# Towards Guidelines for a Development Process for Component-Based Embedded Systems

Rikard Land, Jan Carlson, Stig Larsson, and Ivica Crnković

Mälardalen University, School of Innovation, Design and Engineering, Västerås, Sweden
{rikard.land,jan.carlson,stig.larsson,ivica.crnkovic}@mdh.se

**Abstract.** Software is more and more built from pre-existing components. This is true also for the embedded software domain, and there is a need to consider how development processes need to be changed to best utilize the component-based paradigm, and how processes and technologies must be designed to support each other. To facilitate this change towards component-based embedded software, this paper presents a set of process guidelines, named the Progress Process Guidelines (PPG), which is based on the structure of CMMI. This paper presents the structure of the PPG, and presents and analyzes the PPG parts which most closely relate to system verification, which is typically an important and difficult activity for embedded software.

## 1 Introduction

It is truly a complex task to develop complex, distributed embedded systems. The challenges lies both in technology and tool support, but also in process aspects such as parallel development by distributed teams, short development cycles, reuse of existing system parts of varying quality, and various business relations such as subcontracting and the use of COTS. As new technologies and new paradigms arise, there is a need to evolve existing process paradigms by considering their assumptions. The component-based paradigm, i.e. the approach to build systems out of strictly separated components, is such a new paradigm which introduces new challenges and possibilities. How should a component-based development process and accompanying tools be designed to best be able to e.g. monitor and follow up your projects, minimize rework, increase parallelism, and ensure a high quality of the system? This is the question this paper is answering in part.

The present paper is part of a large research effort[1] where the state of the art in development of embedded systems is advanced by adopting the component-based approach. The research spans component models [1], verification methods through e.g. model checking and static analysis [2][3][4], and runtime support [5][6], all implemented as tool support. As a part of this effort, the *Progress Process Guidelines* (PPG) are being developed, which are intended to guide the design of

[1] See http://www.mrtc.mdh.se/progress

industrial-strength processes utilizing the novel technologies in order to form an efficient, effective, and predictable development process for predictable, component-based embedded systems. The contribution of this paper is 1) to describe the basic structure of the PPG, in particular in relation to CMMI, 2) to describe the information and activities of PPG being central for system verification, and 3) to outline the requirements on tool support required for a component-based development process for embedded systems, provided the necessary tool support exists.

Related work is first presented in Section 2, followed by an introduction of the core PPG concepts in Section 3. These are then related to the CMMI in Section 4, and illustrated with an example process in Section 5. Further implications of the PPG are briefly discussed in Section 6, and Section 7 concludes the paper and outlines future research.

## 2 Related Work

For component-based processes in general, little has been written [7], and nothing as comprehensive as the proposed PPG. In model-driven development (MDD) [8], the scarce publications on processes discuss roles in terms of a meta-team of with language developers and code generator developers [9][10]. MDD relates to the verification-intensive paradigm of PPG, but while MDD relies on forward engineering in order to produce correct software, the PPG permits components produced in many different ways, including the wrapping of legacy code. The exploratory analysis and milestone verification presented in this paper inherits the basic ideas from the concepts of daily builds, continuous integration, continuous verification, and test-driven development [11][12][13][14], and adapts them to fit the component-based approach.

The PPG is currently adapted to ProCom, but the ideas should be general enough to apply to other component models with similar characteristics. A prerequisite is that there is support for compositional reasoning of the attributes, and also that there is a strong concept of component identity from early design to run-time; the mapping from PPG to other component models and UML extensions for this domain such as AADL [15], Autosar[2], SysML[3] and MARTE[4], remain as future work.

There exist few process descriptions which explicitly utilize the structure of CMMI, CMMI+SAFE being the notable exception [16].

## 3 Scope of PPG and Core Concepts

The intended scope of PPG are activities “close to the technology”, i.e. including product requirements but excluding negotiations with stakeholders, including design

[2] http://www.autosar.org/
[3] http://www.sysml.org/
[4] http://www.omgmarte.org/

and implementation, and including verification with respect to specified requirements but excluding validation with respect to real users' needs.

## 3.1 The Component-Based Paradigm

With a component-based approach, the software is designed as components which have clear boundaries, and which interact only through explicit interfaces. This has been successful in e.g. the desktop domain, and has also partly found its way into the embedded systems domain [17][18][19]. From a process perspective, this means the processes of component development and system development are treated separately, but interact [20]. Component development could be a result of system top-down decomposition, and result in either internal development or in hiring a subcontracting. A system may also be built from pre-existing components, such as Off-the-Shelf (OTS) components (components developed for the marketplace), or as part of a product line initiative [21]. The PPG currently utilizes the Progress component model *ProCom* [22]; the generality of PPG with respect to other component models was discussed in Section 2. Semantically, a "system" is a special case of a "component" only in that it is the root node of a hierarchical tree.

From the process point of view, the following aspects of a component are of importance:

- **Composition.** A component may be implemented as a structure of other components.
- **Various other types of implementation.** As realistic problems are being addressed, components are allowed to be of any type of implementation – or unknown type of implementation – such as pre-existing COTS and reuse of legacy subsystems not built with Progress technologies. However, in these cases there are limits as to what can be predicted in general.
- **Attributes.** The concept of attributes is a general and extensible mechanism used to attach information to components. Such information can be technical features like "maximum memory consumption" but also "vendor name" for a COTS component. The type of values may range from simple values and strings, to complex models, such as a timed automaton describing the functionality. (Although this paper mainly focuses on attributes of components, other types of entities in ProCom may also be attributable are component ports, connections, networks, etc. [23].)

## 3.2 Conceptual Product Meta-model

To be able to describe activities, PPG defines a conceptual product model of the information needed, on which the activities operate. To support the process in reality, this model is easy to implement in tools, so that the information about components etc. are stored, updated and managed throughout a project. The practical challenges for tool support are further discussed in Section 6. Fig. 1 describes the information needed to explain the activities in this paper: each *Component* may have a number of *Attribute*s. Each *Attribute* has an *id* which stands for an attribute type such as "maximum memory consumption". Each *Attribute* may be assigned several values, which is

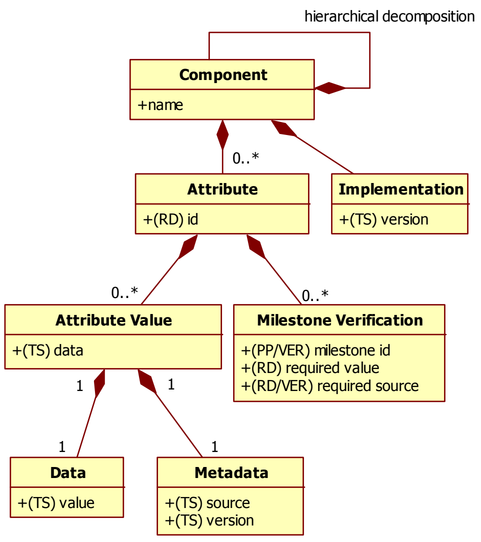

**Fig. 1.** The PPG conceptual product model, showing the concepts relevant for verification

modeled with the concept *Attribute Value*, each with a *data* (e.g. “64 kb” or “61 kb”), a *source* which specifies how the value was obtained, such as “Expert estimate”, “Analysis tool XYZ”, or “Measured”.

For project management to keep track of project progress, a number of *Milestone Verification*s are associated with each *Attribute*. Here is specified a *required value* (of for example “64 kb”) and a *required source* describing how the value must have been obtained in order for the verification to be considered successful. For an early milestone, there may be a required value of “64 kb” as estimated by an expert, and for a milestone half-way into the project there may be a milestone specifying “32 kb” as measured, since this milestone will contain an implementation of half of the features.

Also, there are *version*s and *milestone id*s, concepts which are here used rather informally to denote the need of keeping track of how values are associated with some version of the component. For example, measured values need to be associated with the version of the component used during the measurement. More complicated to model is an early estimate of the final component. These issues are solvable with sophisticated enough configuration management (CMMI process area “CM”), and will be modeled with explicit concepts in further development of the PPG. For now these issues are therefore left out of the discussion.

Once again, here is only presented the small part of the complete PPG which is relevant for this paper; for example, “milestone” is also a first-class concept, the units of attribute values (e.g. “kb”) are modelled in a more sophisticated manner, “measured” will need to be further specified with e.g. number of runs and input profile, etc. Two notes on the notation: first, the prefixes PM, RD, TS, and VER refer to CMMI process areas. This notation indicates that updates of the (UML) attributes of the concepts are within the responsibility of the indicated process area, as will be further explained in the next section. Second, the conceptual model should not be mistaken for an implementation class diagram; we believe that in practice the information will be distributed over several tools, i.e. a requirement management tool is used to store and manage concepts and attributes prefixed with RD, etc.

These concepts will be explained with a simple development scenario in Section 5, but first, the activities of the PPG are described in relation to CMMI.

## 4 PPG in Relation to CMMI Process Areas

CMMI for development [24] is a well-known reference text which defines a number of good practices for software organizations to perform. Its development was initialized in order to improve the quality and timeliness problems experienced in software development projects for complex embedded systems.

The CMMI does not prescribe a *process* in the sense *how* things should be done, but rather defines criteria according to which actual processes can be evaluated. It is organized as a number of *process areas* with suggested good *practices*. The CMMI is perhaps most known in its so-called *staged representation*, which is used to classify organizations into *maturity levels*, but more recently the CMMI also exist in a *continuous representation*, which describes the process areas in four *categories* instead of *levels*. However, the process areas themselves are the same in the two representations, and PPG does not prefer either representation over the other. For pedagogical reasons however, the four categories of the continuous representation are briefly described.

The categories are: *Project Management*, *Process Management*, *Engineering*, and *Support*. Of these, the paper mainly focuses on *Engineering* (in Sections 4.1 through 4.4), but also describes how these process areas interact with the *Project Management* process areas (in Section 4.5) and the *Support* process areas (Section 4.6).

The PPG consists of an augmentation of some goals and practices of some process areas, by adding some *subpractices* and/or *amplifications* [24]. Fig. 2 describes schematically how the PPG augments some elements of CMMI which are used to design concrete organizational processes. The rest of this section describes the PPG activities per each of the following process areas (in alphabetical order): *Requirements Development* (RD), *Technical Solution* (TS), *Product Integration* (PI), and *Verification* (VER), and then briefly describes the rest of the process areas in the *Project Management* category.

### 4.1 Requirements Development (RD)

"The purpose of Requirements Development (RD) is to produce and analyze customer, product, and product component requirements" [24]. Within the focus of this paper, the PPG adds the following RD subpractices or amplifications:

- RD1. Specify which attributes for components on which there are requirements. (This corresponds to instantiating an *Attribute* in Fig. 1 and providing a *Name*.)
- RD2. Specify a *Required Value* for the attribute.
- RD3. Together with VER and PP, specify *required source* for *Milestone Verification*s. (See VER1 in Section 4.4, and Section 4.5.) Also included, but not shown explicitly in the model, is to add some additional conditions, such as for an early analysis of design models, that the design models must be "detailed enough", meaning that the component structure has to be "fine-grained enough".

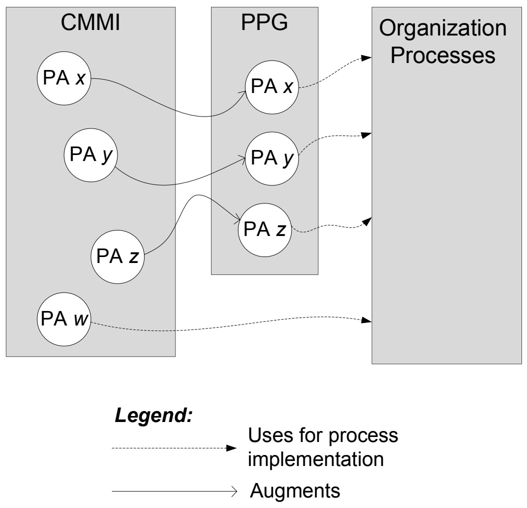

**Fig. 2.** PPG in relation to CMMI and concrete organizations and projects

**Note:** First and foremost, requirements will be specified for the "root component", i.e. the system, and its immediate children, but may also be specified for constituent components, in interaction with design work, i.e. the process area *Technical Solutions*.

## 4.2 Technical Solution (TS)

"The purpose of Technical Solution (TS) is to design, develop, and implement solutions to requirements" [24]. Within the focus of this paper, the PPG adds the following TS subpractices or amplifications:

TS1. Create an *Implementation* of the component (refer to Fig. 1), which automatically gets a *version*.

TS2. Assign values to attributes, by instantiating a new *Attribute Value* and providing its *data*; it automatically gets a *version* and *source*.

TS3. Perform *exploratory analysis* at any time, by comparing any value for attributes with their required values, using any information in the conceptual model available at the time. See Fig. 3.

**Note on TS1.** The implementation could be a subdivision into components, or a "primitive" component of some kind: COTS or some other package from a third party, imported legacy code, or a small, relatively simple implementation of C code (which is how the primitive components in ProCom are implemented).

**Note on TS2.** The assignment of values to attributes could be of several kinds, for example explicit and manual (such as when an architect enters an expert estimate) or explicit and automatic (such as when someone executes an analysis tool on a component), or possibly implicit and automatic (such as when a composite component is analyzed with a tool which traverses each subcomponent and assigns attribute values before composing these values for the containing component).

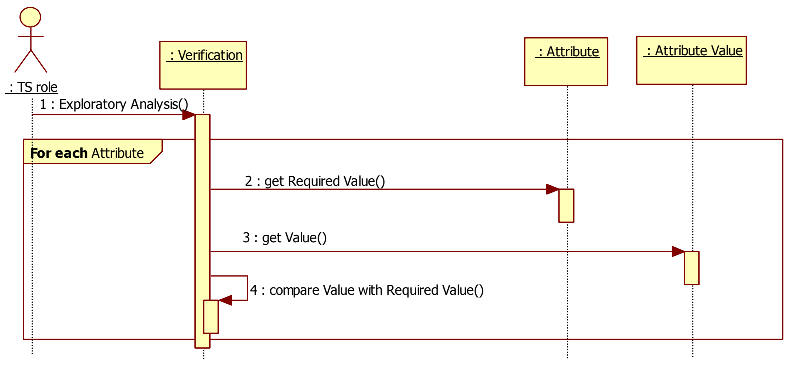

**Fig. 3.** Exploratory Analysis

**Note on TS3.** Exploratory analysis can be seen as more relaxed version of milestone verification (which is the responsibility of VER). Current values of attributes, which may be any combination of estimates, values produced by tools based on incomplete implementations, etc. may be used exploratory to answer questions like “how near the limit for memory consumption are we right now?” There are no absolute guidelines how to perform this type of analysis; in some cases it is reasonable to use the average of several values, or use the most pessimistic of all values for an attribute, or add 20% to measured values to add a safe margin, or because some features are not implemented yet, or decrease 20% from some measured values because it is believed that low-level optimization at the end will achieve this. (It is exactly all these decisions are formalized into milestone verification; see further VER2 in Section 4.4.)

The TS process area is where the main part of the research of the Progress centre lies, and here only the high-level approach is outlined; ideally, for each attribute there exist a way to analyze a primitive component, as well as a composition theory or analysis technology [25]. For example, a primitive component may be analyzed regarding “maximum memory consumption” through static analysis, and/or measurements. It is essential that the developers are aware of alternative methods’ assumptions and limitations; a static analysis usually gives safe but too pessimistic upper bounds, while measurements may give average values and upper bounds which are not safe if total predictability is required. All these issues have to be resolved by the team together, mainly it is the responsibility for TS and VER.

## 4.3 Product Integration (PI)

“The purpose of Product Integration (PI) is to assemble the product from the product components, ensure that the product, as integrated, functions properly, and deliver the product” [24]. This process area concerns how to integrate product parts into a system. It can be expected that any component-based approach, with proper tool support, makes integration a less effort-consuming task [26]. The user will be aware of incompatible components already during design, indeed with a proper tool it is not even possible to connect incompatible interfaces. Similarly, missing connections are detected interactively during design and implementation. In short, some integration

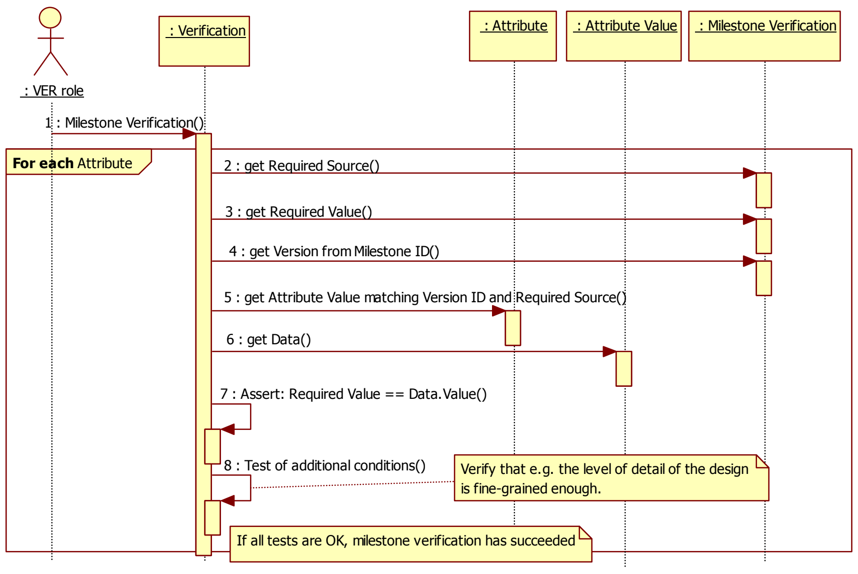
: VER role
: Verification
: Attribute
: Attribute Value
: Milestone Verification
1 : Milestone Verification()
For each Attribute
2 : get Required Source()
3 : get Required Value()
4 : get Version from Milestone ID()
5 : get Attribute Value matching Version ID and Required Source()
6 : get Data()
7 : Assert: Required Value == Data.Value()
8 : Test of additional conditions()
Verify that e.g. the level of detail of the design is fine-grained enough.
If all tests are OK, milestone verification has succeeded

**Fig. 4.** Milestone Verification

activities are covered in the *Technical Solution* process area, and the *Product Integration* process area will require less effort. Also important, it can be expected that less errors are discovered during integration since the system to a large extent should be correct by construction.

However, this said, there will still be many qualified integration tasks to do, which however may be performed earlier in the development process. For example, the deployment to hardware is not addressed in the current version of PPG, nor is the integration of ProCom parts with legacy systems, nor is it clear how the process needs to consider the relationship to software platforms such as operating systems.

## 4.4 Verification (VER)

"The purpose of Verification (VER) is to ensure that selected work products meet their specified requirements" [24]. Within the focus of this paper, the PPG adds the following VER subpractices or amplifications:

VER1. Together with *Requirements Developer*, specify *required source* for *Milestone Verification*s. (See Fig. 1; also refer to RD3 in Section 4.1).

VER2. Perform *milestone verification* as planned by PP. See Fig. 4.

**Note on TS3 and VER2.** To keep the sequence diagrams simple, only the main flow is shown. For example, if some test fails as much information as possible should be shown to the user, or, at least, made available, and it may or may not make sense to continue traversing other attributes.

**Note on TS3 and VER2.** To keep the sequence diagrams simple, it is assumed that the correct attribute value is used, out of the potentially many with different *sources* and *versions*. However, this is not a trivial assumption and is further discussed in Section 6.

### 4.5 The Project Management Category

The above approach enables projects to define requirements and milestones precisely, in terms of attributes of components and their verification. These are apparently intended to be used to monitor and control projects. Although outside the scope of this paper, it can be mentioned that the PPG further describes how this information is used in the *Project Management* category, in summary as follows:

- **Project Planning (PP).** One additional subpractice or amplification is defined: Define major and minor milestones for the project. (This corresponds to instantiating a number of *Milestone Verification*s in Fig. 1 and together with RD, VER, and TS, specify *required source*. (See RD3 in Section 4.1 and VER1 in Section 4.4.)
- **Risk Management (RSKM).** The possibility for almost continuous automatic verification is one method of reducing risks in the project.
- **Project Monitoring and Control (PMC).** Monitor whether milestone verifications have been conducted as specified, and whether the results are satisfactory. (See more details under VER2 in Section 4.4.) Otherwise, some corrective action must be taken, which may e.g. involve renegotiating requirements, schedule, or product features. This involves interaction with the appropriate process areas, but is not is not further elaborated here.
- **Quantitative Project Management (QPM).** The data collected during verification activities help to quantitatively answering questions about the project status.

### 4.6 The Support Category

In the *Support* category, the following process areas also interact heavily with the approach outlined above:

- **Configuration Management (CM).** The approach requires, as described, a mature use of configuration management to keep track of correct versions of the various artifacts.
- **Process and Product Quality Assurance (PPQA).** The whole verification-intensive approach is clearly intended to increase product and process quality, as well as provide the visibility necessary for efficient quality assurance.

## 5 Example

This section uses a simple example to illustrate how the PPG is instantiated in a concrete process. The system to be developed in the example is a refinement of the one found in the ProCom reference manual [22], which is an electronic stability control (ESC) subsystem of a car. Two requirements on the system (as specified in RD) are used to illustrate the use of the PPG: functionality and static memory consumption. Three milestones are defined in the project plan: MS1, MS2, M3 (all features

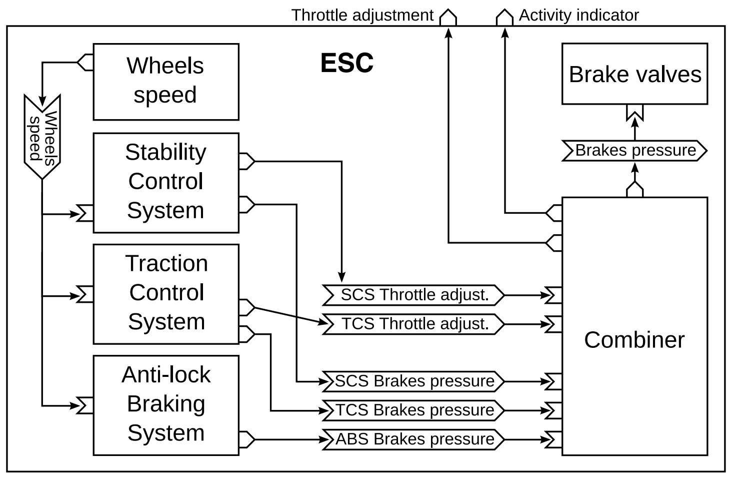

**Fig. 5.** Component design of an electronic stability control (ESC) subsystem of a car

implemented) and MS4 (the finished system). Given the toolset available, specification methods for the attribute values are specified for each milestone, as well as the target value. It can be noted that in some cases, the required value of the milestone is defined as the required final value, but in some cases it makes sense to specify another figure. Table 1 displays all this information; due to space limits of the paper, this single table thus lists the information from several of the concepts of Fig. 1.

Initially, an architect sketches the division of the system into components (process area TS) as depicted in Fig. 5. Together with developers or other experts the memory requirements is divided into budgets for each component, as listed in Table 2. The architect now performs an exploratory analysis, as was described in Fig. 3, to verify that these values in fact compose to the target value for milestone MS4. The composition formula for static memory consumption is (in this simple example) the sum of the static memory consumption of each ingoing component instance, plus 10% communication overhead. Using the values of Table 2, the computed static memory consumption is 810 kb, which is less than the maximum value required of 1024 kb, and the

**Table 1.** Requirements on the ESC system

| Attribute Name | Milestone ID | Required Value | Source |
|---|---|---|---|
| Functionality | **MS1** | (same as MS4) | Manual Review |
| | **MS2** | (same as MS4) | Composition of design models |
| | **MS3** | (same as MS4) | Analysis of implementation |
| | **MS4** | Behavior specified by a timed automaton (not shown here) | Analysis of implementation |
| Static memory consumption | **MS1** | (same as MS4) | Expert estimate of final system |
| | **MS2** | Max 768 kb | Static analysis (only partially implemented) |
| | **MS3** | (same as MS4) + 20% | Static analysis |
| | **MS4** | 1024 kb | Static analysis combined with measurements |
| … | … | … | … |

**Table 2.** Initially estimated memory consumption values for ESC and its subcomponents

| Component Name | Data | Source |
|---|---|---|
| Wheels speed | 64 kb | Expert estimate |
| Stability Control System | 192 kb | Expert estimate |
| Traction Control System | 256 kb | Expert estimate |
| Anti-lock Braking System | 128 kb | Expert estimate |
| Combiner | 32 kb | Expert estimate |
| Brake Valves | 64 kb | Expert estimate |
| ESC | 810 kb | Composition of expert estimates |

architect is satisfied. Furthermore, this exploratory analysis qualifies as verification of milestone MS1, unless there are any additional conditions specified regarding the level of detail and granularity of the design model used as a basis for the estimates.

Some possible further events and decisions made are outlined below, in order to illustrate various scenarios and how these are captured and supported by the PPG:

- For the *Wheels speed* component, there are three potential COTS components available. These are investigated and one of these is chosen. If the component comes as a white box, i.e. with all desired information, including source code, it can be used for the static memory analysis required in some of the milestones. The same is true if the component is packaged as a black box but comes with models of its memory usage, which can be used in the composition theory. (This also requires there are certification mechanisms in place, so that the legal implications are clear regarding the extent to which the system development organization can trust these assertions.) In a less than ideal world, the component comes as a black box and with insufficient information, in which case the system development organization has to rely on thorough testing. In this case, the type of specification methods required in some milestones has to be re-negotiated – or, if measurements are not considered safe enough, the COTS will have to be replaced.
- The *Stability Control System*, *Traction Control System*, and *Combiner* require new development. Of these, *Stability Control System* is outsourced to a subcontractor, while the others are implemented internally. The *Combiner* is straightforward to implement, and implementation is finished within a month, well before milestone MS2.
- The *Anti-lock Braking System* and *Brake valves* will be reused from the previous generation of the car. No modifications are needed, other than those required to function in the ProCom environment. However, it may happen that the source code does not follow some required restrictions by the memory usage analysis tool, and for these components the verification method also has to be re-negotiated, with either the result that for example measurements is a qualified specification method, or that the source code has to be modified so as to fulfill the requirements of the analysis tool, or that the component will be re-implemented completely in ProCom (reusing the previous design). In the

example, assume that *Brake valves* may be statically analyzed, while the *Anti-lock Braking System* cannot but will be wrapped as a ProCom component with as small modifications as possible nevertheless, and the project will therefore have to rely on measurements of this component.

During development, developers and the architect perform exploratory analysis whenever they need to explore some "what-if" scenarios (like "would it be fine to implement an algorithm that executes faster but requires more memory?") or to assure themselves that the current state of the work products will pass the next milestone.

Table 3 lists the values of the memory consumption attribute at the time for milestone MS2. Using these values, the computed memory consumption is 551 kb, which is well less than 768 kb as specified for MS2 in Table 1. However, the specification method is for three components not the required ("measurements" instead of "static analysis"), but this is well motivated and has been approved (and Table 1 updated accordingly) by project management, architects, and verifiers. Also important to consider is the actuality of the figures used: since the *Traction Control System* is lagging behind in implementation, the value 124 kb does not accurately reflect the contents of the system as envisioned for MS2 and should be adjusted for this (probably by adding an expert estimate using the value 124 kb and estimate the memory consumption of the features not implemented in the current version).

Other events that may occur are that during exploratory analysis, or in one of the milestone verifications, the composed values are more than 1024 kb. Clearly, the plan must be changed, either by dropping some functional requirement, or by allocating more memory (which may in some domains with large product volumes be totally infeasible), or by putting more effort into optimizing the memory usage by some algorithms (which may have effects on the time and staffing plan). Hopefully, the possibilities of continuous exploratory analyses, and the possibility to formalize these into milestone verifications to provide process visibility, will ensure that problems like these are discovered as early as possible. But in the end, all plans and estimates are only as good as the people behind them.

**Table 3.** Memory consumption values for ESC subcomponents

| **Component Name** | **Data** | **Source** | **Remarks** |
|---|---|---|---|
| Wheels speed | 63 kb | Measurement | COTS |
| Stability Control System | 98 kb | Static analysis | Partially implemented, according to plan |
| Traction Control System | 124 kb | Static analysis | Partially implemented, lagging behind plan |
| Anti-lock Braking System | 128 kb | Measurements | Previous generation, have not been fully wrapped as ProCom component |
| Combiner | 24 kb | Static analysis | Fully implemented |
| Brake Valves | 64 kb | Measurements | Wrapped as ProCom |
| ESC | 768 kb | Composition of measurements and static analysis | (Inherits every weakness of ingoing composed values.) |

## 6 Open Issues

There are a number of outstanding issues related to attributes. Some values are strongly connected to a specific, existing component version, such as a static analysis or measurements based on an implementation, while others may refer to a not yet existing version of a component, such as estimates of the final system. Also, an estimated attribute value, or a value derived from a design model may be outdated since some requirements have been dropped since the attribute value was specified. In addition, there may exist two estimates which are equally “true”, such as estimates by two different persons, or by a person and by a tool analyzing incomplete design models or unfinished implementations, or when an analysis tool is used which is known to give safe over-pessimistic attribute values. And the possibility of composing values from values of different specification methods (such as measurements and static analysis in the example) hints at the need of some kind of inheritance model. The possibility for exploratory analysis addresses these problems, but only by postponing these decisions to specific organizations and projects. We are currently working on how to create an attribute model which is expressive enough, while not overloading developers and other project members with work [23].

In the present paper, the functionality of a component has been considered to be an attribute, but a component’s implementation has been modeled explicitly as a separate concept. It is yet not clear which aspects of a component should be included in the attribute concept. However, with appropriate languages, functional descriptions can indeed be composable so the general approach holds.

The relatively simple attribute of “memory consumption” has deliberately been used as the main example in this paper (although even this attribute is in reality not quite as simple as assumed in the examples). For other attributes the composition theories are not as straightforward, but are being researched. For example, response time can in principle be calculated by analyzing the structure and aggregate individual components’ response times or execution times, assuming there is also information available about how they are allocated to hardware, how they are scheduled, etc. [27], or by parameterizing the results [4][28]. Error propagation is another type of structural analysis [29]. For the whole community, the composition of each such attribute has to be solved to an industrially relevant level of confidence. From our point of view, it is enough to state that a verification-intensive process relying on the existence and accuracy of theories and tools is only as good as those theories and tools.

## 7 Summary and Conclusion

This paper outlines the *Progress Process Guidelines* (PPG), which augments the CMMI in order to provide support when creating organizational processes utilizing the component-based paradigm with a strong tool-set for analysis and compositional reasoning of various component attributes, such as the ProCom component model and related tools.

### 7.1 Future Work

There are several directions to cover in more detail while developing the final PPG. The conceptual product meta-model presented in this paper needs to be refined, regarding for example attributes, component versions, and more, as indicated throughout the paper. Also to be included is the allocation of components to hardware, which has not been considered at all in this paper. Such allocations must are also subject to verification, to ensure that e.g. the memory at different nodes of the system will be sufficient, and timing requirements such as response time will have to take into account task schedules on each node as well as network transportation delays. The responsibilities and skills needed in the various process areas need to be refined. Especially the TS roles needs to be better defined; possibly there needs to be a distinction between (at least) architect, designer, modeler, and programmer. In addition to the conceptual product meta-model, a large part of the PPG will be structured according to the CMMI, as outlined in this paper, and the guidelines formulated in CMMI terminology, i.e. as subpractices and/or amplifications [24].

In addition to the guidelines approach represented in this paper, some example processes will be modeled, to serve as pedagogic examples to which organizations can relate, as well as to be simulated. Process simulation is typically performed in two steps: an existing process is first modeled and simulated using real data, followed by some experimentation on the thus validated model. In our case, since a quite novel process is modeled, only very modest conclusions can be drawn. Simulations may nevertheless indicate interesting trends and correlations between modeled parameters, and bottlenecks in the process. Expected results include rules of thumb regarding the number of staff in each role is required, and exploration of relationships such as how the level of automation in the analysis affects the time required for exploratory analysis and verification, and this together with the expected number of non-conformances being introduced in the process will affect how often it should optimally be performed.

Modeling and simulation of concrete processes will later be followed by a pilot case in industry, once the tools has been implemented and reached a mature enough state. And finally, the usage of the analysis methods and tools developed at the Progress centre will be related to, and described in terms of, PPG.

## Acknowledgements

This work was partially supported by the Swedish Foundation for Strategic Research (SSF) via the strategic research centre PROGRESS. The authors would like to thank Jörgen Hansson at the SEI for his mentioning “Virtual Verification” in a talk, which brought some structure to our early ideas.

## References

1. Sentilles, S., Vulgarakis, A., Bureš, T., Carlson, J., Crnković, I.: A Component Model for Control-Intensive Distributed Embedded Systems. In: Chaudron, M.R.V., Szyperski, C., Reussner, R. (eds.) CBSE 2008. LNCS, vol. 5282, pp. 310–317. Springer, Heidelberg (2008)

2. Håkansson, J., Carlson, J., Monot, A., Pettersson, P.: Component-Based Design and Analysis of Embedded Systems with UPPAAL PORT. In: 6th International Symposium on Automated Technology for Verification and Analysis, Seoul, pp. 252–257 (2008)
3. Håkansson, J., Pettersson, P.: Partial Order Reduction for Verification of Real-Time Components. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) Proceedings of the 5th International Conference on Formal Modelling and Analysis of Timed Systems (2007)
4. Bygde, S., Lisper, B.: Towards an Automatic Parametric WCET Analysis. In: Worst-Case Execution Time Analysis Workshop, Prague (2008)
5. Shin, I., Behnam, M., Nolte, T., Nolin, M.: Synthesis of Optimal Interfaces for Hierarchical Scheduling with Resources. In: Proceedings of the 29th IEEE International Real-Time Systems Symposium (RTSS 2008), Barcelona (2008)
6. Behnam, M., Nolte, T., Shin, I., Åsberg, M., Bril, R.: Towards Hierarchical Scheduling on top of VxWorks. In: Proceedings of the Fourth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2008), Prague, pp. 63–72 (2008)
7. Crnković, I., Chaudron, M., Larsson, S.: Component-based Development Process and Component Lifecycle. In: International Conference on Software Engineering Advances (ICSEA 2006), Tahiti (2006)
8. Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Pearson Education, Boston (2003)
9. Krahn, H., Rumpe, B., Völkel, S.: Roles in Software Development using Domain Specific Modelling Languages. In: Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM 2006), Portland, Oregon (2006)
10. Aagedal, J., Solheim, I.: New Roles in Model-Driven Development. In: Proceedings of Second European Workshop on Model Driven Architecture (MDA), Canterbury, England (2004)
11. Kruchten, P.: The Rational Unified Process: An Introduction, 3rd edn. Addison-Wesley, Upper Saddle River (2004)
12. McConnell, S.: Rapid Development, Taming Wild Software Schedules. Microsoft Press (1996) ISBN 1-55615-900-5
13. Beck, K.: EXtreme Programming EXplained: Embrace Change. Addison-Wesley, Reading (1999)
14. Duvall, P., Matyas, S., Glover, A.: Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley Professional, Reading (2007)
15. As-2 Embedded Computing Systems Committee: Architecture Analysis & Design Language (AADL). Standard Document Number AS5506 (2009)
16. Defence Materiel Organisation, Australian Department of Defence: +SAFE, V1.2: A Safety Extension to CMMI-DEV, V1.2., Pittsburgh (2007)
17. Hänninen, K., Mäki-Turja, J., Sandberg, S., Lundbäck, J., Lindberg, M., Nolin, M., Lundbäck, K.-L.: Framework for Real-Time Analysis in Rubus-ICE. Hamburg (2008)
18. Larsson, M., Wall, A., Wallnau, K.: Predictable Assembly: The Crystal Ball to Software. ABB Review (2) (2005)
19. van Ommering, R., van der Linden, F., Kramer, J.: The Koala Component Model for Consumer Electronics Software. IEEE Computer 33(3), 78–85 (2000)
20. Crnković, I., Chaudron, M., Larsson, S.: Component-based Development Process and Component Lifecycle. Journal of Computing and Information Technology 13(4), 321–327 (2005)
21. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Reading (2001)

22. Bureš, T., Carlson, J., Crnković, I., Sentilles, S., Vulgarakis, A.: ProCom - the Progress Component Model Reference Manual, version 1.0., Västerås (2008)
23. Sentilles, S., Štěpán, P., Carlson, J., Crnković, I.: Integration of Extra-Functional Properties in Component Models. In: 12th International Symposium on Component Based Software Engineering (CBSE 2009). Springer, Heidelberg (2009)
24. Chrissis, M., Konrad, M., Shrum, S.: CMMI Second Edition: Guidelines for Process Integration and Product Improvement. Addison Wesley, Boston (2007)
25. Hissam, S., Moreno, G., Stafford, J., Wallnau, K.: Packaging Predictable Assembly with Prediction-Enabled Component Technology, Pittsburgh (2001)
26. Larsson, S.: Key Elements of Software Product Integration Processes, Västerås (2007)
27. Santos, M., Lisper, B.: Evaluation of an Additive WCET Model for Software Components. In: 10th Brazilian Workshop on Real-time and Embedded Systems, Rio de Janeiro (2008)
28. Fredriksson, J., Nolte, T., Nolin, M., Schmidt, H.: Contract-Based Reusable Worst-Case Execution Time Estimate. In: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), Daegu (2007)
29. Aysan, H., Punnekkat, S., Dobrin, R.: Error Modeling in Dependable Component-based Systems. In: IEEE International Workshop on Component-Based Design of Resource-Constrained Systems (CORCS 2008), Turku (2008)