar projects that used different methods. Such data does not currently exist and is almost impossible to acquire.

These arguments indicate that in some situations, suppliers do not have to populate delivery plans with method products at all, because the customer is not interested in the methods used but in the quality and price of the end-result. In the situations where the customer is interested in the method used to produce the final state of the IS adaptation, the ability to understand and use different
methods should reside with the suppliers rather than with the customer. If the customer wants the supplier to use method $X$, then the supplier should offer to perform the IS adaptation using method $X$. The method prescribed by the customer may be concerned with the target domain or the project domain. Therefore, the supplier should have a toolkit of method components at its disposal containing method product profiles of many different methods, be able to populate a delivery plan with the products of the method selected by the customer, and be able to follow the resulting delivery plan successfully. Note that this simplifies the general procedure shown in figure 2.1 by shifting work from the customer to the supplier.

- The supplier must define a toolkit of method components and be able to use components from any of these methods in the definition of a delivery plan. This investment is independent from any IS adaptation process.
- The customer defines part or all of a delivery plan and prescribes a method to be used in the IS-adaptation. This may be a specification method, a project management method, a planning standard, etc.
- The supplier populates the decision points with the method deliverable profiles of the required method. If necessary, the supplier defines a sequence of decision points for the delivery plan first.

The customer now compares different delivery plans that use the same method product profiles. If the customer defined the decision points himself, the delivery plans are even more compatible and they are a good indication of the understanding that the supplier has of the required method and of the problem situation. The customer now evaluates suppliers according to the supplier's grasp of the selected methods and of the problem situation. Other criteria may also be used, such as the supplier's maturity level as a development organization [8].

### 3.2 Method product profiles

The structure of method product profiles is rather complex and is confusing at some places.

- Captured properties of target domain deliverables are classified by work area. The Transaction Model Concepts Manual [22, pages 9-10] identifies three work areas: IS development, quality assurance, configuration management. The Deliverable Model Concept Manual [23, pages 40-48] classifies the captured properties of the project domain according to the project domain deliverable taxonomy, called work area in the Method Bridging Guide [27], and this gives us seven work areas: Plans for the three work areas mentioned above, reports about these three work areas, and project status reports. The concept of work area is thus not entirely clear, as it involves different taxonomies.

- One may also wonder whether the classification of work areas is exhaustive. The project domain contains more tasks than quality assurance, configuration management and status reporting; It also contains tasks related to planning (such as strategy selection and cost estimation), organizing, acquisition, directing, and other controlling tasks (such as financial reporting).

- The classification of captured properties contains a further confusing matter. Captured properties of project domain deliverables are classified by deliverable type as shown in figure 2.2, but captured properties of target domain deliverables are not, so that figure 2.2 puts one on the wrong foot here. Captured properties of target domain deliverables are classified according to a different taxonomy, shown in figure 2.5. One must therefore look at two taxonomies to understand this classification, which is confusing. It also leaves one in doubt what the target domain taxonomy in figure 2.2 is needed for.

17
• Looking closer at the target domain deliverable taxonomy in figure 2.2, the differences between
the classes dissolve into thin air. For example, the distinction between operational items and IS
descriptions is less clear than it is at first sight. Some IS descriptions — programs, executable
specifications — are operational items. If one decomposes an IS specification into executable
components, then one obtains operational items. If the specification language is itself executable,
then even the specification itself is an operational item. Depending upon the specification
language, it may also be regarded as a prototype (a ROOM specification is for example a system
prototype [32]; a Statemate system specification can even be supplemented with a realistic
user interface [10]). Some prototypes evolve into operational items, etc. To add some more
uncertainty, the distinction between requirement and specification is unclear: any specification of
behavior can be treated as a requirements on an implementation of that behavior. The concept
of requirement itself is unclear: one person’s implementation is another person’s requirement [4].
The presence of operational items in the taxonomy deserves special mention: it seems to be added
as an afterthought and immediately receives the comment that it should not be in the taxonomy
after all; it then appears occasionally at other places in the Euromethod documentation, usually
with the parenthetical remark that operational items belong to implementation and not to
specification.

• The captured properties of the IS and CS views have been listed in such a manner in figures 2.6
to 2.8 that their similarity is emphasized. This similarity undoubtedly intentional, but it also
indicates that an opportunity for simplification has been missed. In the next chapter, a simpler
classification of captured properties is given.

• Because of the similarity between the different views, it is extra confusing to see some dissimilarities
in the definition of the work practice view and the other views. The work practice view
corresponds partly to the business process/CS function views, and partly to the CS architecture
view. There is however no explicit business architecture view.

• Some properties of the dynamics of deliverables are allocated to the business information view
CS data view, and others to the business process/CS function view. This creates a duplication
and a confusion, both of which should be avoided.

• The captured properties of the project domain as recommended by Euromethod prescribe a kind
of project management standard concerning planning and reporting practices. It may be a more
promising approach to relate the approach to project management to the SEI maturity level of
the development organization and use this in the evaluation of offers by different tenderers.

A diagnosis for this confusion may be that method bridging is a result of a compromise rather than
a derivation from a small set of fundamental principles. It is not explained in the Euromethod
documentation why a method product profile should have the recommended structure and no other,
or why these taxonomies would do the job they are intended to do. In the next chapters, an alternative
structure is proposed, that is as much as possible derived from a small set of simple first principles.
Chapter 4

A taxonomy of views and properties

An important part of method product profiles is the table of captured properties, classified by view or work area. We ignore project domain properties in this chapter and look at the classification of target domain properties. We derive a simple framework of system views from some elementary principles borrowed from general systems theory, populate this framework with a sample of system properties, and compare the result with the Euromethod classification of target system properties into views.

4.1 System views

An information system is a particular kind of system, and to find a framework for the description of IS properties, we should first try to find a framework for system properties in general. We can derive the first version of a general system framework from a frequently used definition of the concept of a system, viz. that a system is a coherent interacting collection of components that behave as a whole [3, 39]. The first two dimensions of our framework for system properties are accordingly:

- Properties of externally observable behavior of a system
- Properties of the decomposition of a system into components

System properties are therefore either properties of its externally observable behavior or properties of its decomposition. A further subdivision of system behavior can be given by applying the well-known concepts of state and state transition. The state of a system is a characterization of the possible future behaviors of the system. This is equivalent to saying that the response of a system to a stimulus is determined by its current state and the stimulus. If the system is deterministic, then its current state in turn is fully determined by the past of the system. A deterministic system state is a memory of its past and the response of a deterministic system to a current stimulus is thus determined by the past of the system and the stimulus.

Note that we restrict ourselves to the concept of observable state. We can observe the current state of a system by performing an experiment: apply a stimulus and observe the response. Parts of the states of system components that do not affect the observable behavior of the entire system are not considered to be part of the system state.

A state transition consists of a change of state of a system. State transitions do not happen without a cause. For the systems we consider, the causes of state transitions always reside in the environment of the system. A state transition is caused either by the passage of time (which is part of
Figure 4.1: A framework for system properties.

The second version of our framework for system properties now has three dimensions, shown in figure 4.1. The systems we are interested in are products, which we define as artificial systems built to satisfy a desire of their users. Products have properties that can be classified along yet another dimension, that of utility. Products satisfy a desire of their users and therefore have a benefit (which, in extreme cases, may be 0); but products are built and therefore have a cost as well. This does not add a new dimension to our framework but it does say that if we apply the framework to products, the behavior and decomposition properties considered must include the costs and benefits of this behavior and decomposition.

Using this simple framework for product properties as our starting point, we now ask what, if anything, should be added in the case of software product properties. Now, a characteristic feature of software, and of computer-based systems in general, is that these systems manipulate symbols. The characteristic feature of symbols is that they refer to something according to a convention. Traffic signs refer to traffic rules, characters refer to letters, etc. We can therefore distinguish a symbol from its denotation. There is nothing in the physical nature of a symbol that makes it refer to its denotation. A symbol refers to its denotation only by virtue of a convention, and this convention is a social construction agreed upon by people.

The behavior of computer-based systems consists of the manipulation of symbols, and the states of the system consists of a collection of symbols. Again, this does not add anything to our simple framework of system properties, but it does say that if we apply our framework to computer-based systems, one of the properties of system behavior is the convention by which the symbols refer to something.

A convenient way to specify the convention is to specify the universe of discourse (UoD) of the system, which is the part of the world to which the manipulated symbols refer. The UoD of a CBS is a system, and it can be decomposed into parts that are often called entities or objects. Each state of the CBS can be structured in such a way that for each existing object in the UoD it contains a symbol that is a surrogate for that object, and for each link between UoD objects, it contains a surrogate for that link. For example, if the CBS is an IS, it is common practice to structure the state this way, but it can be shown that real-time embedded systems have a UoD too [11, 42]. This means that the decomposition of the UoD into objects is specified as part of the specification of the state of the CBS! The situation is more complicated if the CBS is itself decomposed into parts, such as in a distributed system (figure 4.2). This means that a surrogate can be allocated to one component of the CBS or that it can be sliced into pieces that are located at different components.
Figure 4.2: Decomposition of the UoD and of the CBS.
4.2 A classification of system properties

To further classify system properties, we need to fill out the three views by an indication of the kinds of properties that we may want to specify in each of the three views. Figure 4.3 gives a taxonomy based on the systems view outlined above. It assumes that the system interacts with its environment by means of transactions. It incorporates the phenomenon that the decomposition of the UoD into objects is represented in a system state and that the behavior of the system should have a utility. All system properties are divided into structural properties, that concern the kinds of things that exist in this view, and constraints on these things. This partitioning is orthogonal to the partitioning into state, transition and decomposition properties.

The list of properties is not exhaustive. The structural properties are usually regarded as functional. Most of the constraints in the list, except life cycle constraints and probably cardinality constraints, are usually classified as nonfunctional properties. Some of the nonfunctional properties listed by Euromethod appear in the classification:

- Volume (cardinality constraints)
- Frequency (transaction constraint)
- Duration (transaction timing constraint)
- Efficiency (decomposition constraint)
- Security (transaction constraint)

The other nonfunctional properties cannot be classified as state, transition or decomposition properties. They must be listed as a separate category, applicable to the combination of possible states, possible transitions and the decomposition of the system:

- Cost
- Benefit
- Criticality
- Reliability
- Maintainability
- Portability
- Usability

4.3 Comparison with other taxonomies

The resulting framework combines the well-known partitioning of system views into the behavior view, the process view and the data view by Olle et al. [20] with the partitioning of system views into the behavior view, the function view, the system architecture view and the data architecture view of Wood and Wood [47] and of Statemate [7, 9] (figure 4.4).

The process perspective (Olle) functional view (Wood & Wood) always describes the way in which the system responds to stimuli. This may be a simple list of system transactions or it may consist of a more elaborate specification of stimuli-response paths, e.g. by means of data flow diagrams. Often, a function decomposition of the system is made, in which the function that the system has for its users
• State properties.
  
  - Structure.
    * Which surrogate are possible/currently exist in the system
    * What is the state of existing surrogates
    * What are the possible/existing relationships between surrogates
    * What is the taxonomy of surrogate types
  
  - Constraints.
    * Cardinality constraints
      · How many objects of a particular type minimally/maximally/on the average
      · How many instances of a relationship minimally/maximally/on the average
      · How many objects on one type related to surrogates of another type minimally/maximally/on the average
    * Other constraints
      · Value range constraints
      · Exclusion and inclusion constraints
    * Consistency with other views

• Transition properties.

  - Structure.
    * Which transactions are possible/have occurred
    * What are the transaction preconditions
    * What are the transaction effects

  - Constraints.
    * Life cycle constraints
    * Cardinality constraints (how many simultaneous transactions maximum, minimum, average)
    * Utility properties
      · To which function (= useful behavior) does this transaction contribute
      · Transaction cost
    * Other constraints
      · Maximum, minimum, average transaction frequency
      · Security constraints (e.g. safety and liveness constraints)
      · Timing constraints (duration, response time, etc.)
    * Consistency with other views

• Decomposition properties.

  - Structure.
    * Which components exist
    * What is the allocation of state and behavior to components
    * What is the state of the components
      · Location
    * What are the relationships between components
      · Physical relations
      · Social relations
      · Communications

  - Constraints.
    * Resources needed
    * Utility constraints (cost)
    * Consistency with other views

Figure 4.3: A taxonomy of system properties.
is decomposed into subfunctions, until the level of transaction is reached. This is an important part of the specification of the benefit that the system should have for its users.

The behavior perspective gives properties of two or more state transitions. These may be represented by finite state machines, by statecharts, or by other means.

The data perspective always refers to the structure of system states (for example by means of an entity-relationship diagram and to some properties of the states (e.g. cardinality properties). If an entity-relationship technique is used, this is an important part of the specification of the meaning of the symbols manipulated by the system. As explained before, this is at once a specification of the decomposition of the UoD into entities.

### 4.4 Comparison with the Euromethod taxonomy

The Euromethod IS views are really views of three different kinds of system:

- The **computer system views** in the Euromethod classification are views on the information system. We will generalize this a bit and talk of a view on computer-based systems (CBSs). CBSs include information systems, EDI systems, real-time embedded systems, and any other system that contains a component which does symbol-manipulation by means of a computer.

- The **business views** in the Euromethod classification are partly views on the part of the world to which the manipulated symbols refer. These are therefore views on the **Universe of Discourse** of the CBS.

- The **work practice view** in the Euromethod classification are views on the immediate working environment of the CBS. The business views also contain a view on this immediate environment. We call the immediate usage environment of the CBS the **Environment of Discourse** (EoD) of the CBS. The EoD and UoD of a CBS may overlap.

These three kinds of systems appear as the system world, the subject world and the usage world in the NATURE project [12].

The correspondence between the Euromethod views on these systems and the system views defined here is shown in figure 4.5. It is fairly straightforward to verify that our taxonomy of system properties covers all properties listed by Euromethod for the three considered systems (CBS, UoD and EoD) under each of the three views (state properties, transition properties, decomposition properties). This verification is shown in figures 4.6 to 4.12.
<table>
<thead>
<tr>
<th>Euromethod views</th>
<th>Taxonomy defined in this report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views on CBS (information system views)</td>
<td>Any system</td>
</tr>
<tr>
<td>Views on UoD (business views)</td>
<td></td>
</tr>
<tr>
<td>Views on EoD (work practice views)</td>
<td></td>
</tr>
<tr>
<td>Data view</td>
<td>State properties</td>
</tr>
<tr>
<td>Function view</td>
<td>Transition properties</td>
</tr>
<tr>
<td>Architecture view</td>
<td>Decomposition properties</td>
</tr>
</tbody>
</table>

**Figure 4.5:** Comparison between the taxonomy of properties defined in this report and the Euromethod taxonomy.

<table>
<thead>
<tr>
<th>State properties</th>
<th>Computer system data view</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure</strong></td>
<td></td>
</tr>
<tr>
<td>• Which surrogate are possible/currently exist in the system</td>
<td>Retained data</td>
</tr>
<tr>
<td>• What is the taxonomy of surrogate types</td>
<td></td>
</tr>
<tr>
<td>• What are the possible/existing relationships between surrogates</td>
<td>Static dependencies between the data</td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
<td></td>
</tr>
<tr>
<td>• Other constraints</td>
<td>Value ranges</td>
</tr>
<tr>
<td>• Value range constraints</td>
<td></td>
</tr>
<tr>
<td>• Consistency with other views</td>
<td>Events that trigger a change in a data item</td>
</tr>
<tr>
<td>(Redundant)</td>
<td>Events triggered by a change</td>
</tr>
<tr>
<td>(Redundant)</td>
<td>Dynamic dependencies between changes</td>
</tr>
</tbody>
</table>

**Figure 4.6:** Comparison of taxonomies on CBS views.
<table>
<thead>
<tr>
<th>Transition properties</th>
<th>Computer system function view</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure.</strong></td>
<td></td>
</tr>
<tr>
<td>• Which transactions are possible/have occurred</td>
<td>Input messages (messages that trigger a function)</td>
</tr>
<tr>
<td>• What are the transaction preconditions</td>
<td>Preconditions of a function</td>
</tr>
<tr>
<td>• What are the transaction effects</td>
<td>Output messages (messages generated by a function)</td>
</tr>
<tr>
<td><strong>Constraints.</strong></td>
<td></td>
</tr>
<tr>
<td>• Utility properties</td>
<td></td>
</tr>
<tr>
<td>- To which function (= useful behavior) does this transaction contribute</td>
<td>Function decomposition</td>
</tr>
<tr>
<td>• Life cycle constraints</td>
<td>Dynamic dependencies between functions (sequence, alternative, parallelism, iteration, synchronization, data dependencies)</td>
</tr>
<tr>
<td>• Other constraints</td>
<td>Rules or procedures governing a function</td>
</tr>
<tr>
<td>• Consistency with other views</td>
<td>Data accessed (created, read, updated, deleted)</td>
</tr>
</tbody>
</table>

Figure 4.7: Comparison of taxonomies on CBS views.
<table>
<thead>
<tr>
<th>Decomposition properties</th>
<th>Computer system architecture view</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure.</strong></td>
<td></td>
</tr>
<tr>
<td>• Which components exist</td>
<td>Processing units (hardware, system software, application software)</td>
</tr>
<tr>
<td>• What is the allocation of state and behavior to components</td>
<td>Data distribution over storage units</td>
</tr>
<tr>
<td>• What is the state of the components</td>
<td>Function distribution over processing units</td>
</tr>
<tr>
<td>– Location</td>
<td>Locations where processing units reside</td>
</tr>
<tr>
<td>• What are the relationships between components</td>
<td>Relationships between processing units</td>
</tr>
<tr>
<td>– Physical relations</td>
<td></td>
</tr>
<tr>
<td>– Communications</td>
<td>Internal communications between processing units</td>
</tr>
<tr>
<td></td>
<td>External communications with other computer systems or other information systems</td>
</tr>
</tbody>
</table>

Figure 4.8: Comparison of taxonomies on CBS views.

<table>
<thead>
<tr>
<th>State properties</th>
<th>Business information view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identities to the correlation of state properties of CBSs</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.9: Correlation of state properties with properties of the business information view.

<table>
<thead>
<tr>
<th>Transition properties</th>
<th>Business information view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identities to the correlation of transition properties of CBSs</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.10: Correlation of transition properties with properties of the business process view.

<table>
<thead>
<tr>
<th>Transition properties</th>
<th>Work practice view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identities to the correlation of transition properties of CBSs</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.11: Correlation of transition properties with properties of the work practice view.
<table>
<thead>
<tr>
<th>Decomposition properties</th>
<th>Work practice view</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure.</strong></td>
<td></td>
</tr>
<tr>
<td>- Which components exist</td>
<td>Actors (performers or managers of tasks)</td>
</tr>
<tr>
<td>- What is the state of the components</td>
<td>Locations at which actors work</td>
</tr>
<tr>
<td>- Location</td>
<td></td>
</tr>
<tr>
<td>- What are the relationships between components</td>
<td>Organizational relationships between actors (e.g. aggregation into organizational units)</td>
</tr>
<tr>
<td>- Social relations</td>
<td></td>
</tr>
<tr>
<td>- Communications</td>
<td>Communications between actors</td>
</tr>
<tr>
<td></td>
<td>Human-computer interface (static aspects such as layout, and dynamic aspects such as triggering conditions)</td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
<td></td>
</tr>
<tr>
<td>- Consistency with other views</td>
<td>Information access needs and rights of actors (the external views of actors on the information resource)</td>
</tr>
<tr>
<td>- Resources needed</td>
<td>Resources necessary for performing a task</td>
</tr>
<tr>
<td></td>
<td>Computerisation of a task</td>
</tr>
</tbody>
</table>

Figure 4.12: Correlation of decomposition properties with properties of the work practice view.
Chapter 5

A taxonomy of deliverables

A second important taxonomy used in the specification of method product profiles is the taxonomy of deliverables. In this chapter, we give a taxonomy of the tasks of product development. Because each task produces a deliverable, this is also a taxonomy of deliverables of system development. We compare this with the deliverable type taxonomy of Euromethod in figure 2.2.

5.1 Logic, history and rational reconstruction

In any planned process that is performed to achieve a result, we can distinguish the logical structure of the process from the temporal structure of the process. This is obvious in the case of mathematical discovery: a theorem may be arrived at by starting with a hunch, meandering through a host of possibly relevant and possibly irrelevant intermediary results, leading eventually to the desired result. Afterwards, we rationalize the road that leads to it by hacking away all irrelevant sideways and present as simple and clear a proof as we can construct. In some cases it may take centuries to arrive at such a rationalized presentation, but once all the polishing is done, the result and its proof may take half a page. Lakatos [15] gives a famous example concerning Euler’s conjecture. All mathematician’s know from their own experience that the history of the way a proof is found is much more chaotic than the pure logical structure of the proof as presented to others. The logical structure of the proof as presented to others is a rational reconstruction of the actual process that led to the result. The rational construction describes history as it would have been in an ideal world, where we make no mistakes; the actual history is the process followed in the actual world, where we do make mistakes. The rational reconstruction of the proof has two functions:

- It serves as a justification of the validity of the result to others.
- It can serve as a guideline for planning the proof at the start of our search.

In the empirical sciences, we see the same pattern: An empirical generalization may be found in an erratic manner, with unexplicable deviations from expected behavior, faulty instruments, erratic subjects, laboratory personnel turnover and mistakes in computations; but the result is presented in a rationalized manner as if the empirical cycle of discovery were followed in all its purity and simplicity.

Mathematics and empirical science aim at an increase of our knowledge. By contrast, we are interested here in a planned process aimed at changing the world. We call this a development process, taking “development” in the general sense to include initial development from scratch, an adaptation process of an existing system, or a maintenance process. In this case too we can make the same distinction between the logic and the history of the process. The logical structure of a
development process is the rational process as it would take place in an ideal world where we make no mistakes; the temporal structure of the process is the structure of the process as it actually takes place, and this is usually different from the structure of the rational process. We look at the structure of this rational process in later. Here, we note that the rational process must have the same function as the rational processes of mathematics and empirical science have:

- The result can be justified by producing a rational reconstruction of the process that led to the result as if this rational process were followed. This rational reconstruction is the justification of the result.

- The rational process can be used as a guideline for planning the process. The closer we can make the actual process resemble the rational process, the better we are in a position to produce justifications of the result.

This view of the role of a rational development method in the software development process is advocated by Parnas and Clements [21].

An interesting side remark is that this role of rational reconstruction has been observed by Suchman [36, 37] in the behavior of office personnel. The actual course of events in an office is too unpredictable and chaotic to be captured by a procedure that takes care of all errors, like invoices received for the wrong amount, for missing orders, etc. Formal office procedures do not state what to do in the case of every possible error. When an error occurs, office personnel engages in an activity in which they try to find out which sequence of events should have happened, and they then try to construct this history after the fact. They contact the sender of the invoice, they copy missing data from other departments, etc. This constructed history is a rational reconstruction of actual history as it should have taken place according to the office procedures, and it is the history they record in the books. This is not a falsification of history but a rational reconstruction of history as it would have taken place when the world would have been ideal.

- In this way, office employees take responsibility for what happened in the office, and it is the only way they can take responsibility for it [37, pages 326-327]. The rational procedures are an instrument to create accountability.

- The office procedures are the basis for planning the daily course of action.

This does not mean that office workers “fake” the appearance of orderliness in the records. Rather, the construction of orderly records is the construction of evidence of action in accordance with the procedures. Computer scientists too easily look upon these procedures as if they were computer programs to be followed by office workers, just as they construe development methods as programs to be followed by themselves. This should be kept in mind when we define a structure of the rational development process below. Before we do that, we introduce a distinction between technical and managerial development tasks.

5.2 Product engineering versus process management

Management can be defined as the achievement of objectives through others [17]. In order to achieve objectives, we must do the following:

- Plan the process, including the determination of the objectives to be achieved;
- Acquire the resources to perform the process;
- Organize the process into manageable units;
• *Direct* the workers who perform the process;

• *Control* the work by comparing the results against the objectives.

There is no temporal sequence in these activities: In general, management performs all of these tasks in parallel. The principle of the **universality of management** says that the management of any process to achieve a result consists of roughly the tasks just mentioned, regardless of the kind of result produced [38]. This means that we can partition any development process into two kinds of tasks, management tasks, that are universal, and technical tasks, that are oriented to the development of a particular kind of product. Examples of management tasks in a development process can be given readily:

• An important planning task is *strategy planning*, which is the determination of the temporal structure of the development process (rather than its logical structure). Possible development strategies include linear development, incremental development and evolutionary development. The choice between these and other strategies is made on the basis of situational factors, using for example the contingency tables provided by Euromethod [24].

• Two important controlling tasks are *quality assurance* and *configuration management*. These activities must be planned too, so there are corresponding planning tasks.

Our first version of a partitioning of development tasks is therefore the following:

• Technical development tasks

• Management tasks
  
  – Planning
     * Strategy selection
     * QA planning
     * CM planning
     * …
  
  – Organizing
     * …
  
  – Acquisition
     * …
  
  – Directing
     * …
  
  – Controlling
     * QA tasks
     * CM tasks
     * …

5.3 **Product engineering tasks**

We partition technical development tasks into product engineering tasks, discussed in this section, and product evolution tasks, discussed later in this chapter. We discuss the rational structure of these processes, not their temporal structure. These processes have only one rational structure, but
there are many different temporal structures. As mentioned earlier, strategy planning deals with the selection of an appropriate temporal structure for a particular development process.

**Product engineering** can be defined as a process of rational choice aimed at obtaining a specification of required external product behavior. A **rational choice process** has the following structure:

- analysis of the initial situation,
- generation of alternative solutions (also called synthesis),
- simulation of the effects of the solutions,
- evaluation of these effects,
- choice of a solution.

This is the rational problem solving cycle as proposed by Simon [33, 34] and which goes back to Dewey [5, pages 107-115]. It also resembles the underlying structure of management decision making found by Mintzberg et al. [19]. The process of planned change as defined by Kolb and Frohman [14] also has the structure of the rational problem solving cycle. Applied to product development, we get the logical structure of the engineering process shown in figure 5.1, taken from [44, chapter 3]. The use of the rational choice process as a model for the rational engineering process is widespread [1, 2, 6, 13, 29, 31]. This gives plausibility to the hypothesis that this is the logical structure of the engineering process, just as the empirical cycle is the logical structure of the discovery process.

The engineering cycle is characterized by two important features.

- **Specification.** The cycle results in a specification of what is to be done, not in an implementation. Engineering is an example of the adage “think before you act”.

- **Simulation.** An essential part of this adage is that the product, or product version, to be realized is simulated before it is built. This allows a rational choice between specifications, based on a comparison of the simulation results with the change objectives. The simulation may be extremely informal or extremely formal; it may consist of thinking through the specification, walking through it, making a throw-away prototype, performing a statistical simulation, making a mock-up, looking at the experience of the neighbors, executing a specification, proving properties from a formal specification, showing the specification to domain specialists, or anything else that gives reliable information about what would happen if we were to implement the specification.

The engineering cycle gives us one dimension of our taxonomy of engineering tasks. The second dimension will be provided by the **aggregation hierarchy** of systems. In the previous chapter, the standard view of systems as coherent interacting collections of components was used. Since each of the system components can itself be viewed as a system, we get an aggregation hierarchy such that systems at one level of the hierarchy are immediate components of systems at the next higher level of aggregation. Restricting oneself to consideration of systems at one level in an aggregation hierarchy is an important complexity reduction tool, applied in empirical science as well as in engineering. There are many different proposals for aggregation hierarchies; a useful one for software products is the following:

- **The social system** into which a product is embedded. The social system may be human-machine system consisting of one user and one machine, or it may be an organization (in which an IS is embedded), or a group of organizations (in which an EDI system is embedded).

- **Taking the archetypical case of a product developed for use in an organization, the next lower level of aggregation is that of an computer-based system.** Examples of computer-based system are information systems, EDI systems, elevator systems and flight simulators.
Figure 5.1: The engineering cycle.
<table>
<thead>
<tr>
<th>Needs analysis</th>
<th>Behavior specification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Synthesis</td>
</tr>
<tr>
<td>Social system</td>
<td></td>
</tr>
<tr>
<td>Computer-based system</td>
<td></td>
</tr>
<tr>
<td>Software system</td>
<td></td>
</tr>
<tr>
<td>Software subsystem</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.2: A framework for product engineering.

- In the cases of interest for us, the computer-based system is composed, at the next lower level, of hardware and software systems (and possibly other kinds of systems). For example, an information system, viewed as subsystem of an organization, is composed of hardware, software, users, procedures followed by users, and data manipulated by the people and the software. If the computer-based system is composed of hardware and software only, such as an elevator system, then it is customary to call it simply the system and to call the software embedded. There are other relevant kinds of systems at this level of aggregation, such as hardware systems, users, and operators.

- Software systems are in turn composed of software subsystems, which may be called packages, modules, classes, or whatever.

We can use the engineering cycle at any level of aggregation. This gives us the framework for product engineering shown in figure 5.2, taken from [44, chapter 3]. There is a hidden task in this framework, viz. decomposition of a system specified at one level of aggregation into a collection of interacting subsystems at the next lower level of aggregation.

Decomposition can be performed according to the engineering cycle as well: generate alternative decompositions, simulate them, evaluate the simulations, choose. This does not lead to new engineering tasks beyond the decomposition task. Taking this task into account, we now arrive at our second version of our taxonomy of tasks:

- Technical development tasks
  - Product engineering
    * Needs analysis
    * Generating alternatives
    * Simulation
    * Evaluation
    * Decomposition
  - Product evolution

- Management tasks
  - Planning
    * Strategy selection
    * QA planning
    * CM planning
* ... 
- Organizing 
  * ... 
- Acquisition 
  * ... 
- Directing 
  * ... 
- Controlling 
  * QA tasks 
  * CM tasks 
  * ... 

5.4 Product evolution tasks

Product engineering does not result in a product, but in a decision about required product behavior at a particular level of aggregation. If we repeatedly specify a decomposition of the product into subsystems and at each level specify required (sub)system behavior, all we have is a collection of specifications and not a product. A specification of the behavior required of a car and of all its parts is not a car but a collection of text and diagrams. Product engineering is followed by an act of construction or production, use, and evaluation. This progresses in an endless product evolution cycle as shown in figure 5.3 [44, chapter 3]. Some of the use and evaluation of the product takes place in the laboratory and at test sites, some at selected customer sites, but most of it takes place at real customer sites. We deliberately ignore these distinctions in figure 5.3 because they do not affect the logical structure of product evolution. However, note that evaluation takes different forms, depending upon whether it takes place in the development laboratory or at a customer site.

The essential tasks in product evolution are implementation, use and evaluation. These tasks follow the logic of the regulatory cycle shown in figure 5.4 [44, chapter 3]. The regulatory cycle is the logical structure of a feedback control loop. The process of organizational consultancy defined by Kolb and Frohman [14] consists basically of an action preceded by a rational choice process, embedded in a regulatory cycle. Van Strien [35] argues that the regulatory cycle is the logical structure of any planned changed process. In [44] it is shown that the standard V-strategy of software product development results from a recursive application of the regulatory cycle at increasingly lower levels of aggregation. The essence of the regulatory cycle can be paraphrased by the adage “think after you act”. It is concerned with systematic collection of data about the results of an action after it is performed, comparison with the objectives, and planning of further actions in the light of these evaluations.

Software products however have the peculiar property that we can specify their decomposition in such a way that, at a sufficiently low level of aggregation, we have an executable specification of the product. If an execution of this specification has the required performance properties, it is an implementation of the product. To implement a software product, we decompose a specification and then repeatedly integrate the parts until we have the entire product. We incorporate the integration and evaluation tasks in our taxonomy. The entire taxonomy is shown in figure 5.5.

5.5 Comparison with the Euromethod taxonomy of deliverables

Our taxonomy of tasks readily gives a taxonomy of deliverables, shown in figures 5.6 and 5.7.
Figure 5.3: Product evolution. The arrows indicate direction of influence.
Comparing this with the Euromethod taxonomy in figure 2.2, at the top level of both taxonomies, there is a clear correspondence, but there is divergence at lower levels. In particular, figures 5.6 and 5.7 contain a more detailed breakdown of deliverables. The Euromethod target domain deliverables can be mapped roughly to corresponding deliverables in figure 5.6, that are generalizations of the Euromethod target deliverables:

- Requirements are generalized to product objectives.
- Prototypes are one particular example of simulations.
- Operational items correspond very roughly to decompositions.
• Technical development tasks
  - Product engineering
    * Needs analysis
    * Generating alternatives
    * Simulation
    * Evaluation
    * Decomposition
  - Product evolution
    * Integration
    * Product evaluation
      - Testing (unit, integration, acceptance, installation testing)
      - Customer satisfaction evaluation

• Management tasks
  - Planning
    * Strategy selection
    * QA planning
    * CM planning
    * ...
  - Organizing
    * ...
  - Acquisition
    * ...
  - Directing
    * ...
  - Controlling
    * QA tasks
    * CM tasks
    * ...

Figure 5.5: A taxonomy of development tasks.
Figure 5.6: Taxonomy of technical deliverables of the development tasks.
Figure 5.7: Taxonomy of management deliverables of the development tasks.
Chapter 6

Discussion

6.1 Assumptions

So far, we have given a taxonomy of system properties classified under three views (state, transition and decomposition properties), and we have given a taxonomy of tasks and deliverables of system development. These taxonomies are not not subject to the critique of chapter 3. They are derived from a small set of simple and uncontroversial assumptions. The assumptions of the taxonomy of system properties are:

- Systems have behavior and decomposition.
- System behavior is characterized by state space and state transitions.
- Products have a benefit and a cost.
- Computer-based systems manipulate symbols.
- Symbols have a denotation by convention.

The list of properties used to populate the taxonomy also assumes that systems respond to stimuli by means of transactions (i.e. without observable intermediary states).

The assumptions of the taxonomy of development tasks are:

- There is a distinction between technical tasks and management tasks.
- Rational engineering follows the rational choice process.
- Product evolution follows the regulatory cycle.
- A particular aggregation hierarchy has been assumed.

6.2 Towards a toolkit for the development of computer-based systems

Our revision of the two taxonomies does not change the way method product profiles are specified very much. If one accepts the validity of the two alternative taxonomies proposed here, then the only consequences for method product profiles are the following:

- The space of deliverable types is larger
• The list of captured properties is longer

• The internal and external consistency rules of a method product profile are already specified as part of the constraints in the different system views.

This does not invalidate the use of method product profiles. The critique of section 3.1 goes further and says that the approach to method bridging in the current version of Euromethod is itself unsound. Instead of providing a uniform descriptions of their method, that must be read and understood by customers, a suppliers should have a toolkit of method components so that it can use the method prescribed by the customer, adapted to the development situation.

The toolkit should contain tools that can be used for at least one development task. In particular, there should be tools for the specification of product objectives and product behavior. Example tools for objectives specification are:

• Analysis of problems and interest groups (ISAC [16])
• Critical Success factor analysis (IE [18, 30])
• Function decomposition (IE [18])

There are many more tools, some of which are analyzed in detail in [44]. In order to discover the objectives of a product one must analyze its EoD in order to discover the needs that live in the EoD. Some methods recommend making a rough model of the EoD consisting of one or both of the following:

• A model of business activity (ISAC [16])
• A model of UoD decomposition into objects (IE [18])

The reason for including a model of UoD decomposition here is that business activity in the EoD is itself symbol manipulation and to understand this activity, one must have a model of the UoD. As explained earlier, modeling the decomposition of the UoD is (also) part of modeling the structure of the states of the CBS.

Examples of tools for behavior specification of CBSs are:

• Tools for generating alternatives
  – Brainstorming, brainwriting, etc.
  – JAD sessions

• Tools for behavior specification
  – ER diagrams to specify the state space and some of its cardinality constraints.
  – Data flow diagrams to specify transactions (stimulus-response paths).
  – Finite state machines or statecharts to specify life cycle constraints

• Simulation tools
  – Walkthroughs
  – Throw-away prototyping
  – Making mockups
  – Low-tech (paper-and-pencil) user-interface simulation

• Evaluation tools
- Controlled convergence [29]
- Making product profiles
- Ranking and weighting

- Decomposition tools
  - Information hiding
  - Object-oriented decomposition heuristics
  - Structured decomposition heuristics

It should be noted that quantitative evaluation methods are hardly used in software product engineering. The examples listed above are taken from a book on industrial product engineering [31].

Obviously, defining a set of coherent and well-integrated tools for behavior specification alone is still a research topic. It would go beyond the bounds of this report to discuss in detail what how this can be done. A preliminary version of such a toolkit is given in [41]. A detailed analysis of some existing methods in search of useful tools is given in [45, 43, 46, 44, 40]. The results of this search will be used to define a second version of the toolkit. The two taxonomies defined in this report play the following role in the definition of this toolkit:

- Each tool should have a “direction for use” that says for which tasks in the taxonomy of development tasks it is intended to be used, and

- there should be tools for the specification of all behavior or decomposition properties in the taxonomy of properties.

This research can benefit from research in formal specification, because by formally specifying a tool and the results that can be delivered by it, we can eliminate ambiguities, spot redundancies and inconsistencies and in general strip down our tools to their simplest essentials. This does not mean that the users of the toolkit need to use formal methods or even be aware of them: The user of a hammer does not need to be aware of any of the technology that went into the design and production of the hammer.
Chapter 7

Summary and conclusions

We have given a summary of the Euromethod version 0 method bridging procedure and analyzed the structure of method product profiles. The method bridging procedure was criticized for placing too heavy a demand on the customer, and an alternative was proposed, in which suppliers have a toolkit with method components from which they assemble the method prescribed by the customer. Method product profiles are classified according to deliverable type, and contain lists of captured properties classified according to the view on the target domain or project domain to which they belong. The taxonomies of deliverables and views were found to be confusing and ad hoc. After an analysis, two alternative taxonomies have been proposed, based on a small number of simple and plausible assumptions.

Further work includes comparison and analysis of existing methods in search of useful method components, and integration of these results into a coherent toolkit. The resulting toolkit must be validated in realistic applications. We can expect new tools to be developed all the time, but the introduction of these tools should be a pleasant surprise for the tool users rather than a blinding experience that makes the tool user forget about all tools that he or she was familiar with.
Bibliography