Using Formal Specifications in the Implementation of CMMI

Bernd-Holger Schlingloff, Satish Mishra*
*Fraunhofer FIRST and Humboldt University, * Institut für Informatik, HU Berlin*
{holger.schlingloff,mishra}@first.fraunhofer.de

Abstract

There are two main aspects of quality assurance in computational systems development and maintenance projects: the process and product view. Several standard models have been proposed for a systematic process improvement, e.g., CMM/CMMI, Agile, SPICE, or the ISO 9000 family. However, even the best process can not guarantee that the resulting products are as expected. For a rigorous analysis of the products, formal specification based development methods have been proposed. Examples are VDM, Z, LOTOS, CSP and CASL.

In this paper we connect these two aspects by showing how to integrate formal specification based methods in process improvement models. In particular, we investigate the use of the specification language CSP-CASL for the implementation of CMMI within an organization. CMMI is based on the notion of process area, which is a cluster of best practices with particular goals in a certain area. For each of the relevant process areas, we show in detail how using formal specifications can help to achieve specific goals. This is a first step to systematically combine product based quality assurance methods with process improvement models.

1. Introduction

Currently, computational systems (hard- and software) are incorporating more and more functionality, which leads to an ever increasing complexity. This is the main reason for errors in the development of these systems. For the development of a complex computational system which nevertheless is reliable both product and process based quality assurance methods are necessary. CMMI (Capability Maturity Model® Integration) [6] [13] is a well-established process improvement model which has proved its benefits in hundreds of companies and thousands of projects. For product based software quality improvement, formal specification methods have been in existence for many years. Several of these methods have proven to be of advantage for the development of safety-critical systems, e.g. in aerospace or in the medical industry.

In this paper we describe the use of formal specifications for the implementation of CMMI in an organization. This way, the benefits of a broadly accepted, standardized process improvement model can be combined with the advantages of formal specification for product assurance. CMMI is the enhanced model of CMM, the Carnegie Mellon Software Engineering Institute's Capability Maturity Model, which was introduced in 1991. The latest version is "CMMI for Development 1.2", which is a process improvement maturity model for the development of products and services. It consists of best practices that address development and maintenance activities which cover the whole product lifecycle, from conception through delivery until maintenance [14] [7] [8] [9] [10].

A formal specification language consists of a well-defined syntax and a mathematical semantics. A formal method based on a specification language includes some transformation algorithm, which makes it appropriate for specifying, verifying and validating systems. Examples for formal methods which are being used in industry include VDM [17] and Z [12], Larch or, more recently, CASL [1]. These methods focus on the specification of structural properties, whereas CSP [2] [3], CCS [19], LOTOS, StateCharts, or Temporal Logic focus on behavioral properties. Many research papers have been published on the application of these or other formal languages in industrial practice [24][23][20].

In this paper we use CSP-CASL, which is a rather new algebraic / process algebraic specification language [22]. This choice is based on the fact that it includes means for both structural and behavioral properties. CSP (Communicating Sequential Process)

has been in existence since decades. It is a language well suited for the specification of reactive behavior. CASL (the Common Algebraic Specification Language) has been developed by CoFI, the common framework initiative, and can be used to specify structural properties of systems such as data types, functions and objects.

Using CMMI and CSP-CASL as an example, we analyze the integration of formal specification methods into process improvement models. We discuss how formal specification can contribute in the implementation of a process area, and, in particular, to achieve the specific goals associated to that process area.

Our paper is organized as follows. In the next section we will give a brief overview of CMMI, and in section three we will present the required details of CSP-CASL. In a subsection of this section we describe a case study which will be referred throughout the paper. Section four contains the main results about the potential contribution of a formal specification language in a CMMI implementation. In section five we give a conclusion and describe future work.

2. CMMI, Capability Maturity Model Integration

CMMI is a framework for assisting organizations to improve their development and maintenance processes for software product development. CMMI is based on the notion of *process area (PA)*. A process area is a cluster of related practices in an area. A typical process area is requirements management, which subsumes all practices necessary for dealing with demands the software has to fulfill. When a process area is implemented, it satisfies several goals which are considered important for making improvements in that area. CMMI offers two representations for its implementation, a *continuous* representation and a *staged* representation. The continuous representation offers more flexibility for process improvement. An organization can choose a focused process area, determine the dependent process areas, improve these at priority, and then concentrate on other process areas. In the staged representation, process areas are grouped together into capability maturity levels.

In this paper we are considering the staged representation. Since the notion of process area is independent from the representation, our results hold for the continuous representation as well. CMMI has 22 process areas which are considered important for the improvement of an organization. Generally, the domain of work which is performed within the organization can be divided into four groups: process management, project management, engineering and support. Each of these groups has a set of process areas for improving the capabilities of its processes. Associated with each process area is a set of goals which have to be satisfied as a measure for improvement in that process area.

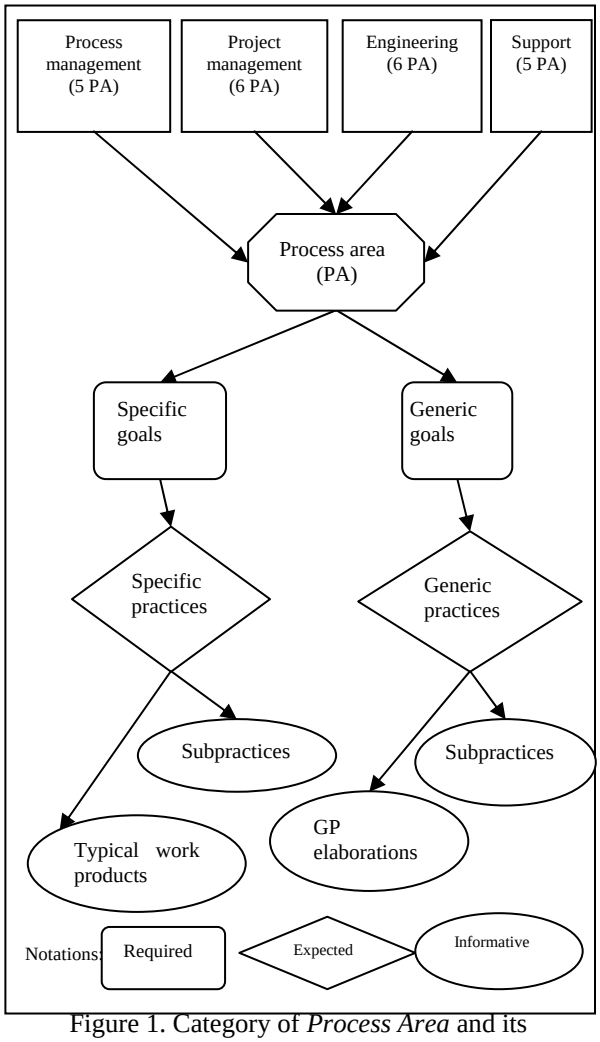

Figure 1. Category of *Process Area* and its components

Process areas are detailed by so-called *model components*. There are three types of model components. *Required* model components describe what an organization must achieve to satisfy a process area. *Expected* model components describe what it may implement to achieve the required components. *Informative* model components provide details which

help to initiate the approach the required and informative model component. Figure 1 shows the groups of process areas and its model components.

CMMI has a specific approach to describe the process areas. The description of a process area starts with introduction, purpose and relation with other process areas. These are informative components. Main characteristics of a process area are described by using *specific goals (SG)* and *generic goals (GG)*. The specific goals are unique characteristics that must be present to satisfy the associated process area. A *specific goal* is a required model component. A *specific practice (SP)* is the description of an activity that is considered important in achieving the associated specific goal. A specific practice is an expected model component.

The *generic goals* are the characteristics that must be present to institutionalize the processes which implement a process area. *Generic goals* are called "generic" because the same goal applies to multiple process areas. A generic practice is the description of an activity that is considered important in achieving the associated generic goal. Thus, a generic practice is an expected model component. For analyzing the integration of formal specification methods into process improvement models we only have to consider specific goals and its specific practices of the process areas. Generic goals are mainly for institutionalization of each process areas.

3. CSP-CASL, Formal Specification Language

CSP-CASL is a process algebraic specification language. For the specification of any system, processes of the system are specified by CSP and communications between these processes are the values of CASL data types. All the required features of CSP and CASL are supported by the combination [22]. The basic syntax of CSP-CASL is an integration of the CSP syntax and the CASL syntax. Syntactically a CSP-CASL specification *Sp* consists of a data part *D*, which is a CASL specification, an (optional) channel part *Ch* to declare channels, which are typed according to the data specification, and a process part P written in CSP, where CASL terms are used as communications. Thus, the generic format of a CSP-CASL specification is

(ccspecification Sp = data D channel Ch process P).

The semantics of CSP-CASL is defined in three steps. In the first step each channel is encoded in CASL. In the second step CASL data types are evaluated, where families of processes are generated according to the data model. In the last step the evaluation according to CSP is taking place.

3.1. CSP, Communicating Sequential Processes

CSP is a formal language for describing reactive systems. Such systems often have complex interactions among their processes which make them difficult to comprehend. CSP provides a set of mathematical symbols to analyze such complex systems with clarity and preciseness.

In CSP, an object ($e \rightarrow P$) is described with an event *e* and process *P*, which states that the object first engages in an event *e* and then behaves like process *P*. The meaning of a CSP process *P* is given by its traces *tr(P)*, which is the set of sequences of events that a process may possibly perform. Two primitive processes of CSP are STOP and SKIP, where STOP is the process which never engages in any event and SKIP is the process which does nothing but terminate successfully. CSP has a rich syntax for describing processes; some of the syntactic features are described subsequently.

An event can be combined with a process using the prefix operator ($\rightarrow$). A process may be defined in terms of itself using recursion [3]. There are CSP operations which combine processes in a sequential or parallel manner. CSP supports two types of choice operators, external and internal choice. The interleaving form of parallel combination is supported by CSP where processes don't communicate with each other. One of the important features for abstract specification of systems in CSP is nondeterminism, where a process can choose amongst several alternatives. Hiding of events is supported for abstraction; this may also lead to nondeterminism.

For CSP, several semantics have been defined: operational, denotational, and algebraic semantics. The semantics can be based on a trace model, where a trace is sequence of communications between the environment and the system's processes.

3.2. CASL, Common Algebraic Specification Language

CASL has been developed by the international CoFI (Common Framework Initiative). It subsumes most of the previous approaches in algebraic specification [1][22]. The CASL can be separated into four parts,

which facilitates the description of systems with an appropriate detail. These parts are basic specifications, structured specifications, architectural specification and library based specification. A basic specification includes first-order total functions, partial functions and predicates. To describe a system completely, subsorts, axioms and datatype declarations are integrated into the basic specification. Keywords of basic specification are *sorts*, *ops*, *preds*, *vars* etc. A structured specification is a combination of such basic specifications into a larger specification in a hierarchical and modular fashion. It mainly allows translation (keyword *with*), reduction (keyword *hide*), union (keyword *and*) and extension (keyword *then*) of specifications. An architectural specification allows systems to be developed with reusable components. A library based specification allows easy distribution and reusability in a user friendly way.

A small example of a CSP-CASL specification is the case study given below. The keywords in this example are interpreted as follows. Keyword *op* and *ops* are used to declare the operations, keyword *pred* and *preds* are used to declare the predicates. The keyword *sort* is interpreted as data type, similar to data types in any programming language. The keyword *then* facilitates the specification extension with more features. Further sections will elaborate more about the case study and its applicability in our approach.

3.3. Case study

In this subsection we describe a small case study of a basic remote control for electronic devices such as TV, VCR etc. The example is used to demonstrate the approach of specification based development in a CMMI environment, in particular the relationship among requirements, design documents, test cases and implementation. This case study is too small to demonstrate all necessary details for the compliance of CMMI with a formal specification based development, but it gives an intuitive idea of the ideas. To assess in which way formal specifications can contribute to a CMMI implementation, we have applied this approach in an industrial case study of a medical embedded system [20]. For sake of simplicity, in this paper we restrain ourselves to the basic remote control case study. A more elaborate version of this example can be found in [27]

3.3.1. Informal description. The basic remote control unit (RCU) can be described as follows. It has 12 buttons which are button_0, …, button_9, button_on and button_off. Whenever any button is pressed, the corresponding sixteen bits stream of signal, in specific pattern is send via an LED (Light Emitting Diode). The remote control has an internal table for determining the key code of the respective buttons.

3.3.2. CSP-CASL specification (Requirement Analysis)

```
library Basic/Numbers get Nat, Int
Ccspec BasicRCU
  data
        sort Button
        sort Signal = {s:Signal • #s = 16}
        op button_0, button_1, …, button_9: Button
        op button_on, button_off : Button
        op responseSignal : Button → Signal

  channel
        Keypad: Button
        Infrared: Signal
  process
     Control =  Keypad ? x:Button →  Infrared !
                responseSignal (x) →  Control
end
```

The above description is a formal specification of the RCU which is informally defined above. The complete specification is divided into three parts, namely data, channel and process part. In the data part sorts, operations and predicates are declared. The first line of specification includes the basic library of data types. The specification name is given in the next line. After that, *Button* and *Signal* are declared as data type of *sort* which is further used to declare all the buttons of RCU. In the last lines of the data part the operation *responseSignal*, is declared which takes *Button* as input and returns *Signal* as output.

The next part of the specification declares the channels which are used by the CSP processes for communication with the environment. Here *Keypad* and *Infrared* are declared as channels of type *Button* and *Signal*, respectively. The last part of the specification declares the CSP processes which define the reactive behavior of the RCU. Whenever a remote control button is pressed, the respective signal is passed on the channel. This is described by the recursive process *Control*. This process waits for an event (of type *Button*) to occur. After execution of this event it performs the operation *responseSignal*, which accepts a pressed button as input, outputs the respective signal on the channel *Infrared* and then continues recursively.

3.3.3 CSP-CASL specification (Design Document)

A software development process can be understood as the construction of a sequence of increasingly more detailed descriptions from an initial requirements document, until an executable program is reached. At each step design decisions are fixed coherent with the requirements. The process of fixing design decision is referred as *refinement*. In our case study, the pattern of 16 bits signal is fixed according to the standard guideline: The initial four bits are a company ID, the next four bits are a device ID and the other remaining bits represent the control signal for the functionality of a particular button. Subsequently we give the refinement of our earlier specification, according to this pattern for a specific company.

```
library Basic/Numbers get Nat, Int
Ccspec BasicRCU
  data
        sort Button
        op button_0, button_1, …, button_9: Button
        op button_on, button_off : Button
        free type Bit ::= 0 | 1
        then List[Bit]
        then
        sort Signal = {l:List[Bit] • #l = 16}
        DeviceId :  List[Bit] = [0010]
        CompanyId : List[Bit] = [1000]
        op responseSignal : Button → Signal
        axioms
        responseSignal(button_0) = DeviceId ++
        CompanyId ++ [00000000]
        responseSignal(button_1) = DeviceId ++
        CompanyId ++ [00000001
        …
        responseSignal(button_On) = DeviceId ++
        CompanyId ++ [00000100]
        ∀ b : Button • ∃ L : List[Bit] •
        responseSignal (b)= DeviceId ++ CompanyId ++ L
        • #L=8
  channel
        Keypad: Button
        Infrared: Signal
  process
      Control =  Keypad ? x:Button →  Infrared !
                 responseSignal (x) →  Control
end
```

In the design document, data types are refined, and additional constraints are added to satisfy certain product properties. The operation *responseSignal* is extended with details of the button and its corresponding signal is defined by axioms on the operation level. An initial group of axioms fixes the *responseSignal* for a set of buttons. The last axiom in this specification confirms that all the buttons of the RCU follow the pattern of signal which is decided by the company. We will refer this specification to demonstrate which features are well suited for CMMI compliance.

4. Formal specification and CMMI

To analyze the contribution of formal specification for the implementation of CMMI in an organization we developed the following the grading scale.

1) **Fully Contributed (FC):** A process area is satisfied as FC if 90-100% of its specific goals are achieved using formal specification. A specific goal is achieved as FC when 90-100% of its specific practices can be performed by formal specification.

2) **Largely Contributed (LC):** A process area is satisfied as LC if 60-89% of its specific goals are achieved using formal specification. A specific goal is achieved as LC when 60-89% of its specific practices can be performed by formal specification.

3) **Partially Contributed (PC):** A process area is satisfied as PC if 30-59% of its specific goals are achieved using formal specification. A specific goal is achieved as PC when 30-59% of its specific practices can be performed by formal specification.

4) **Not Contributed (NC):** A process area is NC if 0-29 % of its specific goals is achieved using formal specification. A specific goal is NC when 0-29% of its specific practices can be performed by formal specification.

Table 1, 2 and 3 show an overview of our categorization on a process area which demonstrates the contribution of formal specification to it.

Table 1. Formal specification contribution categorization on *CMMI*

Process Area (PA)	**Specific Goals of a PA (% of total)**	**Specific Practices of SG (% of total)**
Fully Contributed (FC)	90 to 100	90 to 100
Largely Contributed (LC)	60 to 89	60 to 89
Partially Contributed (PC)	30 to 59	30 to 59
Not Contributed (NC)	0 to 29	0 to 29

Table 2. Categorization overview on *Specific Goal* and *Specific Practice*

Specific Goal	**Specific Practices (% of total)**
Fully Contributed (FC)	90 to 100
Largely Contributed (LC)	60 to 89
Partially Contributed (PC)	30 to 59
Not Contributed (NC)	0 to 29

Table 3. Categorization overview on *Specific Practice* and its *Activity*

Specific Practice	**% of Activity**
Fully Contributed (FC)	90 to 100
Largely Contributed (LC)	60 to 89
Partially Contributed (PC)	30 to 59
Not Contributed (NC)	0 to 29

Formal specification has many features which can contribute for the compliance of CMMI in an organization. Model development, model complexity analysis, refinement of model, structured development, verification and validation are the main features. In the subsequent parts of this paper we illustrate the integration of these features for the compliance of CMMI practices with the help of our example.

4.1 Contribution of formal specification at CMMI maturity level 2

At maturity level 2 of CMMI, it is required that there is a written policy such that all the project related processes are planned and used. All the projects of an organization derive the processes according to a defined policy for the organization. Project employees must have proper understanding of the processes and their controlling behavior. Processes must be reviewed, controlled and monitored at specific milestones. Once planned processes are in place, projects are managed according to these plans. This maturity level is referred as the *managed* level. At this level of maturity there are seven processes areas. In this paper we just show results for a single process area, namely "Requirements Management". There are similarly many contributions of formal specification to the compliance of other process areas such as "Process and Product Quality Assurance", "Project Monitoring and Control", and "Measurement and Analysis".

4.1.1 Requirement Management

The process area "Requirements Management" provides guidelines for dealing with demands for product features and product component features. In addition to this, it also provides guidelines for removing inconsistencies between requirements and other work products. Formal specification methods can contribute to a *specific goal* and a *specific practice* of this process area as shown in Table 4.

Table 4. Requirement management process area

Specific goals and specific practices	**FSC**
SG 1 Manage Requirements	LC
SP 1.1 Obtain an Understanding of Requirements	FC
SP 1.2 Obtain Commitment to Requirements	LC
SP 1.3 Manage Requirements Changes	LC
SP 1.4 Maintain Bidirectional Traceability of Requirements	LC
SP 1.5 Identify Inconsistencies Between Project Work and Requirements	LC

In this table, we present the contribution of formal specification for each Specific Practice (SP). Each of these estimations is based on our experiences with CSP-CASL; for space reasons, in this paper we give only some example justifications.

Since the formulation of requirements in a formal specification has a precise and unambiguous semantics, it provides a way to ensure specific practices of this specific goal. The refinement relation among requirement, detailed design and implementation provides an approach for maintaining consistency and traceability of the requirement throughout the software development lifecycle. Once a requirement is formally specified, many aspects of the development life cycle and work products can be automated.

Table 5. Refinement relation

In requirement
responseSignal: Button → Signal
In design document
responseSignal: Button → Signal axioms responseSignal(button_0) = DeviceId + + CompanyId ++ [00000000] responseSignal(button_1) = DeviceId + + CompanyId ++ [00000001] …….
In test case
Inp: button_0 out: DeviceId ++ CompanyId ++ 00000000 Inp: button_1 out: DeviceId ++ CompanyId ++ 00000001 Etc

Table 5 shows selected part of the case study. This formal specification demonstrates the refinement relation between requirement and design document. The operation *responseSignal* is refined into detailed design by restricting its models with the help of axioms. Axioms are added in detail design to ensure that the output of a pressed button produces a specific value of the signal. There have been various approaches for the derivation of test case from formal specifications [24][6][24]. Here, we consider the case that each axiom leads to a test case, and then we deduce some test cases which are shown in the Table 5. Each axiom is a template for test cases; we can derive test cases from the design document or the requirements. In addition, a formal specification also provides an approach for establishing a relation among requirements, design and test cases which gives the possibility for bidirectional traceability among the work products.

Verification and validation activities can also be automated by using formal specifications. This allows maintaining consistency between requirements and verification / validation related work products.

4.2 Contribution of formal specification at CMMI maturity level 3

At maturity level 3, processes are well understood within the organization. At this level, an organization must possess and maintain a set of standard processes which is used by all projects. Tailoring of processes is permitted, to tackle the specific requirement of a specific project, but it has to follow certain fixed tailoring guidelines. In this way the organization is able to preserve consistency amongst the processes used in the projects. Maturity level 3 is the *defined* level and has eleven process areas. In this paper we consider five engineering related process areas for our study.

4.2.1 Product Integration

The process area "Product Integration" guides the integration of component's functions according to the requirements, and the integration of components into a complete product. Formal specification can contribute to the specific goals and the specific practices of this process area as shown in the table below.

Table 6. Product integration process area detail

Specific goals and specific practices	**FSC**
SG 1 Prepare for Product Integration	LC
SP 1.1 Determine Integration Sequence	LC
SP 1.2 Establish the Product Integration Environment	LC
SP 1.3 Establish Product Integration Procedures and Criteria	LC
SG 2 Ensure Interface Compatibility	PC
SP 2.1 Review Interface Descriptions for Completeness	LC
SP 2.2 Manage Interfaces	LC
SG 3 Assemble Product Components and Deliver the Product	PC
SP 3.1 Confirm Readiness of Product Components for Integration	PC
SP 3.2 Assemble Product Components	PC
SP 3.3 Evaluate Assembled Product Components	PC
SP 3.4 Package and Deliver the Product or Product Component	PC

Formal specification has been proposed for component based development, e.g., in [26]. Specially, CSP-CASL provides significant features for component based development, such as giving a structural and architectural approach to requirements engineering [1]. In addition, the advantage of CSP-CASL for product line based development has been studied in [22]. Process algebra has very powerful features for mastering the complexity of processes via parallel and sequential composition. An unambiguous definition of interfaces and its behaviors reduces the complexity of implementing the whole system. Formal specification plays a significant role for the verification and validation of safety-critical software systems. Test

suite reusability has been studied for component based development in [24]. Formal specification based development guarantees various quality aspects for a product which is composed of many components.

4.2.2 Requirement Development

Customer, product and product component requirements are analyzed and produced using the "Requirement Development" process area guidelines. This process area describes customer requirements, product requirements and product component requirements. The contribution of formal specifications to this process area is shown in the Table 7.

Table 7. Requirement development process area detail

Specific goals and specific practices	FSC
SG 1 Develop Customer Requirements	FC
SP 1.1 Elicit Needs	LC
SP 1.2 Develop the Customer Requirements	FC
SG 2 Develop Product Requirements	FC
SP 2.1 Establish Product and Product Component Requirements	FC
SP 2.2 Allocate Product Component Requirements	FC
SP 2.3 Identify Interface Requirements	LC
SG 3 Analyze and Validate Requirements	LC
SP 3.1 Establish Operational Concepts and Scenarios	LC
SP 3.2 Establish a Definition of Required Functionality	LC
SP 3.3 Analyze Requirements	LC
SP 3.4 Analyze Requirements to Achieve Balance	PC
SP 3.5 Validate Requirements	FC

The main advantage of a formal specification is its unambiguous and precise description of the system. Formal specification of a requirement starts by specifying it abstractly. Then the abstract specification is augmented with more details to fix the system properties. This process of stepwise refinement is repeated until all design decisions are fixed. Formal specification also uses refinement to establish the consistency between requirement, design and test cases. Formal specification has shown significant benefits when generating frameworks for the validation and verification of products and product components. In the case of the "Requirements Development" process area, the specification language can make major contributions by ensuring various specific practices of SG 1, SG 2 and SG 3. It can support various approaches for modeling and analyzing the requirements. This is essential for prototyping a product before proceeding to product development. In safety-critical systems development [16], the use of formal specification for achieving a better product quality has been extensively studied.

4.2.3 Technical Solutions

The "Technical Solution" process area provides guidance for design, development and implementation of solutions of the given requirements. The technical solution process area must be applied at any level of the product architecture, to every product and product component. The main focus of this process area is to evaluate and select a solution, to develop a detailed design of the selected solution and to implement the design as a product or product component. The contribution of formal specification methods is presented in Table 8.

Table 8. Technical solutions process area detail

Specific goals and specific practices	FSC
SG 1 Select Product Component Solutions	LC
SP 1.1 Develop Alternative Solutions and Selection Criteria	LC
SP 1.2 Select Product Component Solutions	LC
SG 2 Develop the Design	PC
SP 2.1 Design the Product or Product Component	LC
SP 2.2 Establish a Technical Data Package	PC
SP 2.3 Design Interfaces Using Criteria	PC
SP 2.4 Perform Make, Buy, or Reuse Analyses	PC
SG 3 Implement the Product Design	PC
SP 3.1 Implement the Design	PC
SP 3.2 Develop Product Support Documentation	PC

To implement a product requirement in the best possible way, ideally the system should be designed and analyzed with several different approaches. After proper analysis, the best approach should be selected for further implementation. Formal specification can serve as one way for prototyping the model before its implementation. Once a system is specified formally, the formal specification can be refined into a product design specification, such that the consistency is

automatically maintained.

However, since this ideal is seldom followed in practice. In practice software development is constrained with time and cost which generally restricts our ideal approach until it is not attached to the safety critical aspects.

The specification language CSP-CASL is well suited for specifying industrial applications [23]. CSP-CASL features are supported by step wise refinement. This is appropriate for specifying requirements, starting with a loose specification and later fixing the design properties to reach a deterministic specification. Refinement for parts of our case study specification is shown in Table 9. In this table, it is shown how a loosely defined requirement is refined into a design specification. Here, Signal of type sort is refined such that its value will be the concatenation of 0 and 1. The transformation of a design specification into an implementation can also be regarded as a (provable) refinement step. However, this refinement is rather specific to the particular implementation language. Here conformance testing plays a significant role to guarantee that the implementation conforms to the specification. Conformance testing for CSP-CASL has been studied in [27].

Table 9. Refinement in development lifecycle

Requirement	**Formal Design**	**Implementation**
Sort Signal = {s:Signal • #s = 16}	free type Bit ::= 0 | 1 then List[Bit] Sort Signal = {l:List[Bit] • #l = 16}	In implementation language according to their syntax

4.2.4 Validation

The purpose of the activities in the "Validation" process area is to demonstrate that a product or product component fulfills its intended use when placed in its intended environment. The contribution of formal specifications for the validation process area is as follows.

Table 10. Validation process area detail

Specific goals and specific practices	**FSC**
SG 1 Prepare for Validation	FC
SP 1.1 Select Products for Validation	LC
SP 1.2 Establish the Validation Environment	FC
SP 1.3 Establish Validation Procedures and Criteria	FC
SG 2 Validate Product or Product Components	FC
SP 2.1 Perform Validation	FC
SP 2.2 Analyze Validation Results	LC

The formal specification based system development approach can make major contributions to this process area. It can help to achieve all specific goals by satisfying the specific practices. Each SG1 of process is implementation requires that an environment is set up for the validation of the product. To validate a product in an early stage of the project life cycle, specification based model development can contribute as a prototype for the complete product.

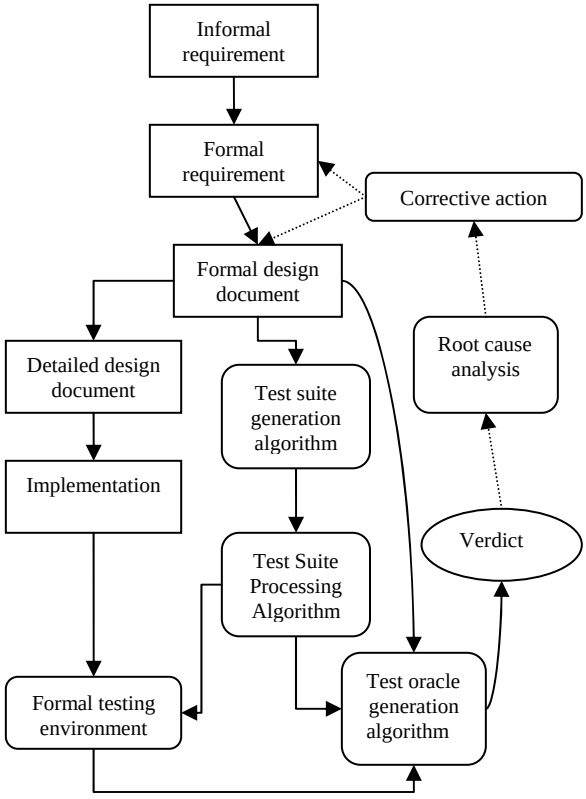

Figure 2. Testing architecture

In the complete life cycle of a product, the process area demands that a validation framework is set up. Given a formal specification, in such a validation framework certain aspects of testing such as test generation, test execution and test evaluation can be automated. Therefore, the formal specification approach can help to achieve the goals of this process

area. Specification based test case generation has been studied in many research papers, and has been found one of the most important advantages of specification based development[21][24]. Test generation and reusability of test suites and various testing environments are being investigated for CSP-CASL in [20][4]. The specification language also provides ways to analyze the results from the test execution for specific test cases. The automatic generation of test oracles has been found useful in recent industrial case studies [4]. Figure 2 shows a typical arrangement for test automation. This ongoing research will enhance the applicability of CSP-CASL in the testing domain.

4.2.5 Verification

The "Verification" process area ensures that the products which are the result of the processes under improvement meet their specified requirements. Formal specification methods contribute to this process area as follows in table 11.

Table 11. Verification process area detail

Specific goals and specific practices	FSC
SG 1 Prepare for Verification	LC
SP 1.1 Select Work Products for Verification	LC
SP 1.2 Establish the Verification Environment	LC
SP 1.3 Establish Verification Procedures and Criteria	LC
SG 2 Perform Peer Reviews	NC
SP 2.1 Prepare for Peer Reviews	NC
SP 2.2 Conduct Peer Reviews	NC
SP 2.3 Analyze Peer Review Data	NC
SG 3 Verify Selected Work Products	PC
SP 3.1 Perform Verification	LC
SP 3.2 Analyze Verification Results	PC

Formal specification methods have various ways to contribute to the verification process area, the main two being model checking and theorem proving. Model checking is the process of building a model of a system and checking whether desired properties hold in this model [18] [15]. Theorem proving is the process of finding a proof of a property from the axioms of a system, where the property and the system are expressed in the formal specification language [11]. These two properties are well suited for SG1 and SG3 and its specific practices. Presently, the application area and size of products which can be verified by model checking and theorem proving is rather limited. However, advancements in technologies will facilitate the development of tools which will be able to verify many system properties with reasonable time, resource and effort. For CSP-CASL, model checking and theorem proving has been investigated in detail [28]. Therefore, this specification language is well-suited for the achievement of specific goals in this process area.

5. Conclusion

In this paper, we have studied the contribution of formal specifications for the implementation of the CMMI process improvement model. Formal specification based process area satisfaction is investigated via the process algebraic specification language CSP-CASL. A small case study is presented to illustrate the applicability of a specification language for achieving goals of the process areas. Out of 22 process areas from CMMI, six process areas can be significantly improved with the help of formal specification methods. These process areas and respective contribution from formal specification are summarized in Table 12.

Table 12: CMMI *Process Area* and formal specification contribution

Process area	FSC
Requirement Management	LC
Product Integration	LC
Requirement Development	LC
Technical Solutions	LC
Verification	LC
Validation	LC

In our research, we concentrated on the contribution of CSP_CASL as a formal specification language. Although we believe this language to be particularly well-suited, most of our results hold for other specification formalisms as well..

What is left open in this paper is the quantitative analysis of cost and benefits in practical examples. In particular, for CMMI level 4 (Quantitatively Managed) a quantitative measurement of the process area implementation level is required. In CMMI level 5 (optimizing) a continuous improvement is targeted. How to fit the present approach with these requirements needs more research.

6. References.

[1] Michel Bidoit, P. D. Mosses The Common Algebraic Specification Language Users/Reference Manual LNCS 2900/2960.

[2] C. A. R. Hoare: Communicating Sequential Processes. Commun. ACM 21(8): 666-677 (1978).

[3] A. W. Roscoe, The Theory and Practices of Concurrency. Prentice Hall, 1998.

[4] Patricia D. L. Machado Springer, Testing from Structured Algebraic Specifications, LNCS 1816/2000.

[5] J. A. Bergstra, A. Ponse, and S.A. Smolka, Handbook of process algebra, Elsevier, 2001.

[6] http://www.sei.cmu.edu/CMMI/ Software Engineering Institute, Carnegie Mellon , Pittusburgh, PA.

[7] Ahern, Dennis M. : CMMI distilled : a practical introduction to integrated process improvement / Dennis M. Ahern; Aaron Clouse; Richard Turner. - 2. ed. . - Boston [u.a.] : Addison-Wesley.

[8] Walker Royce, Software Project Management A unified framework, Addison Wesley.

[9] Pressman, Roger, Software engineering : a practioner's approach, , McGraw-Hill, 2000.

[10] David J. Anderson, Stretching Agile to fit CMMI Level 3 - the story of creating MSF for CMMI® Process Improvement at Microsoft Corporation, IEEE, ADC 05.

[11] Nagoya F, Shaoying Liu, Yuting Chen, A tool and case study for specification-based program review, COMPSAC 2005. 29th Annual International Volume 1, 26-28 July 2005 Page(s):375 - 380 Vol. 2.

[12] Kreuz, D, Formal specification of CORBA services using Object-Z Formal Engineering Methods, 1998. Proceedings, Second International Conference on 9-11 Dec. 1998 Page(s) :180 – 189.

[13] David J. Anderson, Stretching Agile to fit CMMI Level 3 - the story of creating MSF for CMMI® Process Improvement at Microsoft Corporation, ADC05, 193-201.

[14] Yoo, Junho Yoon, Byungjeong Lee, Chongwon Lee, An Integrated Model of ISO 9001:2000 and CMMI for ISO Registered Organizations, APSEC 04, 150-157.

[15] Alur R Henzinger, Automatic symbolic verification of embedded systems IEEE Transactions on Software Engineering 22,3, 181-201.

[16] Craigen D, Gerhart, Formal methods in critical systems IEEE Software 11,1 Jan.

[17] Jones C B, Systematic Software Development Using VDM Prentice Hall , International New York ,1986

[18] Roscoe A, Model checking CSP In A Roscoe Ed A Classical Mind Essays in Honor of C A R Hoare Prentice Hall 1994.

[19] Milner A, A Calculus of Communicating Systems Volume 92 of Lecture Notes in Computer Science_ Springer, 1980.

[20] Satish Mishra, Specification Based Software Product Line Testing: "Concurrency, Specification and Programming" CSP2006.

[21] M. C. Gaudel, Perry R James, Testing Algebraic Data Types and Processes: A Unifying Theory, Formal Aspect of Computing 10(5-6): 436-451 (1998).

[22] M. Roggenbach, CSP-CASL, A New Integration of Process Algebra and Algebraic Specification, Theoretical Computer Science 354 (2006), 42-71.

[23] A. Gimblett, M. Roggenbach, and H. Schlingloff, Towards a formal specification of electronic payment systems in CSP-CASL, WADT 2004.

[24] Jan Tretmans, Axel Belinfante, Automatic Testing with Formal Methods, In Proceedings of the 7th European International Conference on Software Testing.

[25] M.-C. Gaudel, Testing can be formal, too, LNCS 915, pages 82–96. Springer, 1995. .

[26] Elsa Estevez, Pablo Fillottrani, Algebraic Specifications and Refinement for Component-Based Development using RAISE, JCTS 2000

[27] Temesghen Kahsai, Markus Roggenbach, and Bernd-Holger Schlingloff, Specification-Based Testing for Refinement, SEFM 2007

[28] Y.Isobe, M.Roggenbach: A generic theorem prover of CSPrefinement, Proceedings of TACAS 2005