Towards Measurable Process Models

Tineke de Bunje¹, Gregor Engels², Luuk Groenewegen²,
Michael Heus², Aart Matsinger¹

¹ Philips Research Laboratories, Prof. Holstlaan 4, NL-5656 AA Eindhoven
² Leiden University, Dept. of Computer Science, P.O. Box 9512, NL-2300 RA Leiden

1 Motivation and Background

Due to high market pressure, development of industrial software has to become cheaper, more flexible, more reliable, and more efficient. Thus for all software development projects, improvement of the software development process has become an important objective. In order to be able to speak about and to reach improvement, two prerequisites have to be fulfilled. First, explicit models of (parts of) the software process are needed, and second, metrics have to be defined to get concrete, comparable values about the quality of the software development process.

The need for, and the benefits of, combining process models with metrics has been stated in several publications ([1, 2, 7, 9, 11]). In particular, successful experiences with the combination of the so-called Goal-Question-Metrics (G-Q-M) paradigm and process modelling are described in e.g. [10] and [4].

This position paper continues and exceeds substantially [4] by making their approach much more concrete. We here report about a combination of the G-Q-M paradigm with the SOCCA approach ([6]) to modelling of software processes. This has been applied to modelling, as part of a complete software process, the configuration management within a large software project at a product division of Philips in Eindhoven (NL). The achieved measurable SOCCA model has been analysed by the analysis and simulation tool HIT (Hierarchical Evaluation Tool) ([3, 5]), which was originally developed for performance evaluation of computing and communication systems.

In chapter 2 we will explain our approach to reach measurable SOCCA models. Chapter 3 summarizes the setting and the results of an experiment executed at a Philips product division. The lessons learned as well as our position statements about the usage of measurable process models finish this paper. A more detailed description about the topics discussed here, can be found in [8].

2 Measurable SOCCA Models

Modelling a software process means to abstract from real-world situations and to identify the objects of (current) interest, their allowed behaviour as well as their allowed (coordinated) interactions. Objects of interest are, for instance, human as well as non-human process agents, process documents and process activities.
Process modelling languages (PMLs) offer language constructs to define these objects, their behaviour and their interactions.

For instance, the PML SOCCA provides object-oriented class diagrams and extended forms of state transition diagrams (cf. [6]) to model all aspects of a software process. In particular, state transition diagrams are used to model the external behaviour of an object, i.e., the allowed sequences of operation calls to an object, and to model the internal behaviour of an operation, i.e., the possible sequences of calls to operations on other objects or the object itself. All use, i.e., import relations between classes are described by an interaction diagram.

The choice of objects of interests as well as the level of detail within a process model are determined by the process modeller, based on the goal for which the process is being modelled. First experiences with development of measurable models show that this meta (modelling) process has to be closely coupled with the development of customized metrics, cf. [4] and [10]. In [4], De Bunje and Saunders created the term Sandwich Approach for an integrated, combined top-down and bottom-up approach for developing a measurable process model based on the G-Q-M paradigm. To give a global impression, figure 1 shows the steps of the top-down part of the Sandwich Approach, where in this case SOCCA has been chosen as process modelling approach.

![Diagram](chart.png)

**Fig. 1. Top-down part of the meta process**
The steps are:

1. Establish primary goals for the process
2. Make high-level description of the process
3. Formalize goals, questions, and metrics
   loop
   4. Refine the description of the process
   5. Refine goals, questions, and metrics
   end loop

The result of this meta process is a SOCCA model enhanced by a number of metrics. Metrics are coupled with object classes or activities of the SOCCA model, and evaluation rules are given how to compute concrete values for them. Depending on the kind of metrics, the computation is static or dynamic. A static computation is a computation in one go, based on an analysis of the model but independent from the model's execution. A dynamic computation takes the model's execution into account and can be either a computation in one go or an incremental computation. A one-go computation is based on an analysis of this execution without really performing the execution. In case of an incremental computation, the steps of it are coupled with (the) steps as occurring in the model's (simulated) execution.

For dynamic computations, based on SOCCA model's execution, an evaluation tool is of great benefit. The analysis and simulation tool HIT ([3, 5]) has been chosen, as it allows a straight-forward translation of a SOCCA model into a HIT model. A HIT model consists of a definition of components and exported services for each component. It is indicated for each service, which other services are used to realize it. All these informations can directly be derived from the class diagram, the interaction diagram as well as the internal behaviour STDs from the SOCCA model. The realization of services is described in HIT in a pseudo-code like language HI-SLANG. In addition to usual control flow constructs, HI-SLANG offers e.g. features to specify probabilities for control flow decisions, scheduling of called services, or global parameters to evaluate the model under different situations. Also the realization of services can be directly derived from the internal behaviour STDs of the SOCCA model.

HIT provides a development component for defining a (translation of a SOCCA) model with its metrics incorporated. Furthermore, HIT provides an evaluation component for either analysing or simulating - i.e. executing - the model together with the metrics computation. In this case, the HIT model, including all HI-SLANG procedures, is automatically translated into a SIMULA program.

3 Practical Experiences

The above described meta process for developing a measurable SOCCA model has been validated by modelling a specific part of a concrete software process at a product division of Philips. The tool-assisted configuration management process within a multi-site software development project has been chosen as case study. In this project, all source files are stored in a central source file version container.
All source files within a developer's configuration are checked out and thus are copies of sources files from the central container. A librarian is responsible for maintaining a so-called latest accepted configuration, which is built of newest versions of all source files.

Parts of the configuration management are supported and automatically done by a configuration management tool (called CMD), while other parts are done manually, i.e., are explicitly activated by the librarian or a developer. It was felt by several project members that more manual actions should be automated by extending the functionality of the CMD tool. Thus, the question has been raised to measure the level of automation of the configuration management process, since you can't see whether you have improved without knowing where you are at the moment.

Within a first (let's say naive) approach it was tried to extend a formerly made SOCCA model by appropriate metrics. It became obvious that this was impossible because in this model the CMD tool was not explicitly modelled as a class on its own.

Therefore, a second SOCCA model has been developed by following the meta process explained above. In this case, the CMD tool appeared as a class in the class diagram and metrics could be defined and attached to measure the frequency of calls to the CMD tool.

For evaluation purposes the resulting SOCCA model has been transformed into a HIT model. By using realistic data from former projects for e.g. the number of (versions of) source files or number of developers, the HIT environment calculated the metrics. Finally, the level of automation, i.e., the number of calls by the tool divided by the total number of calls, was calculated as 98.5%. This seems already a high value, considering that the aim was to increase the level of automation.

A critical review of this experience showed that the reasons for this "unexpected" result are a too detailed modelling of the behaviour of the CMD tool while ignoring (or abstracting) a great part of the behaviour of humans. Moreover it became clear from the review, that a further understanding of the goal(s) was needed in order to react properly to the metrics obtained. Such a review then leads to further iterations of the loop in the above meta process, or to some form of backtracking of it.

4 Conclusions and Statements

Summarizing we come to the following conclusions. The followed approach leads to a customised model in relation to the defined goal. Moreover it provides a set of relevant metrics in a natural way. Since SOCCA models can be translated into HIT models relatively easy, HIT evaluation can be applied successfully in order to obtain actual values for the defined metrics. This technique can be applied effectively in what-if analysis. Although in this case the approach has only been applied for a low-level goal, it seems also valid and promising for more large-scale goals. This is due to further investigation.
The lessons learned in applying the described approach towards measurable process models can be summarized as follows.

1. Carefully identify the goals of the modelling process in the early stages, as these will heavily influence the resulting model.
2. Develop the process model in close cooperation with the process owner in order to be sure to reach the intended goals, and in particular
3. determine the level of detail of parts of the software process model carefully.
4. Follow an integrated approach for developing measurable process models, and
5. be careful with the (right) interpretation of modelling and evaluation results.

At the same time, these lessons are to be viewed as our position statements aiming at reaching really suitable, measurable process models in the future.

References