FORMAL MODELING FOR DEPLOYING IMPROVEMENT AND INNOVATION IN INFORMATION TECHNOLOGY
Position paper

Philippe Michelin
AEBIS, 1155 René Lévesque, Montréal, Québec, Canada, H3B2K4
pmichelin@aebis.com

Marc Frappier
Dept. Computer Science, Université de Sherbrooke, Québec, Canada, J1K 2R1
marc.frappier@usherbrooke.ca

Keywords: Definition, distinction, information technology innovation and improvement.

Abstract: In this paper, we show the importance of building precise models of basic concepts before conducting strategic information technology improvement and innovation. We use the notions of distinction and definition to build precise models. Our modelling approach mixes both natural language and mathematical definitions, to reach the appropriate level of precision. This modelling approach constitutes a form of knowledge management similar to approaches like enterprise architecture, but focusing on modelling the impacts of a given technology innovation and conducting projects in a more agile style.

1. INTRODUCTION

Information is everywhere in a modern Organization; Information Technology (IT) has become the key technology to capacity innovation, productivity improvement, extension of services, and time to market. But it is not so easy for Business and IT people to weigh and plan the impact of IT innovations that are supposed to fulfill their needs.

In the experience we have acquired over the years in IT projects, clarity of discourse is a foremost success factor we have regularly identified. The ability to settle the most important issues in a project invariably relies on a clear vision of the main concepts at stake. All too often, issues arise from misunderstandings between project stakeholders. The field of IT is complex, riddled with ambiguities which are hard to dispel. There are countless examples of ambiguities that could be drawn from the literature in IT. For instance, Service Oriented Architecture (SOA), to pick one, is a hot technology that is currently attracting much interest in the IT community. However, SOA is also a poorly understood concept, encompassing several notions from which too many people make unrealistic promises, or expect unrealistic benefits.

The single most important tool to dispel ambiguities is to practice clarity by means of two simple concepts: *distinction* and *definition*. This could also be called formalization of domain concepts. When we attack a problem, our first step is to make sure that we identify a handful of basic concepts, make clear-cut distinctions for them and provide precise definitions. In our experience, too often people assume that they are working with well-defined concepts, and that they share with their partners a common understanding of these concepts. Hence, definitions and distinctions are frequently overlooked, deemed too obvious to bother with. It is a serious mistake which has severe impacts on the course of a project.

In this paper, our purpose is to illustrate this simple idea of definition and distinction on typical examples from IT innovation and improvement. We shall provide definitions mostly using plain natural language, but our work is inspired from a formal, mathematical language that we have defined over the past years and which we often use to bring additional precision in our discourse [2]. In this paper, we shall use some of its constructs when necessary, to illustrate the concision and clarity brought by simple mathematical operators. This formal language is supported by a parser and an interpreter which allows one to build formal glossaries which can be queried and executed. The language is inspired from logic and functional programming. The interpreter answers queries by applying definitions to rewrite queries until a normal form has been reached. Normal forms are either Boolean terms or terms from some universe of discourse. We use a three value logic that include true, false and unknown.

Our work is also inspired from the work of George Spencer-Brown [1], who has formalized the calculus of distinctions (Laws of Form – LoF). LoF was extended by Francisco Varela to introduce a third Truth Value, to encompass the occurrence of self-referential situations. Although this work is sometimes seen as a reformulation of Boolean algebra, it also involves a strong methodological emphasis on making distinctions, a corner-stone concept of this work.

Hence, we do have means to construct formal glossaries that can be used for knowledge engineering. However, our emphasis in this paper is to show knowledge built by means of simple plain natural language definitions, supplemented when necessary with mathematical operators, is sufficient.

For illustrative purposes, the terminology of our examples is built from a subset of the glossary of the Capability Maturity Model Integration (CMMI)® [3], proposed by the Software Engineering institute (SEI). The CMMI is a process improvement integrated approach that transcends disciplines and provides organizations with the essential elements of effective processes. Although the CMMI product suite is a very strong foundation for process improvement, it is not without terminological problems that may lead to misunderstanding. As an example, in the CMMI for Services V1.2 glossary:

- The definition of 'lifecycle model' starts with "A partitioning of the life of a product, service, or project into phases."
- The definition of 'product lifecycle' starts with "The period of time, consisting of phases, that begins when a product or service is conceived and ends when the product or service is no longer available for use."

These two simple definitions prompt the following questions: What is a phase? Is it a period of time? Is the notion of time crucial, or is just the notion of sequentiality that is critical in characterizing a phase? What distinguishes a product life cycle from a project life cycle? The notion of 'phase' is not defined in the CMMI's glossary. We shall provide our own definition in this paper and try to clarify these concepts through distinctions.

Pedagogy is an essential aspect of understanding. We certainly cannot expect that throwing a bunch of definitions at someone will allow him to understand them. No one can understand physics simply by looking at all its laws. Pedagogy is an essential part of communication which we shall try to use to the best of our knowledge in this paper, but we forcefully admit that it is a goal which one is never sure of attaining.

2. LIFECYCLE

A precondition for the deployment of an IT improvement or innovation is to have a precise definition of the term 'lifecycle', applicable to projects and products.

2.1 What is a Lifecycle?

Organizations, their products and projects go through phases in their life, like living organisms in nature. Following Humberto Maturana [4], we assume that time and space are not explanatory principles in Management and Engineering. CMMI definitions hint at time in the notion of phase, but one cannot simply wait and let time pass to complete a phase in an on-going project! Time and space are basic concepts that are scientifically defined and universally measured. Time can be measured for a phase, but it does not constitute the main concept for defining a phase. We define a *phase* as a transformation of some inputs into some outputs. As such, a phase can be represented by a function f, in the traditional mathematical sense. The sequentiality of phases can be represented by function composition, typically noted "$\circ$", also in the traditional sense. If there are n phases in some lifecycle, then their composition is represented by

$$f_n \circ \ldots \circ f_1$$

Function composition also takes into account the fact that the output of one phase becomes the input of the next phase. The actual transformation carried out by a given phase need not be made explicit at this point. Knowing that a given phase is represented by some function is sufficient to precisely define the concept. For example, the lifecycle of a butterfly (metamorphosis) can be composed as follows: egg becomes caterpillar (larva); caterpillar becomes chrysalis (pupa); chrysalis becomes butterfly (adult); this can be formalized as follows:

$$\textit{butterfly} \circ \textit{chrysalis} \circ \textit{caterpillar} \circ \textit{egg}$$

Here, *egg*, *caterpillar*, *chrysalis*, *butterfly* are seen as functions: *egg* is a function with no inputs (if one want to abstract from its inception); hence, it is some constant function. In the sequel, we shall use the symbol "_" for function composition, which can be also be used on words as function's signature. Hence, our butterfly example can be formalized as:

butterfly_chrysalis_caterpillar_egg

A *lifecycle model* of a *type* of thing (a product, a project …) is a composition of phases encompassing the whole life of any *instance* of that thing.

Lifecycles models concern types, not instances; instances are concerned with their own life, not with cycles. Lifecycles are models used for planning in an organization.

Should we want to make precise the notion of *typing*, i.e. "is a", used in the definition above, we could use a mathematical notation to define its properties (for example: typing transitivity).

When producing definitions in a given context, one always have to judge the level of formality needed and determine what is supposed to be known from the reader and what isn't. This is a subjective decision, but at least one must deliberately pay attention to it, not simply overlook it. Our formal language interpreter we mentioned in the introduction can of course not afford to be given expressions containing undefined operators. Hence, there is a price to pay to use a completely mathematical notation for computerized execution. This is why we use a mixture of both in this paper.

2.2 Project Lifecycle

A project lifecycle model is a lifecycle model where the type of thing is "project". Following the IEEE/EIA 12207 standard [5], an example of project lifecycle model could consist of the following phases:

1. Feasibility Study and Planning;
2. Requirements Analysis;
3. Design;
4. Development;
5. Integration;
6. Verification & Validation;
7. Deployment;
8. Post mortem.

2.3 Product Lifecycle

A product lifecycle model is a lifecycle model where the type of thing is "product". It is a model for planning expected transformations of an on-going product.

A product lifecycle model could consist of the following phases:

1. Vision and Architecture;
2. Design & implementation;
3. Testing and piloting;
4. Operation and Support;
5. Phase out.

3. DISTINCTION BETWEEN PROJECT AND PRODUCT

3.1 Lifecycles of Project and Product Distinction

We haven't said so far what a project is and what a product is. We know what a project lifecycle is: it is simply an instance of a project life cycle model. Similarly for product. Applying our definitions, we know it will include a composition of phases, each phase being a function that transforms some inputs into outputs.

We saw that both project and product share the same notion of life cycle model. So what distinguishes them? First, they are of different types: something cannot be a project and a product; this means that we can establish a distinction between a project and a product. This is the essence of a distinction: if two concepts are distinct, then one thing cannot be an instance of both at the same time. In mathematical terms, we could say that project is a type, and represent a type by a set. Then we could state that distinction as follows:

$$\textit{project} \cap \textit{product} = \varnothing.$$

The simple notion of distinction, which most people know from common sense, can be represented in various forms in mathematics. This is another argument for the use of plain natural language for some aspects, because of its sheer concision and simplicity.

3.2 Works of Project and Product Distinction

Work products are *artefacts* created or transformed by *actors working in an organization*. Work products intermediate interactions between actors, eventually the same actor at different points of time. Work products can be developed, maintained or acquired by an organization.

A *product* is a work product that is delivered to a customer; a product is either a good or a service.

A *project* is a managed set of interrelated actors, work products and consumables. A project is never a good, nor a service. Project and product are connected in some other ways; for example: a project *delivers* a product.

Both projects and products include work products. Given a work product, it may belong to the product or the project that delivers the product.

For instance, a class diagram of some design component is a work product of a project and a product. An iteration plan is a work product of a project only; it bears no interest for the product itself.

In what follows, we make a distinction between work products that belong to the project, and work products that belong to the product.

3.3 Architecture and Work Units

Architecture is a discipline that addresses critical product qualities and design constraints; architecture provides views of the work products that belong to the product into a set of models. A product architecture model is a representation of the more structural, stable and invariant aspects of the product.

Product architecture models describe a partition of the product into *work units*.

A product P, partitioned into n work units inside a product architecture model can be formalized by a reunion operation:

$$P = (P_1 , \ldots , P_n)$$

3.4 Project and Product Arrangement

The decomposition of the product built by a project changes at each phase of the Project lifecycle, as the product evolves:

- The partition of the product into work units depends on the phase of the project lifecycle.

As mentioned earlier, a project lifecycle model is a composition of phases:

- A phase can be applied to 1 or many work units.

We can distinguish between different lifecycle models by looking at how work units are grouped and how phases are applied, which we called henceforth an arrangement.

The choice between different arrangements depends on the requirements for the project or the preferences of the organization. For instance, the following groupings, inducing different lifecycle models, are often considered:

1. waterfall model
2. iterative, i.e. time-boxed;
3. incremental, i.e. by work units that deliver value;
4. frequent customer feedback;
5. just in time detailed requirements.

The waterfall model is characterized by the following arrangement, where phases are successively applied to the single group of work units:

$$(L_m \ldots\_(L_2\_(L_1\_(P_1 , \ldots , P_n)))\ldots)$$

The incremental lifecycle model is characterized by the following arrangement:

$$(L_m \ldots\_L_1)\_P_1,$$
$$(L_m \ldots\_L_1)\_P_2,$$
$$\ldots ,$$
$$(L_m \ldots\_L_1)\_P_n$$

Interestingly, the end result is “equivalent” in either case when these functional expressions are rewritten according to the following simple laws:

1. *Associativity:* $X\_(Y\_Z) = (X\_Y)\_Z$
2. *Factoring:* $(X\_Y),(X\_Z) = X\_(Y, Z)$

After applying law #1, m-1 times, the waterfall arrangement is rewritten into:

$$(L_m\_ \ldots\_L_2\_L_1)\_(P_1 , \ldots , P_n)$$

After applying law #2, n-1 times, the incremental arrangement is rewritten into:

$$(L_m\_ \ldots\_L_2\_L_1)\_(P_1 , \ldots , P_n)$$

Each phase has been applied to the initial work units and their resulting work units. Of course, typically, the phases of a waterfall model are not exactly the same as those of an incremental lifecycle model. Moreover, the result of applying

$$(L_m \_ \ldots L_1)\_P_1,$$

has a influence on the choice of P_2. The feedback obtained from the user after producing the output of all phases on P_1 allows software developers to refine the selection or definition of the work unit P_2. In the waterfall model, the sequence of function application evaluation does not allow such refinement.

Modeling phases as functions allows a simple reasoning of the behavior of phases, and to reason about them. It also offers a concise and crisp description of what a lifecycle model is all about.

In software engineering theory, there exists well known lifecycle models and it is often assumed that they are reused as is on projects. The reality is far from that. Each project is almost unique. Lifecycle models are never applied as is, but adapted to the given context of a project. Each project may deserve its own arrangement.

When a new project is planned in a large organization, people from different backgrounds (external consultants from various companies, internal people from various department), are grouped together for the duration of a project. Hence, definitions of basic concepts must be made precise, to guarantee some cohesion in the project team. Thus, it is often assumed that everybody in the team has the same vision of the lifecycle, because model from the literature are considered to be applied everywhere the same way. In practice, each individual has its own interpretation of the standard models according to his personal story.

4. HOW TO TRANSFORM AN ORGANIZATION BY IT INNOVATION?

4.1 How to deploy an IT innovation?

Deploying an IT innovation in an organization is a significant challenge which must be addressed in a methodical manner. An approach which is commonly suggested is to build a “model” of the enterprise and use it to plan the implementation of the innovation. For instance, suppose a large bank wants to use a new security architecture. Then, in this case, what is a good model of the bank? It is pointless to start modeling the bank for that purpose. Rather, what is needed is a model of the things (products) that will be impacted by this new innovation and a model of the way (projects) this new innovation will be deployed. Thus, this is why an organization needs good maps of its existing products and projects. Now, imagine the number of work products that one has to analyze to conduct the impact analysis. Which work products should be analyzed? Work products of the products will be analyzed to see how the new security architecture can affect them. Work products of the on-going projects will be analyzed to see the impact of changes on costs, schedules, and detailed plans. These observations stress the importance of distinguishing between projects and products, and work products belonging to the projects and work products belonging to the products.

The project responsible for introducing the IT innovation will have to be structured in a very specific way, according to the impact analysis conducted. Its lifecycle model may be very different from the existing projects’. This again stresses the fact that the arrangement of a project is very specific, but surely inspired from generic project lifecycle models.

4.2 What is the real weight of deploying an IT innovation?

Table 1 (see next page) provides a structured list of all activities for conducting the impact analysis on an IS, based on the distinction between projects and products.

The real weight of deploying an IT innovation is the sum of the unitary weights of ***all*** *these activities and their interactions.*

This model is well adapted for Business Information Systems, but not for real-time systems or embedded systems, because they require a different product lifecycle model.

5. CONCLUSION

In this paper, we have presented an approach to technology evolution which is based on simple concepts like definition and distinction. We advocate that generic models often cited in the literature cannot be reused as is in an organization, but must always be interpreted and adapted, as it is stated in a CMMI-based approach. In order to avoid ambiguities, confusion and misunderstanding, an organization must build its own models using simple definitions. These will help it in implementing improvements and innovations, by giving the ability to conduct proper impact analysis. We favor the use of mathematics whenever it can help in dispelling ambiguities.

Table 1 - Impact analysis activities for an IT innovation

Product	Project
Vision & Architecture • Modeling Components Separation induced by the candidate innovation • Modeling Components Reusability induced by the candidate innovation **Design & implementation** • Mapping existing systems concerned by the candidate innovation (IT infrastructure, applications and data bases) • Mapping existing business concepts concerned by the candidate innovation (Data & Processing) **Test & pilot** • Test cases dedicated to the candidate innovation **Operation and Support** • Models of Components needed for the candidate innovation **Phase out** • Models of Components needed for the candidate innovation	**On-going Projects Deployment** • Impact Analysis of the candidate innovation **Starting-up Projects Deployment** • Impact of the candidate innovation on the Project planning phase • Impact of the candidate innovation on the Requirements Analysis phase • Impact of the candidate innovation on the Design phase • Impact of the candidate innovation on the Development phase • Impact of the candidate innovation on the Integration phase • Impact of the candidate innovation on the Verification & Validation phase • Impact of the candidate innovation on the Deployment phase • Impact of the candidate innovation on the Post mortem phase

Using definitions and distinctions is certainly not a new approach. It is the basic foundation that any systematic endeavor typically uses, like in science and engineering. However, our experience is that it is often overlooked by IT practitioners.

Our approach to innovation can be contrasted with other approaches like enterprise architecture (EA). Like us, EA favors the use of models and proposes a process to conduct IT innovation. EA architecture frameworks like TOGAF [6] provide a comprehensive set of generic models and methods. Any potential user of TOGAF must pick what is needed in this framework (it is too large and too generic to be used as is). This is where we fit, by stating that the basic concepts of these models should be adapted to the context of an organization and well-defined, by applying definition and distinction.

REFERENCES

[1] George Spencer-Brown. *Laws of Form*. Allen and Unwin, London, 1969.

[2] Jean Maynier, Modélisation des liens sémantiques d'un langage métier, M.Sc. Thesis, Dept. Of Computer Science, Université de Sherbrooke, 2003

[3] CMMI® for Services, Version 1.2, Software Engineering Institute, Carnegie Mellon University, USA.

[4] Humberto Maturana. The nature of Time, http://www.inteco.cl/biology/nature.htm

[5] Standard for Information Technology-Software Life Cycle Processes, IEEE press, 1995.

[6] TOGAF v9, The Open Group Architecture Framework, The Open Group, 2009.