Introducing CMMI Measurement and Analysis Practices into Scrum-based Software Development Process

Viljan Mahnic, Natasa Zabkar

***Abstract*—** Introduction of CMMI practices for Measurement and Analysis Process Area into Scrum is described with the aim of monitoring and improving software process performance. A meta-model of Scrum is given first, followed by the specifications of base and derived measures that can be used to monitor satisfaction of different stakeholders. Points on the process timescale are defined where the proposed measures are collected without harming the agility of Scrum. Finally, a solution for measurement repository design is described and attributes of the corresponding database tables are specified.

***Keywords*—**CMMI, measurement repository, Scrum, software measures

I. Introduction

NUMEROUS agile methods [1] have appeared in the last decade that – in contrast to disciplined approach advocated by the quality models like CMMI (Capability Maturity Model Integration [7]) – value individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan [16].

According to the research from March 2007 [2], 69% of 781 respondents work in companies that adopted one or more agile techniques. The Standish Group 2006 research report [28] states that 41% of agile projects succeeded as opposed to 16% of waterfall projects. Experience has also shown that adopting agile methods improves management of the development process and customer relationships [6], and decreases the amount of overtime and increases customer satisfaction [17].

According to [22] the most commonly used agile methods are XP [3] and Scrum [21]. More than 14.500 ScrumMaster certificates were issued since 2003 [23] and the list of companies using Scrum includes IBM, Microsoft, SAP, Google and Yahoo [24]. In the last few years several successful implementations of Scrum have been reported in the literature ([20, 27, 17, 13]).

At first glance, agile concepts seem to be in conflict with disciplined approach advocated by CMMI, but several authors suggest that it is possible to build organizational software process through a balance of agility and discipline ([4], [26]). Synergy between CMMI and Scrum has been explored by providing mappings between CMMI and Scrum practices [18] and criteria for defining appropriate agile measurement have been defined, pointing out that improper measures simply adopted from plan-driven approach not only waste resources but also skew team behavior in counter-productive ways and undermine culture change inherent in agile work [11].

The aim of this paper is to demonstrate how CMMI practices for Measurement and Analysis (MA) Process Area can be introduced without harming the agility of a Scrum-based software development process.

We focus on the following specific practices:

- SP 1.1 Establish Measurement Objectives
- SP 1.2 Specify Measures
- SP 1.3 Specify Data Collection and Storage Procedures
- SP 1.4. Specify Analysis Procedures.

After a short description of Scrum concepts and meta-model we present mapping between CMMI and Scrum for the aforementioned practices. Based on our previous research in the Scrum usage and performance measurement ([13], [14], [15]), we then propose a set of base and derived measures that can be used to monitor satisfaction of different stakeholders involved in the software development process. Points on the process timescale are described where the proposed measures are collected, and procedures for deriving appropriate performance indicators are specified.

After measures definitions, a generic data model of measurement repository for collecting and storing measurement results is presented. Then a detailed description of database tables structures is given. This is followed by the directions for further research in the area of business intelligence, especially performance dashboards.

Manuscript received April 15, 2007; Revised version June 7, 2007

V. Mahnic is with the Faculty of Computer and Information Science, University of Ljubljana, Trzaska 25, SI-1000 Ljubljana, Slovenia (phone: 386-1-4768-447; e-mail: viljan.mahnic@fri.uni-lj.si).

N. Zabkar is Ph. D. student at Faculty of Computer and Information Science, University of Ljubljana Trzaska 25, SI-1000 Ljubljana, Slovenia (e-mail: nzabkar@email.si).

II. Scrum Overview

A. Process Description

Scrum [21] is an iterative and incremental software development method driven by the Product Backlog list, which contains all active product requirements. The Product Backlog is managed by Product Owner, who is the only person authorized to change priorities of the requirements.

All work is done in Sprints. Each Sprint is an iteration of 30 consecutive calendar days and is initiated with a Sprint planning meeting, where the Sprint Backlog is agreed. This is a subset of Product Backlog requirements that defines functionality to be developed in the current Sprint. Every requirement is further broken into tasks that each takes roughly 4 to 16 hours to finish.

Functionality is developed by the Team, i.e. a group of developers that are collectively responsible for the success of each iteration and of the project as a whole. Teams are self-managing, self-organizing, and cross-functional, and they are responsible for figuring out how to turn Product Backlog into an increment of functionality within an iteration.

The ScrumMaster is responsible for managing the Scrum process so that it fits within an organization's culture and still delivers the expected benefits, and for ensuring that everyone follows Scrum rules and practices. Every day ScrumMaster leads a 15-minute Daily Scrum meeting where every Team member answers three questions: What have you done on this project since the last Daily Scrum Meeting? What will you do before the next meeting? Do you have any obstacles? ScrumMaster is also responsible for resolving impediments encountered during the Sprint in order to assure smooth running of the development process.

At the end of the Sprint, a Sprint review meeting is held at which the Team presents Sprint results to the Product Owner. After the Sprint review and prior to the next Sprint planning meeting, the ScrumMaster also holds a Sprint retrospective meeting in order to ensure continuous improvement.

B. Meta-model of Scrum

For the purpose of introducing appropriate measures we present a meta-model of Scrum using the entity-relationship notation. The meta-model is further expanded in Section IV in order to describe the design of the measurement repository that serves for storing project data and measurement results obtained during the development process. In Section V the suggested structure of the database tables is presented.

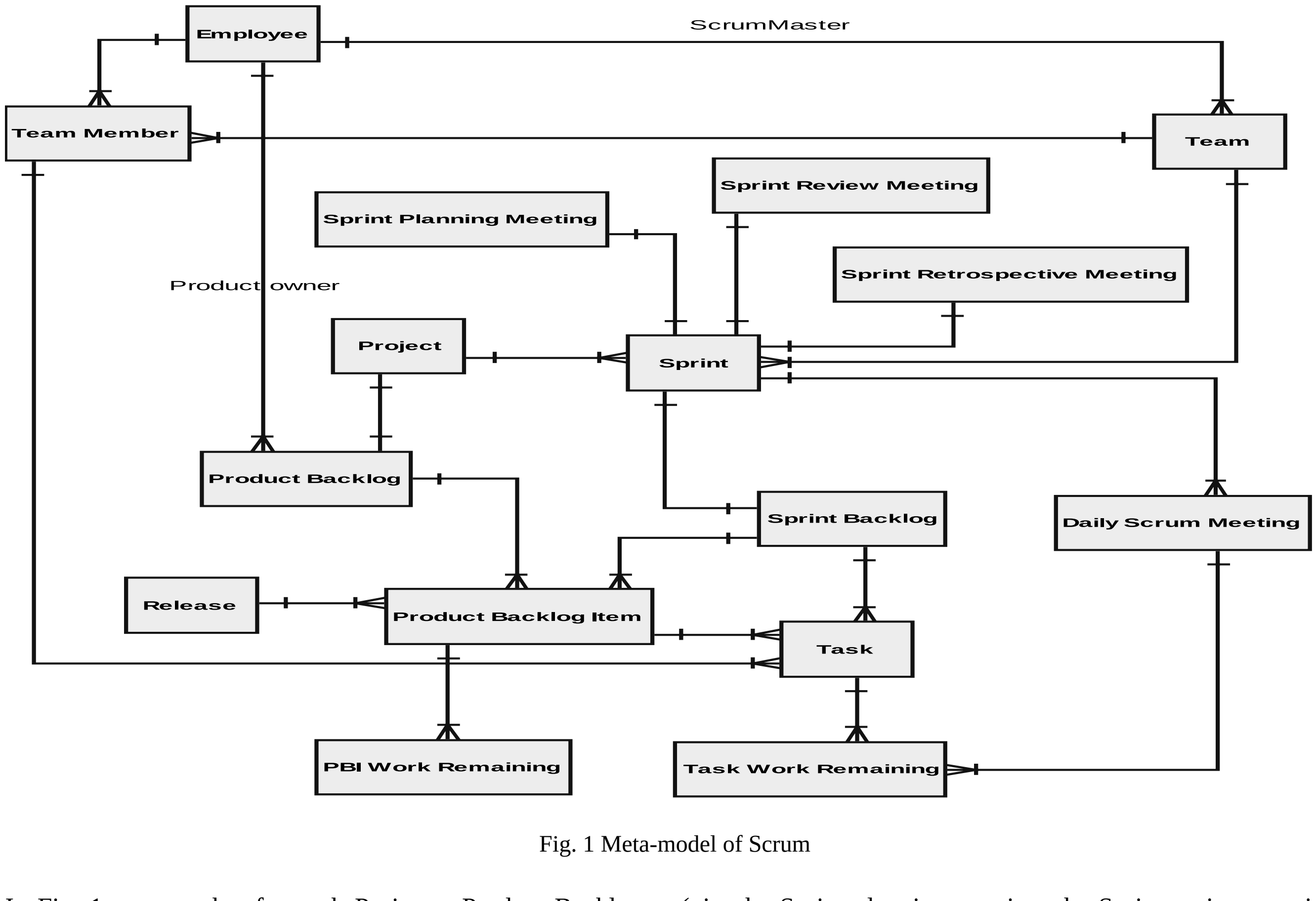

Fig. 1 Meta-model of Scrum

In Fig. 1, we see that for each Project a Product Backlog must exist that contains several Product Backlog items (PBI). For each PBI one or more estimates of work remaining are provided. The Project is implemented through several Sprints.

For clarity reasons, the events associated with each Sprint (viz. the Sprint planning meeting, the Sprint review meeting, the Sprint retrospective meeting, and the Daily Scrum meetings) are shown as separate entities. For each Sprint a Sprint Backlog must be maintained that corresponds to PBIs the Team committed to implement during the Sprint. Each PBI is split into several tasks and for each task the estimates of work remaining are provided on daily basis. The implementation of a Sprint is appointed to a Team that consists of several members each of them being responsible for several tasks. For each project a Product Owner must be assigned (either being an employee or a customer representative) and for each Team a ScrumMaster must exist. Several Sprints can be combined into a release; however, the experience of the developers of some tools that support Scrum process [e.g., 8] has shown that a mandatory relationship between Sprints and a release represents a problem when a Team works on multiple releases within the same Sprint. Therefore, the relationship between Sprint and release entities is only provided through PBIs.

III. CMMI Practices Implementation

A. *Establish Measurement Objectives*

CMMI provides examples of different measurement objectives like reduce time to delivery, reduce total lifecycle cost, deliver specified functionality completely, improve prior levels of quality, improve prior customer satisfaction ratings, etc. Achievement of all these objectives can be measured through continuous monitoring of the software process performance. Therefore, the implementation of CMMI Measurement and Analysis practices described in this paper is based on the assumption that the main measurement objective is to monitor and improve software process performance.

Kueng [12] defines process performance as "the degree of stakeholder satisfaction"; consequently, in order to monitor and evaluate process performance we must consider views of different stakeholders that take part in the process. The best performance is achieved when the goals of all stakeholders are satisfied. The achievement of goals should be measured quantitatively and qualitatively, thus giving a comprehensive view of the process performance. Based on our previous research [15], we have identified four stakeholders for Scrum process: IT management, Team members, ScrumMaster and Customers. Their goals are shown in Table I.

TABLE I
Stakeholders' Goals

Stakeholder	Goal
IT management	Timely information on project performance Quality improvement
Team members	Job Satisfaction
ScrumMaster	Efficient Impediments Resolution
Customers	Customer Satisfaction

IT management is mainly concerned with traditional aspects of software development performance considering time, cost, and quality. The main goal of Team members is "Job satisfaction". Team members are most productive if they have good working conditions enabling a sustainable pace of progress without excessive workload and working overtime. The ScrumMaster's main role is facilitating the use Scrum and creating conditions for smooth running of the development process; therefore, his main goal is "Efficient impediments resolution". From the perspective of a software development organization the main goal regarding customers is "Customer Satisfaction".

B. *Specify Measures*

According to CMMI, measures may be either "base" or "derived". While data for base measures are obtained by direct measurement, data for derived measures come from other data, typically by combining two or more base measures. Derived measures serve as performance indicators showing the achievement of particular goals. In this subsection only base measures are described. Derived measures will be presented in subsection III.D.

Originally, Scrum had only one base measure: the estimate of the amount of work remaining that needs to be done in order to complete a Product Backlog item or a task in the Sprint Backlog (SB). At the task level, this measure is collected every day for each task in the Sprint Backlog separately. At the PBI level, the amount of work remaining for each PBI is estimated at the beginning of each Sprint. Using this measure, burndown charts can be developed showing work remaining over time. The Scrum books define a Sprint Burndown chart as a place to see daily progress, and a Product Burndown chart as where to show monthly (per Sprint) progress.

In order to measure the achievement of stakeholders' goals identified in III.A, we have defined some additional measures as shown in Table II. The proposed measures can be introduced stepwise giving each software development organization freedom to adapt the measurement plan to its specific needs. Nevertheless, we suggest the amount of work spent metric to be introduced first since it fits to the concept of Daily Scrum meetings and is analogue to the estimate of the work remaining metric already proposed by Scrum.

TABLE II
Base Measures

Goal: Timely information on project performance
Work remaining on day d for each task in the SB Work spent on day d for each task in the SB
Goal: Quality improvement
The number of errors found during the Sprint review meeting (for each PBI separately) The number of errors reported by the user in a fixed period after release (for each PBI separately) The size of the code (for each PBI separately) Total number of PBIs committed in the release/Sprint The number of PBIs completed in the release/Sprint Total number of tasks in the Sprint The number of tasks completed during the Sprint
Goal: Job satisfaction
Administrative days Results of the survey (Job Satisfaction) conducted at the Sprint retrospective meeting. Each question is marked between 1 and 5, where 1 is the worst and 5 is the best mark.

Goal: Efficient Impediments Resolution
The number of impediments that refer to the given Task/Sprint/Team The date the impediment was encountered The date the impediment was resolved
Goal: Customer Satisfaction
Results of the survey (Customer Satisfaction) conducted at the end of each Sprint/release. Each question is marked between 1 and 5, where 1 is the worst and 5 is the best mark.

C. *Specify Data Collection and Storage Procedures*

All base measures proposed in the Table II (including some basic parameters that must be established at the beginning of each Sprint) can be easily collected during meetings already prescribed by Scrum. The only exception is the number of errors reported by the user after release. Base measures collected at each type of meeting are shown in Table III.

TABLE III
DATA COLLECTION POINTS

Sprint planning meeting
Sprint length (number of working days in the Sprint) The number of Team members (the size of Team) Percentage of Team member's engagement in the project Cost of each Team member's engineering hour
Daily Scrum meeting
Work remaining on day d for each task in the SB Work spent on day d for each task in the SB Administrative days Impediment data
Sprint Review meeting
The number of errors found during the Sprint review meeting (for each PBI separately Results of the survey (Customer Satisfaction) conducted at the end of each Sprint/release
Sprint Retrospective meeting
The size of the code (for each PBI separately) Total number of PBIs committed in the release/Sprint The number of PBIs completed in the release/Sprint Total number of tasks in the Sprint The number of tasks completed during the Sprint Results of the survey (Job Satisfaction)

At the Sprint planning meeting the values of the basic parameters must be established: the Sprint length, composition of the Team (the number of the Team members, percentage of each Team member's engagement in the project), and costs of each Team member's engineering hour.

At Daily Scrum meetings the Sprint Backlog is maintained. For each task Team members report the amount of work spent and estimate the amount of work remaining. The amount of work spent is obtained simply when each Team member answers the question what he/she has done on the project since the last Daily Scrum. If a new task is added, the type of work performed and the cost of the engineering hour must be defined. For Team members not present the administrative days are recorded.

During the Sprint review meeting the number of errors reported by the user is recorded and a survey of customer satisfaction can be done.

During the Sprint retrospective meeting the code size of each PBI is measured and the numbers of PBIs/Tasks committed, but not completed are determined. However, these numbers can be computed on spot by an appropriate project management tool. At this meeting the survey of job satisfaction can also be done.

The computation of indicators is best done by an appropriate project management tool. Since tasks in the Sprint Backlog emerge as the Sprint evolves (e.g., a task that was only roughly defined at the beginning is split into several smaller ones) the tool should maintain a list of active tasks and keep history of all changes in order to compute the indicators properly.

D. *Specify Analysis Procedures*

Base measurement data are grouped into derived measures or indicators that serve for analyzing software process performance in comparison to target values set by software development organization.

Achievement of the goal "Timely Information on Project Performance" is analyzed using the following indicators:

- Work Effectiveness,
- Schedule Performance Index (SPI), and
- Cost Performance Index of labor costs (CPI).

Work effectiveness refers to the ratio between the decrement of work remaining and the amount of work spent. Ideally, the decrement of work remaining between days *d1* and *d2* of a Sprint should be equal or greater to the amount of work spent in the same interval. Therefore, the target value of this indicator is 1 or more; however, values significantly greater than 1 may be a sign of poor planning.

Schedule Performance Index (SPI) refers to the ratio between the earned value (i.e., the value of all tasks completed) and the planned value (i.e., the initial estimate of value of all tasks to be completed till a certain point within the project). The target value for SPI is 1 or more. SPI greater than 1 means that the project is ahead of schedule.

Cost Performance Index of labor costs (CPI) refers to the ratio between the earned value (measured in units of currency) and actual costs. The target value for CPI is 1 or more, indicating that the cost of completing the work is right on plan or less than planned.

Indicators for goal "Quality Improvement" are:

- Error density, which refers to number of errors per KLOC (kilo-lines of code),
- Costs of rework, which refers to the product of hours spent on rework and cost of an engineering hour,
- Fulfillment of scope, which shows if all Product Backlog Items (PBIs) and Sprint Backlog Tasks have been implemented, and refers to the ratio between the number of tasks completed in the Sprint and total number of tasks in the Sprint Backlog or between the number of PBIs completed in

the release and total number of PBIs committed.

Achievement of the goal “Job Satisfaction” is measured quantitatively and qualitatively through the following indicators:

• The average amount of overtime at Sprint/release/project level considering the expected hours, the amount of work spent and administrative days,

• The average number of projects the employees work in parallel,

• Qualitative evaluation of working conditions like communication and teamwork, physical discomfort, psychological well-being, workload, supervision, opportunities for growth, etc.

ScrumMaster’s goal “Efficient Impediments Resolution” is measured by computing the average number of impediments per Task/Sprint/Team and the mean time for resolving an impediment (at Task/Sprint/Team level).

Indicators for goal “Customer Satisfaction” are measured quantitatively and qualitatively. The quality of product and the completeness of product delivered at the end of each Sprint or release can be expressed in terms of quality improvement indicators “error density” and “fulfillment of scope”. Values of qualitative indicators are gathered from the survey allowing the customers expressing their subjective opinion regarding:

- price adequacy,
- reliability in terms of time and costs,
- flexible handling of changes in requirements,
- good collaboration with the development team,
- adequate training and documentation, etc.

In the subsection III.E we provide formulae for computation of Schedule Performance and Cost Performance indices. Detailed descriptions and formulae for evaluation of other aforementioned indicators can be found in [15].

E. Measuring Earned Value

Adapting the Earned Value Management method [19] for Scrum projects is a challenge that has not been completely resolved yet (e.g., [5], [25]). We propose to compute the Schedule Performance and Cost Performance indices using the work remaining and work spent measures defined in III.B. Since Scrum does not prescribe the project schedule model, we assume that the amount of tasks that must be accomplished at a certain point in the Sprint is proportional to the time elapsed from the beginning of the Sprint. The work remaining and work spent measures allow a precise definition of the earning rule $ER_{d,j}$ for each task j in the Sprint Backlog on the day d of a Sprint. It can be computed as a ratio between the amount of work already spent and all the work required (spent and remaining) to accomplish the task:

$$ER_{d,j} = \frac{\sum_{i=1}^{d-1} WS_{i,j}}{\sum_{i=1}^{d-1} WS_{i,j} + WR_{d,j}} \quad . \tag{1}$$

$WS_{i,j}$ denotes the amount of work spent for task j on day i, $i=1,2,\ldots,d-1$, and $WR_{d,j}$ denotes the amount of work remaining for task j on day d.

Using the earning rule from formula (1), the SPI on day d is computed as

$$SPI_d = \frac{\sum_{j=1}^{n} ER_{d,j} \cdot WR_{init,j}}{\sum_{j=1}^{n} WR_{init,j}} \cdot \frac{SL}{DE} \tag{2}$$

where $WR_{init,j}$ denotes the initial estimate of the work remaining for task j, SL the Sprint length, and DE the number of days elapsed.

While the computation of SPI allows the earned value to be measured in any of the units (we use the initial estimates of hours of the work remaining for each task j in the Sprint Backlog) the computation of CPI requires the earned value and actual costs to be expressed in units of currency. Using the work spent measure, we can compute the actual labor costs exactly by multiplying hours spent and the cost of an engineering hour CEH_j for all tasks in the Sprint Backlog. Similarly, the earned value is computed by multiplying the earned hours and CEH_j. CPI for labor costs is then computed as a ratio between earned value and actual labor costs as shown in formula (3):

$$CPI_d = \frac{\sum_{j=1}^{n} ER_{d,j} \cdot WR_{init,j} \cdot CEH_j}{\sum_{i=1}^{DE}\sum_{j=1}^{n} WS_{i,j} \cdot CEH_j} \quad . \tag{3}$$

IV. Repository Design

In this section we present data model of the measurement repository for storing data that arise during the software development process (see Fig. 2). The data model is derived from the meta-model in Fig. 1 by adding entity types that describe appropriate measures and enable the accommodation of measurement results. Beside, some new entity types are introduced in order to enable impediments tracking, describing the classification of tasks (regarding the type of work performed and current status), and keeping records of administrative days when a Team member is not at work. The model serves as a logical data model of the repository database, each entity type representing a corresponding database table. Detailed structure of database tables is presented. in Section V.

The model is generic and does not prescribe in advance the kind and number of measures, thus enabling a stepwise introduction of the measurement program. New measures can be simply added and the measures that are no more needed or proved to be useless can be simply removed. The only prerequisite is that all measurement results are of the same type (viz. numeric). Each measure is represented as an instance of the `Measure` entity type (i.e., as a row of the corresponding relation containing measure key, name, description and other attributes), while the measurement results are stored in different tables, depending on the level and point in the process they are collected. E.g., values that are measured at the Task/PBI/Sprint/Release level are stored in the `Task/PBI/Sprint/Release Measurement Result` table. Each row of these tables contains a compound primary key (a part of which is the measure key) and the measurement result.

Impediments tracking is introduced in order to provide data required for the computation of the average number of impediments per Task/Sprint/Team and the mean time for resolving an impediment (at Task/Sprint/Team level). Each impediment is described within the `Impediment` table containing the impediment key, description, the dates when the impediment occurred and when it was resolved, and foreign keys providing relationship with the `Team` that encountered the impediment, the `Sprint` in which the impediment was encountered, and the `Employee` who was responsible for the resolution.

The classification of the type of work performed is necessary if we want to track the amount of different kinds of work during each Sprint, e.g., development, testing, rework due to error reported by the customer, rework due to the change in requirements, etc. Each row of the `Task Type` table defines one of the aforementioned types of work, thus allowing each organization to specify the classification that best suits its needs. For the proper functioning of the measurement system it is necessary that each task in the Sprint Backlog is assigned the corresponding task type.

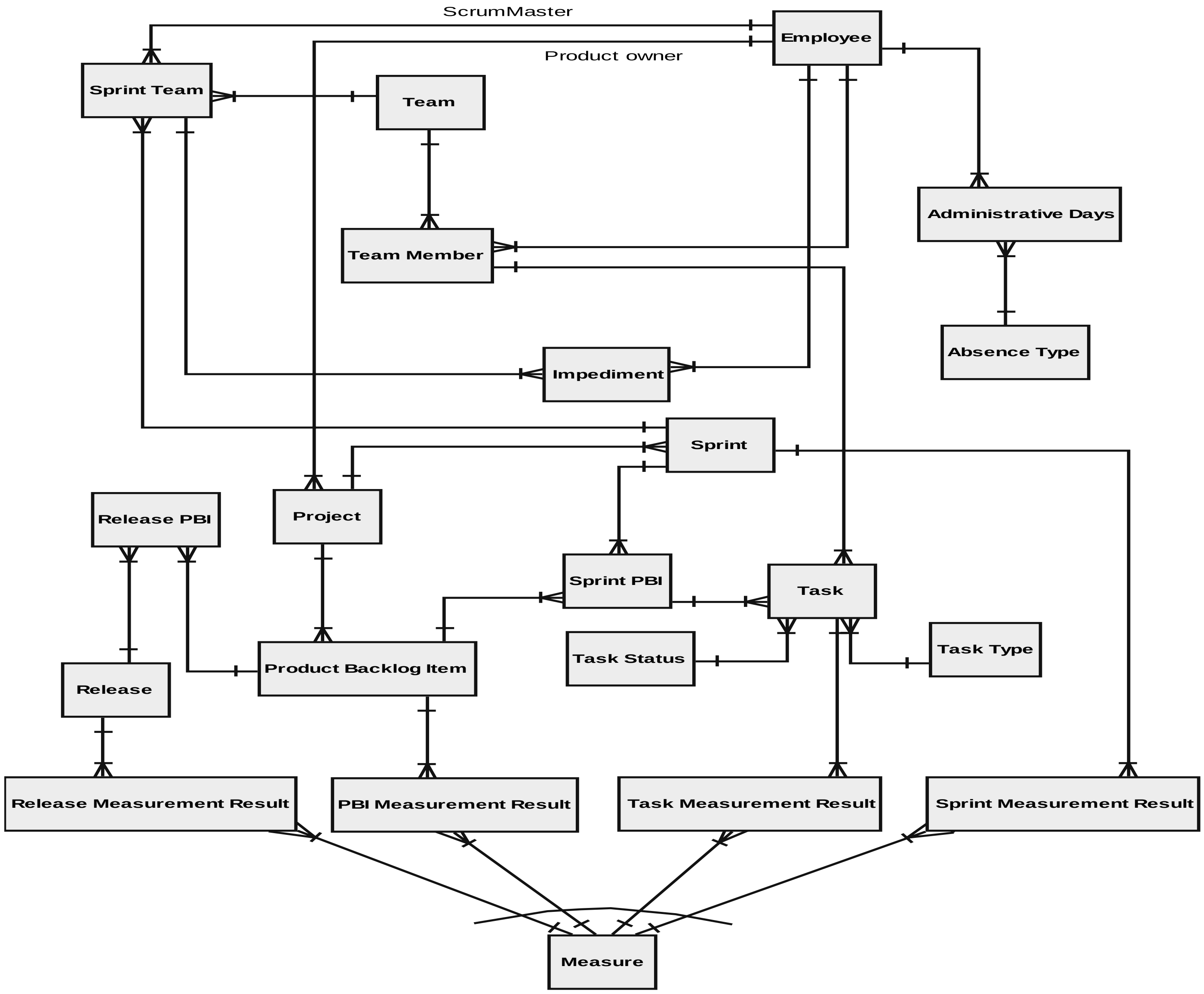

Fig. 2 Measurement repository design

In a similar way the classification of tasks according to their current status is introduced. The rows of the `Task Status` table describe all possible statuses of a Sprint Backlog task, e.g., not started, in progress, completed, omitted, moved into next Sprint, etc. Again, each organization is allowed to specify different possible statuses according to its needs. The ScrumMaster maintains the status of each task during the Daily Scrum Meeting, at the same time when the amount of the work spent and the estimate of the work remaining for the task is entered.

Keeping records of administrative days (viz. sick days, vacation, course days, compassionate leave etc.) when a Team member is not at work enables exact calculation of overtime. When a Team member is absent (and consequently does not attend the Daily Scrum meeting), the ScrumMaster simply records his absence as a new row in the `Administrative Days` table. The `Absence Type` table is necessary if we want to track different types of absence more in detail.

In order to calculate the labor costs precisely, the hourly rate of each Team member must be provided. To avoid problems with tracking history if the hourly rate changes frequently, it seems to be the best solution to record this attribute within the `Task` table at the time when the task is created and a Team Member assigned to it.

A careful reader has also noticed that (in comparison to Fig. 1) all 1:1 relationships were removed and corresponding entity types merged into a single entity type. Additionally, the 1:N relationships connecting `Product Backlog Item` to `Release` and `Sprint` were changed to M:N relationships in order to accommodate peculiar situations when the implementation of a Product Backlog Item requires more than one Release or Sprint. The M:N relationships were removed by the introduction of the `Release PBI` and `Sprint PBI` entity types. Similarly, the `Sprint Team` entity type was introduced in order to support the concept of Scrum of Scrums that allows several Teams to work on the project within the same Sprint.

V. Database Tables

In this section we present one possible solution for the more detailed level of the previously introduced data repository design. Entity types from Fig. 2 are presented in the group of tables: Project Tables, Release Tables and Measurement Tables, where the attributes are specified for each database table.

First, the suggested attributes for the Project Tables (Table IV) are presented. These tables describe relationships among projects, teams and employees, and include recording of administrative days and absence type. For each administrative day the number of hours the employee was not at work is recorded.

TABLE IV
PROJECT TABLES

Table	Attributes
Project	(Project ID#, Project Description)
Employee	(Employee ID#, Employee Description)
Sprint Team	(Sprint ID#, Team ID#)
Team	(Team ID#, Team Description)
Team Member	(Team ID#, Employee ID#, Percentage of Engagement in the Project)
Administrative Days	(Employee ID#, Date#, Hours Not Worked, Absence Type ID#)
Absence Type	(Absence Type ID#, Absence Type Description)

A possible structure of the Release Tables is proposed in the Table V. These tables are related to release development and include release, sprint and task level management of product backlog items (PBI).

TABLE V
RELEASE TABLES

Table	Attributes
Release	(Release ID#, Release Description)
Release PBI	(Release PBI ID#, PBI ID#)
Sprint	(Sprint ID#, Sprint Description, Sprint Begin Date, Sprint End Date, Sprint Length, Sprint Estimated Date, Team ID#, Project ID#)
Sprint PBI	(Sprint ID#, PBI ID#, Sprint PBI Priority, Sprint PBI Status, Task ID#)
PBI	(PBI ID#, PBI Description, PBI Priority, PBI Category, PBI Status, Project ID#, Release ID#, Sprint ID#)
Task	(Task ID#, Task Description, Task Cost of Engineering Hour, Task Date, Task Active, Task Type ID#, Task Status ID#, PBI ID#, Sprint ID#, Team ID#, Employee ID#)
Task Status	(Task Status ID#, Task Status Description)
Task Type	(Task Type ID#, Task Type Description)
Impediment	(Impediment ID#, Impediment Description, Impediment Occurrence Date, Impediment Resolution Date, Sprint ID#, Team ID#, Employee ID#)

Finally, Table VI contains suggested attributes for the Measurement Tables. The history aspect is provided through the Date attribute, so that reporting can be performed using the data that were active at the selected point in time. The measurement results for the base measures form basis for the calculation of the derived measurement results.

TABLE VI
MEASUREMENT TABLES

Table	Attributes
Measure	(Measure ID#, Measure Name, Measure Description)
Release Measurement Result	(Release ID#, Measure ID#, Date#, Measurement Result)
Sprint Measurement Result	(Sprint ID#, Measure ID#, Date#, Measurement Result)
PBI Measurement Result	(PBI ID#, Measure ID#, Date#, Measurement Result)
Task Measurement Result	(Task ID#, Measure ID#, Date#, Measurement Result)

The results of this paper present the basis for the research of the use of modern concepts of business intelligence [10] in the area of agile software development. Data described by the proposed generic data model can be used as the main input to a data warehouse and performance dashboards [9].

VI. Conclusion

In this paper we presented the design of measurement repository that enables monitoring and continuous improvement of the performance of a Scrum-based software development process. Base and derived measures were defined considering characteristics of Scrum and practices prescribed by the Measurement and Analysis Process Area of CMMI. In order to preserve agility, all measures have been chosen in such a way that can be collected during meetings already prescribed by Scrum, thus not requiring a substantial additional effort of the Team. Derived measures serve as indicators for measuring achievement of goals of different stakeholders that are involved in software development process.

The proposed measures are incorporated in the suggested solution for data repository design. The more detailed data repository design is presented including the database tables and the most important attributes. Each measure is represented as an instance of the `Measure` entity type while the measurement results are stored in `Release`, `Sprint`, `PBI` and `Task Measurement Results` tables.

The results of this paper can be used when developing data warehouse containing all data pertaining to the software development process organized in a way that enables detailed analyses through drilling-down and reporting techniques of business intelligence. Using indicators described in Section III, performance dashboards can be developed, providing real-time information of the software process performance and thus enabling an immediate reaction in the case of deviation from target values. We hope that the presented approach will be helpful for the further research of the use of business intelligence in the area of Scrum-based development process.

References

[1] P. Abrahamsson, O. Salo, J. Ronkainen, J. Warsta, *Agile software development methods.* VTT Electronic, Espoo, 2002.

[2] S. Ambler (2007, December 4). March 2007 Agile Adoption Survey [Online]. Available: www.agilemodeling.com/surveys/

[3] K. Beck, *Extreme Programming Explained.* Addison-Wesley, 2000.

[4] B. Boehm, R. Turner, *Balancing Agility and Discipline – A Guide for the Perplexed*. Pearson Education, 2004.

[5] A. Cabri, M. Griffiths, "Earned Value and Agile Reporting," in *Proceedings of AGILE 2006 Conference (AGILE'06)*, pp. 17-22.

[6] M. Ceschi et al., "Project Management in Plan-Based and Agile Companies," *IEEE Software*, May/June 2005, pp. 21-27.

[7] *CMMI, CMMI® for Development, Version 1.2. CMU/SEI-2006-TR-008,* Software Engineering Institute, Carnegie Mellon University, 2006.

[8] Danube (2008, February 9). ScrumWorks Basic User Guide [Online]. Available : http://www.danube.com/docs/scrumworks/1.8.2/userguide.html

[9] W. Eckerson, *Performance Dashboards*. John Wiley & Sons, Inc., 2006.

[10] M. Golfarelli et al., "Beyond Data Warehousing: What's Next in Business Intelligence?," in *Proc. DOLAP'04*, Washington, DC, 2004.

[11] D. Hartmann, R. Dymond, "Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value," in *Proceedings of AGILE 2006 Conference (AGILE'06)*, pp. 126-134.

[12] P. Kueng, "Process performance measurement system: a tool to support process-based organizations," *Total Quality Management*, Vol. 11, No. 1, 2000.

[13] V. Mahnic, S. Drnovscek, "Agile Software Project Management with Scrum," in *Proc. EUNIS 2005 Conference – Session papers and tutorial abstracts*, University of Manchester, 2005.

[14] V. Mahnic, S. Drnovscek, "Introducing agile methods in the development of university information systems," in *Proc. 12th International Conference of European University Information Systems EUNIS 2006*, Tartu, 2006.

[15] V. Mahnic, I. Vrana, "Using stakeholder driven process performance measurement for monitoring the performance of a Scrum based software development process," *Electrotechnical Review*, Ljubljana, Vol. 74, No. 5, 2007, pp. 241-247.

[16] Manifesto for Agile Software Development [Online]. Available http://www.agilemanifesto.org/

[17] C. Mann, F. Maurer, "A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction," in *Proceedings of the Agile Development Conference (ADC'05)*, pp. 70-79.

[18] A. S. Marcal et al., "Mapping CMMI Project Management Process Areas to SCRUM Practices," in *Proc. SEW 2007, 31st Annual Software Engineering Workshop*, Loyola College, Baltimore, MD, 2007.

[19] F. Quentin, J. Koppelman, *Earned Value Project Management.* Third Edition, Project Management Institute, 2005.

[20] B. Schatz, I. Abdelshafi, "Primavera Gets Agile: A Successful Transition to Agile Development," *IEEE Software*, May/June 2005, pp. 36-42.

[21] K. Schwaber, *Agile Project Management with Scrum*. Microsoft Press, 2004.

[22] C. Schwaber et al. (2007, December 3). The Truth About Agile Processes [Online]. Available: www.forrester.com

[23] Scrum Alliance (2007, September 28). Scrum Alliance Membership Survey Shows Growing Scrum Adoption and Project Success [Online]. Available: www.scrumalliance.org

[24] Scrum Alliance (2007, December 4). Firms Using Scrum [Online]. Available: http://scrumalliance.pbwiki.com/Firms-Using-Scrum

[25] Sulaiman, B. Barton, T. Blackburn, "AgileEVM - Earned Value Management in Scrum Projects," in *Proceedings of AGILE 2006 Conference (AGILE'06)*, pp. 7-16.

[26] J. Sutherland et al., "Scrum and CMMI Level 5: The Magic Potion for Code Warriors," in *Proc. AGILE 2007.*

[27] B. Upender, "Staying Agile in Government Software Projects," in *Proceedings of the Agile Development Conference (ADC'05)*, pp. 153-159.

[28] T. Wailgum (2007, December 3). From Here to Agility [Online]. Available: http://www.cio.com

Viljan Mahnic received his B. Sc. Degree in Computer Science in 1978 at the Faculty of Computer and Information Science at the University of Ljubljana in Slovenia, where he got his M. Sc. Degree in 1981 and Ph. D. degree in 1990.

He is currently an associate professor at the Faculty of Computer and Information Science at the University of Ljubljana, where he teaches courses on computer programming, software engineering and information systems. His research interests include software engineering and information systems development. He participated in several international projects within Tempus and INCO Copernicus programs, published several books on programming (in Slovene) and more than 70 papers in journals and scientific conferences.

Dr. Mahnic is member of the Board of Directors of EUNIS (European University Information Systems Association) since 2002.

Natasa Zabkar received her B. Sc. Degree in Computer Science in 1989 at the Faculty of Computer and Information Science at the University of Ljubljana in Slovenia and her M. Sc. Degree in Management and Organization in 1998 at the Faculty of Economics at the University of Ljubljana.

She is currently Ph. D. student at the Faculty of Computer and Information Science at the University of Ljubljana. Her research interest is software engineering and information systems auditing. She holds CISA (Certified Information Systems Auditor) designation since 2001. She has published more than 20 papers in different conferences.

N. Zabkar is member of ISACA (Information Systems Audit and Control Association) since 2000.