ABSTRACT
The CMMI defines two process areas associated with requirements elicitation: Requirements Development (RD) and Requirements Management (REQM). The Measurements and Analysis process area (MA) requires measurements and quantitative objectives for RD and REQM, but nowhere does it state what those measurements are. Furthermore, in order to extract measurements and evaluate them, a process must enable or otherwise support the taking of measurements. It is especially difficult to do this during requirements development, as it is generally viewed as a writing activity that does not lend itself to quantitative measurements. This paper describes a CMMI compliant formal approach to measurement and analysis during a model-driven requirements development process. It presents a set of metrics that were used successfully on several Siemens projects, describing team dynamics, project size and staffing, how the metrics were captured and used, and lessons learned.

Categories and Subject Descriptors
D.2.1 [Software Engineering]:Requirements/Specifications – elicitation methods, methodology, tools.

General Terms
Management, Measurement, Documentation, Verification.

Keywords
Requirements Engineering, metrics, analysis, UML, modeling.

1. INTRODUCTION
We have previously defined quality and productivity heuristics for UML analysis models [1]. Because of the difficulty of extracting meaningful metrics from collections of text documents, the requirements engineering program at Siemens Corporate Research (SCR) has focused on “model-driven” requirements processes for elicitation, organization and documentation of requirements. Starting with a set of features, usually originating in a sales or marketing group, detailed, verifiable requirements are discovered by creating a use case model of the processes associated with the product being developed. It is then possible to extract requirements and test cases programatically from the model [2][3]. Other researchers have focused on the need for formality in the modeling process, but, to date, because of the difficulty of working with natural language text, little work has been done in the area of requirements verification and validation [4].

In this paper, we discuss our experiences with the instrumentation of a model-driven requirements process, such that measurements can be taken and analyzed, describing a viable set of metrics that have successfully been used on Siemens projects. These measurements conform to the spirit of the CMMI, and provided feedback in the two areas of project progress and work product quality.

2. CMMI REQUIREMENTS RELATED PROCESS AREAS
CMMI contains a wealth of information about the “upfront” gathering of requirements for a new development effort [5]. The process areas most focused on for this paper are requirements management, requirements development, measurement and analysis, process and product quality assurance, and project monitoring and control.

Requirements Management is the management of all requirements received by or generated by the project, including both technical and non-technical as well as those requirements levied on the project by the organization.

Requirements Development is the production and analysis of customer, product and product-component requirements to represent the needs of relevant stakeholders.

Measurement and Analysis is the development and monitoring of a measurement capability that supports management information needs.

Process and Product Quality Assurance provides the objective insight into processes and associated work products by objectively evaluating process performance and the quality of work products produced.

Project Monitoring and Control measures the project’s progress compared to the planned activities, taking corrective action if the project deviates from the projects expected performance.

3. THE REQUIREMENTS PROCESS
It is easier to build measurements into a process than to try to add them later. That having been said, we can look at the CMMI for help with what is needed [5].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE’06, May 20–28, 2006, Shanghai, China.
Copyright 2006 ACM 1-59593-085-X/06/0005...$5.00.
3.1. Align Process With Needs

Different projects have different needs. In order to understand what measurements are necessary for a requirements elicitation and management effort, we can look at the roles, and then see what each role would require. There would typically be project management, subject matter experts, requirements analysts, quality assurance personnel, designers and testers associated with the requirements development phase of a project. At a high level we can say that:

• Subject matter experts would like to know which areas need further elucidation, and whether they have previously reviewed the material.
• Requirements analysts needs would be the same as subject matter experts, but they would also be interested in the quality of the UML model (keeping in mind that this is a “model driven” effort) and the work products and requirements to be derived from the model. They would also want to know what areas needed further investigation.
• Quality assurance personnel want to be sure that in-place processes are being followed, and that the quality of the work products can in some tangible way be measured.
• Designers are interested in coverage and clarity.
• Testers are primarily concerned with the generation of a test plan and test cases, e.g. how difficult it would be to generate a viable set of test cases from the completed model and associated documentation [3].
• Project managers are concerned with resource utilization, skill level of the project team, and progress (e.g. percent completeness). Of course they are also interested in the quality of the work products.

3.2. A Model-Driven Requirements Process

Defining measurements for a process is iterative. That is, we define a process, and then see if it meets our measurement objectives (what we are trying to measure). Of course, once the process is defined, new measurements may be discovered that lead to rethinking the process, resulting in new and/or different measurements.

For the projects described in this paper, objectives of the requirements development process included:

- Sufficiently formal that completeness could be measured
- Straightforward, with a well defined set of activities and guidelines for creating work products
- Programmatic extraction of requirements from the UML Model
- Programmatic extraction of test cases from the model
- A well defined set of process review criteria (not to be confused with domain criteria)
- Complete backwards-forwards traceability and
- Suitable for a distributed effort (i.e. the participants were in different locations).

3.2.1. Formal Requirements Development Process

A formal process [8] was specified that incorporated both new and previously defined heuristics [1], tailored for the tools that were used.

- A single Use Case context diagram was the starting point for navigation. This diagram showed all the actors that would be using the product under construction.
- Product Features were defined using abstract use cases.
- Use Cases were decomposed hierarchically.
- Each non-leaf use case was shown with its included, extending and inheriting use cases on a separate diagram(s).
- Each use case diagram was linked to the parent use case symbol on a higher level diagram.
- Abstract and concrete use cases were clearly differentiated. Abstract use cases were treated as placeholders, and as such, did not need exposition in the form of sequence, collaboration, state or activity diagrams and would not require test cases.
- Every actor shown on the context diagram had to communicate with a concrete use case through a boundary. This forced a full inventory of all the interfaces that the product would need.
- Every concrete use case, representing a process or activity, had to be explained on a sequence, activity or state diagram:
  1. On a sequence diagram the use case was described as a message (class method), or
  2. On an activity diagram, the use case was shown as an activity or
  3. On a state diagram, the use case was shown as a state transition

Process discipline needed to be imposed such that descriptions were filled in, artifacts (diagrams, use cases, etc.) had a visible status, and the model was kept clear, straightforward, and, above all, programmatically navigable.

3.2.2. Definition of Model Completeness

In order to determine progress it was necessary to define model “completeness”, that is, the point at which modeling could stop and the requirements and test cases could be extracted. We wanted a definition that could be programmatically verified. A model was declared complete when:

- Every leaf use case (a use case with no included, extending or inheriting use cases) was concrete, with a special stereotype. The stereotypes of functional and non-functional requirements were used, and the stereotypes were given special icons for ease of visual inspection.
- The leaf use cases became the high level requirements that were subject to a formal review. The model was also subject to a review, primarily to check for completeness and defects.

After having applied the approach described above, we discovered additional completion rules and metrics that were very useful when conducting model reviews (see section 11 below).

3.3. Requirements Generation and Management

Upon completion of the modeling effort, the leaf use cases as described in 3.2.2 were transcribed from the model to the requirements repository. Since different requirements tools were
used on each project, the transfer mechanism was programmatically different, but the algorithms were the same.

3.4. Formal Model Reviews
Both informal and formal model reviews were conducted. During the course of modeling, the DesignAdvisor\textsuperscript{TM}\cite{6} tool was used to add information to each diagram, and to each major symbol on each diagram as to status (e.g. use cases but not their relationships). This information was then used to determine whether various parts of the model were ready for an informal review. Once all the various model artifacts (diagrams, use cases,) had undergone informal reviews, formal reviews of the model with stakeholders were held. Note however, that these reviews, although important, did not carry the weight of a requirements review and were primarily informational in nature.

3.5. Formal Requirement Reviews
It was found empirically that only 4-10 requirements per hour could be processed during a formal, sit-down review (with stakeholders present). As the projects typically contained several thousand requirements in the database, the leaf use cases (stereotype of functional or non-functional requirements) turned out to be the appropriate level for a formal review.

4. MEASUREMENT OBJECTIVES
The measurement objectives were twofold a) measure project progress and b) measure product quality. We had to be able to:

- Specify Measurement and Analysis Procedures
- Specify Results
- Collect, Analyze and Interpret the Measurements
- Store Data And Results
- Communicate Results

In addition, the measurements were used to perform analyses that were compared to benchmark values. The metrics were used to determine:

- Was the staff was adequately trained?
- Was the team paying enough attention to the process (e.g. was the defect rate acceptable)?
- Was the overall work product quality adequate?
- Were the requirements traceable?
- Were the work products suitable for a) in-house manufacturing/development or b) outsourcing?

5. Realized Project Measurements
Project measurements were to be used for day-to-day evaluation of project status and quality, and to determine completeness to trigger the review process. The process, measurements and tools were mutually interdependent. Without the process there could be no measurement, without the measurement, process progress and work product quality could not be measured, and without tool support, taking measurements would be extremely time consuming to the point of being impractical. The process and measurements described in this paper were used on several Siemens projects to elicit requirements for new products, including medical systems, mail sorting systems, material management systems, and building management systems; three of these projects were selected for this paper because they demonstrated the efficacy of the measurements in use. In each case, there were a very large number of requirements that had to be managed, as the products were complicated and to be built by large teams.

6. PROJECT DESCRIPTIONS
The three projects were executed using the measurement techniques described below. They were reasonably large, involving distributed teams with 7 or more analysts, typically 50-200 developers, with full time project management. These projects were considered large in size due to the length of time for development, number of requirements and complexity.

Project 1 (see Table 1) involved the definition of a next-generation mail sorting system for a federal postal service. Project 2 involved the creation of a master set of requirements for building management control software (e.g. fire, alarm, security, heating and air conditioning, etc.). Project 3 defined the requirements for financial software for Integrated HealthCare Networks.

Projects one and two defined products that contained both electromechanical and software components, the third product was pure software.

<table>
<thead>
<tr>
<th>Table 1 Project Information</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Project 1</th>
<th>Project 2</th>
<th>Project 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling Tool</td>
<td>Rose\textsuperscript{TM}</td>
<td>DOORS\textsuperscript{TM}</td>
<td>Caliber\textsuperscript{TM}</td>
</tr>
<tr>
<td>Requirements Database</td>
<td>RequisitePro\textsuperscript{TM}</td>
<td>179</td>
<td>804</td>
</tr>
<tr>
<td>Use Cases</td>
<td>40</td>
<td>224</td>
<td>1013</td>
</tr>
<tr>
<td>Use Case Diagrams</td>
<td>20</td>
<td>160</td>
<td>96</td>
</tr>
<tr>
<td>Actors</td>
<td>26</td>
<td>138</td>
<td>74</td>
</tr>
<tr>
<td>Avg. Use Cases per diagram</td>
<td>5.5</td>
<td>5.0</td>
<td>2.4</td>
</tr>
</tbody>
</table>

While the projects were large, lasting several months, projects 2 and 3 were big by any standard, whereas project 1 was closer to being a “typical” Siemens project. It was interesting to note that the measurements taken were effective in reporting progress and quality for all the projects, scaling quite nicely from “normal” to “big”.

7. MODEL-DATABASE RELATIONSHIP
The objective of each project was, of course, to define the requirements for new Siemens products. While it would have been very nice to keep all the requirements in the UML model, it was not practical. Once a certain level of detail was reached, requirements when shown on use case diagrams ceased providing useful information and became clutter. Samples of such requirements would include the color of a screen banner, the size and shape of a bolt or screw, the exact structure of an operator panel including toggle switch placement, etc. These requirements were placed only in the requirements repository (Figure 1). During the course of modeling, the following rules were adhered to:
• Features were shown as abstract use cases
• Testable requirements (deriving from features) were shown as concrete use cases
• Leaf testable functional requirements were shown using a special stereotype. They appeared on use case diagrams as gears
• Leaf non-functional requirements were shown using a special stereotype. They appeared on use case diagrams as oil cans.
• A terminal use case was one with no included, derived or extending use cases (only the include, generalization and extension relationships were permitted on use case diagrams).

8. PROJECT METRICS
In this paper, only metrics and measurements associated with the UML model are described, as it is a relatively new area. Requirement Metrics were available as part of the standard requirements database (RequisitePro™, DOORS™, Caliber™) products. Project metrics included those metrics associated with progress and status (as well as completeness). CMMI goals direct attention to status of project plans and resource management. Resource capability is a factor of how well resources perform tasks during these projects. The ability of some resources to complete work faster or slower can directly impact progress. Quality Metrics are described in the next section.

8.1. Use Case Status
The status of each use case was an attribute of the use case symbol, e.g. draft, accepted, preliminary, etc. Reports generated from the model periodically showed the status of each use case. This approach made it easy to track down incomplete or unreviewed use cases, and to observe the status on a diagram visually (the color of the use case could be programmatically changed based on the use case status). Business rules were defined and implemented for programmatically determining whether a use case could take a value of accepted:

• In order for a use case to have a status of “accepted”, each included, child or extending use case had to also have a status of “accepted” and
• It could have no errors as defined in sections 3.2.1 and 9.

8.2. Use Case Diagram Status
Diagram status was determined by the status of all the contained use cases:

The diagram status was set manually. Diagram status could not be set to accepted unless each of the use cases on the diagram had a status of accepted. In order for a model to be submitted for formal review, all diagrams in it had to have a status of accepted. At any given point in time, it was possible to determine the internal review status by getting the percent of use cases and use case diagrams that had been accepted.

8.3. Missing Boundary
One of the modeling rules was that all actors had to communicate with a concrete use case through a boundary. By enforcing this rule (through quality assurance scripts) it was possible at the conclusion of modeling to obtain an inventory of all the interfaces and their functions. Moreover, actors not communicating through boundaries provided a rough estimate of the UI (and software interface) requirements that had not yet been defined in the model. This error type also included the related errors:

• A boundary does not communicate with an actor
• A boundary does not communicate with a concrete use case.

8.4. Dangling Use Case
A dangling use case is a leaf use case that has not been marked as terminal, leaving the reader uncertain as to whether it is unfinished or whether further details appear only in the requirements database. As mentioned above, terminal use cases were given special stereotypes. The percentage of dangling use cases in the model provided a gross estimate of the subject areas that had not yet been covered. If the dangling use case was abstract it typically meant that a major subject area remained undefined.

9. QUALITY METRICS
Quality metrics are those which impact the quality of the completed model, but do not necessarily convey information about progress or productivity. While there are many metrics to choose from, a subset was selected as criteria for determining model completion, e.g. a model could not be declared complete and available for stakeholder review until the count for critical errors of these types went to zero. CMMI considers the quality of requirements as a value pre-requisite to completion of requirements development tasks. Quality metrics thresholds are set to ensure requirements errors are solved prior to moving forward in development.

9.1. Concrete Use Case Not Defined
The number of errors of this type was determined by evaluating each concrete, non-leaf use case to insure that it was described with one or more sequence, activity, or state diagrams.

9.2. Hidden Artifact
Hidden artifacts were items in the model database that did not appear on any diagram. They were not permitted.

9.3. Class not Instanced
An instance of each concrete class describing a business object had to appear on a sequence or activity diagram. If the object did
not appear, then the description of how the object was used was missing.

**Figure 2 Holistic Measure of Diagram Clarity**

9.4. Circular Dependency

Cycles were not permitted. For example, if use case A included use case B which included use case C, then use case C could not include use case A. This error was caught programmatically by treating the use case model as a directed graph [2].

9.5. Use Case Completeness

Use case completeness for non-terminal concrete use cases meant that any included or extending use cases would be shown on a sequence or activity diagram associated with the parent use case.

9.6. Illegal Association

Illegal associations included illegal extending and interface associations, e.g. the relationships had to be between concrete artifacts.

9.7. Mixed Use Case Relationships

The included/extending use cases for a parent use case had to be either all concrete or all abstract. If a use case had two included use cases, for example, one being concrete and the other abstract it was considered ambiguous, and as a result any requirements derived from those use cases might also be ambiguous.

10. QUALITATIVE METRICS ANALYSIS

The project and quality measurements described in 8 and 9 above, provided project progress and quality information respectively. However, in addition to raw percentages, it was possible to look at trends, and gain additional insight as to how the projects were doing. The project and quality measurements described in Project METRICS and Quality Metrics above, provided project progress and quality information respectively. However, in addition to raw percentages, it was possible to look at trends, and gain additional insight as to how the projects were doing. For example, in order to determine the clarity of the use case diagrams, statistics were collected on the density of major artifacts on the diagrams and then charted. As can be seen in Figure 2, it can be inferred from the metrics that the analysts on the project were skilled in creating useful diagrams [7].

**Figure 3 Accelerated Modeling Process**

11. PROJECT EXECUTION

The project and quality metrics described in this paper were viewed on a daily basis. The fact that the metrics were captured programmatically made the monitoring effort easy. The contrast between this approach and text-based requirements capture was startling; analysts received immediate feedback on the quality of their work, and processes were transparent to project management.

Progress in both projects was rapid. As the domain specialists gained expertise with the tools and the UML, they were able to continue the modeling effort unsupervised in small teams working in parallel, with model reviews conducted each week.

The use of tools to check for missing artifacts on activity and sequence diagrams dramatically improved the quality of those diagrams (describing business rules and logic) and resulted in very little rework.

As the models got larger, completeness checks had to be conducted programmatically. Every morning, the modeling team would query for incomplete use cases and diagrams. This was done every day until the number of errors reported went to zero (Figure 3). Internal reviews of the artifacts and diagrams were conducted both top down (diagrams) and bottom up (use cases). Once a use case was marked as accepted, it was not necessary to open it up, look at the hierarchy and owned diagrams as those would all have been previously accepted. The programmatic marking of diagrams and the formal definition of acceptance resulted in increased productivity as it was not necessary to descend the diagram trees to hunt for incomplete items.

A viable estimation of percent complete could be determined by looking at reported metrics:

- An orphan abstract use case meant that an entire subject area had not yet been covered
- Dangling use cases (paragraph 8.4) as a percent of the total number of use cases in the model gave a rough estimate of the work remaining prior to model review.
12. FROM USE CASE MODEL TO REQUIREMENTS REPOSITORY

At some point in the modeling effort a large number of requirements had been identified and marked using the stereotype mechanism. The stereotypes of "requirement" and "nf requirement" were added to the UML profile for use cases, and special icons, a "gear" and an "oil can" were used to identify use cases that were candidates both for review and for linkage to the requirements database [see Figure 4].

At this point two different approaches were used to move the requirements from the model to the requirements repository.

In project 1 the decision was made, because of the integration of Rose™ and RequisitePro™, to create the requirements repository directly from inside the model.

By selecting a use case in Rose™ it was possible to create the corresponding requirement in RequisitePro™ without leaving the Rose™ environment. At this point we ran into some difficulties with RequisitePro™:

A child requirement must have the same type as its parent in RequisitePro™. This prevented us from having mixed children, e.g. a requirement with lower level requirements of type security, performance, etc. The requirement type was set using an attribute. While this solution mitigated the problem, it did not completely solve it; it was not possible to determine the requirement type without selecting the requirement and inspecting it (or creating a matrix view).

The name of the use case in Rose™ became the description of the requirement. If the requirement description was changed in RequisitePro™, the coupled use case name would automatically change in Rose™ (a very undesirable side effect). The solution here was to use an attribute to hold the real requirement description, also a not completely acceptable solution.

Project 2 took a different approach. The requirements, along with the hierarchy, were programatically generated from the model [2], and a DOORS™ database was created. Because the algorithm creating the requirements hierarchy had to break relationships (e.g. where a use case is included in two other use cases only one can be its parent in the database) trace relationships were created in the DOORS™ database to retain the deleted links containing model structure. While a “one time” dump from the model to DOORS™ worked extremely well, the approach is not without real problems, and an open issue remains that of repairing the model should requirements in DOORS™ be reorganized.

In Project 3 the modeling took place after a significant number of requirements had already been defined. The model was used to verify the existing requirements, and then as a starting point for generating a new requirement set in Caliber™ linked to the model using the facilities that supported traces between Rose™ use cases and requirements in Caliber™:

1. The model was initially created.
2. The existing Caliber™ database was used as a checklist to verify that every requirement had been covered by one or more use cases.
3. The “old” requirements database was migrated to a new database where the hierarchical requirements organization reflected the use case structure in the analysis model.

This approach gave the team valuable insight into the significant advantages of a formal modeling approach. The analysis team observed that significant features in the “old” requirement set were missing, and in areas where there were requirements, they were insufficiently complete in many cases to pass to development and/or to use for the creation of a test plan.

13. CONCLUSIONS

For the projects, the definition of progress and quality metrics and the ability to programatically extract them dramatically improved the productivity of the requirements teams and the quality of the delivered requirements. These metrics were identified and used as a result of implementing a formal requirements process which has been shown to be effective in other large requirements efforts [8]. Moreover, the availability of
measurements on an on-demand basis changed the way the teams worked. Team leads came to rely on metrics for decision making. Model reviews tended to occur in smaller chunks with greater frequency (e.g. every morning) with a commensurate increase in quality.

The ability to mark items as complete in a bottom up fashion significantly reduced (in fact eliminated) the need to manually search the model looking for incomplete items and provided transparency in terms of progress to project management.

As the team members gained experience with the measurement tools and increased ability with the UML, their productivity and confidence rose dramatically.

Unfortunately, both UML modeling and requirements tools still have some real deficiencies. We are hoping that the next generation tool sets starting to appear will provide improvements over their predecessors, especially in the area of requirement-model integration and traceability.

14. References