Requirements Eng (2015) 20:23–44
DOI 10.1007/s00766-013-0182-7

ORIGINAL ARTICLE

Reuse of requirements reduced time to market at one industrial shop: a case study

Leah Goldin · Daniel M. Berry

Received: 27 December 2011 / Accepted: 20 August 2013 / Published online: 27 September 2013

Abstract Many a computer-based system (CBS) developed in one division of the Israel Aerospace Industries, is a derivation of formerly delivered CBSs. Sometimes, a CBS that was developed over decades needs to be redesigned for a new customer or for new technology. In addition, the requirements management process in the division has to deal with an extensive system of systems, a matrix organizational structure, and many subcontractors in order to develop any fully operational CBS. It has become obvious that any generic building block in any product developed by any project in a project family stands a chance to be reused to build new products in other projects of the same project family. Moreover, this reuse is best if it begins with the consideration of the reuse of a requirements specification of the building block, in order to save the time and money of the block's development. The reused artifact can be the contract description of the building block, from very early in the lifecycle. Alternatively, the reused artifact can be detailed hardware and software specifications for the building block, from later in the lifecycle. With which specification reuse begins depends on the purpose of the reuse. No matter when a reuse began, it led to some significant reduction in the product's time to market, i.e., its time to delivery. This paper is a case study of several instances in an industrial setting of building-block reuse within a large-scale system being developed in the division. It reports how these reuses were discovered and carried out. Among the contributions of this paper is a description of the division's multilevel requirements hierarchy that is based on the multilevel architecture of the CBS to be developed. Also contributed the division's idea of project-family-based reuse, as opposed to product-line-based reuse. The paper shows how basing reuse on a project family made proactive reuse possible. The paper then analyzes data from the division's requirements-management tool about these reuses to assess the impact of requirements reuse on time to market. Finally, it confirms this quantitative assessment by a qualitative assessment garnered from quotations gathered from interviews of four key people two-and-a-half years after the end of the period of the case study.

Keywords Large-scale system · Management · Project family · Process · Requirements · Requirements management · Reusable software · Reuse · Specifications · Tools

This paper is an enhancement of a paper titled "Reuse of Requirements Reduces Time to Market" [1]. The conference paper has been extended by more detailed explanations of the company's project structure, of the studied project, of the data, and by lessons learned from interviews of key personnel.

L. Goldin
Department of Software Engineering, Afeka—Tel Aviv Academic College of Engineering, Tel Aviv 69107, Israel
e-mail: l_goldin@computer.org

D. M. Berry (✉)
Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
e-mail: dberry@uwaterloo.ca

1 Introduction

The goal of this paper is to show how reuse of managed requirements artifacts measurably helped reduce the time to produce a very large software-intensive system at one industrial organization. The paper achieves this goal by presenting an after-the-fact case study, including gathered metrics, of one development project at the organization.

The paper estimates the amount of reuse in this project using data gathered after the fact from this project and from two other related projects that serve as a comparison baseline. It reports also quotations from interviews of key requirements-management (RM) personnel from the organization conducted two-and-a-half years after the period of the case study. These quotations qualitatively confirm the quantitative conclusions.

The paper shows also that the tool-based RM process enacted in the organization plus storage of all relevant requirements and development artifacts both enabled and encouraged requirements reuse. When coupled with the anticipation of potential reductions in time to market as a result of proactive, rather than opportunistic, requirements reuse, requirements reuse began to consistently happen, and the anticipated time reductions materialized. Basically, any product is developed in a single project, using building blocks from multiple product lines. Each project acts selfishly, focusing on developing its own high-quality product as rapidly as possible. It both (1) makes high-quality building blocks for its own use and (2) actively seeks to reuse high-quality building blocks from other projects to reduce its own duration. If all projects behave this way, and each project's lead engineers are proactively searching for building blocks to reuse, the two aspects of a project's behavior feed on each other to make proactive reuse work, and work well. There is no need for any project to develop building blocks specifically *for* reuse. Moreover, the reuse options in a new development project are identified in the very early stage of requirements analysis, leading to a clear savings in development costs.

The paper shows also how the organization has adopted a project-family organizational structure as opposed to the more typical product-line organizational structure. This new organizational structure turns out to help the proactive reuse described in the previous paragraph happen.

1.1 Requirements management

Requirements management for computer-based systems (CBSs) includes storing, updating, and linking all requirements artifacts for the purpose of tracing between them and other artifacts. RM is done in order to support requirements change control and impact analysis [63].

When requirements are managed correctly, they facilitate the effective analysis, design, implementation, and testing of the CBS under development. When requirements are not managed correctly, the resulting bad requirements proliferate through multiple branches of the development process, leading to a low-quality CBS [16].

Poor RM is generally considered one of the major causes for product failure [30, 46]. After all, if we do a poor job of understanding our customer's needs, if we do a poor job of deciding the right CBS to build to meet those needs, and if we do a poor job of specifying what we think the customer wants from the CBS, how can we possibly expect a successful project to develop the CBS [15]?

Implementing a formal RM process, such as that mandated for CMMI Maturity Levels 2 and higher [63], for the development of a CBS enables an organization to find defects in the requirements specifications early in the CBS's development lifecycle, to keep the requirements specifications up to date, and to communicate requirements changes to the entire team implementing the CBS.

1.2 Reuse

Reuse is a major component of many software productivity improvement effort, because reuse, when practiced carefully, can result in higher quality software systems at a lower cost [5] and with a shorter delivery time [29]. One way to succeed at reuse is to adopt a product-line approach [6, 11].

The granularity of reuse can vary from reusing a single artifact, such as a component, document, or test case, to reusing a whole product. Reuse can take place during any phase of a CBS's development, including during proposal consideration and marketing analysis,[1] requirements elicitation, requirements analysis, architecture design, code implementation, and testing. Reuse at any lifecycle phase may imply reuse at all subsequent phases. Thus, reusing requirements can be most beneficial, because if it leads to off-the-shelf reuse of the required product, it results in the greatest reduction of development effort and time to market.

1.3 The industrial organization

The studied organization is the Mabat Missiles Division (MMD) and is one division of the Systems, Missiles and Space Group of Israel Aerospace Industries (IAI) [26–28, 72]. MMD "specializes in the design and production of various Missile Systems, Naval Attack Missiles, Loitering Weapon Systems and Precision Guidance Munitions. The Division also offers integrated Naval Combat suites for new ships and as upgrades to existing ships. [MMD] … is also a leader in advanced Air Defense and Anti-Missile systems, both naval and ground based" [26]. Each product

[1] Even though the studied organization develops only bespoke CBSs and does not develop CBSs for the mass market, the department that responds to requests for proposals from external potential customers is called the "Marketing Department". Thus, what this department is doing are marketing and marketing analysis. Consequently, in the organization, a CBS's time to delivery is called the CBS's "time to market". This paper uses this terminology when it talks about time to delivery.

of MMD, developed for potentially a different customer, is an embedded system that controls and senses hardware similar to the way embedded systems that control aircraft, i.e., avionics systems, do. These embedded systems are classified as very large software systems [70], built of dozens of subsystems, some of which are products in their own right and some of which are products supplied by other organizations, within or outside of IAI. A typical such product could have 120 requirement documents, comprising thousands of requirements.

1.4 Research questions

The main research question is:

RQ1: In the development of large-scale systems, does institutionalized, proactive requirements reuse payoff?

This main question leads to three subquestions:

RQ1.1: What are the quantitative and qualitative evidence of this payoff?
RQ1.2: How much does proactive requirements reuse save in the subsequent CBS development?
RQ1.3: What is necessary behaviorally and technologically in an organization for it to be able to institutionalize proactive requirements reuse that payoff?

The rest of this paper is devoted to answering these four research questions.

1.5 Case study

According to Robson's [58] classification of research, the case study of *this* paper is at once

- descriptive—portraying, in Sect. 6, a situation or phenomenon, namely proactive requirements reuse,
- explanatory—seeking and appearing to find in Sects. 6.3, 6.4, and 7, an explanation of a situation or problem, mostly but not necessarily in the form of a causal relationship, namely to see if the proactive requirements reuse accounts for the observed cost reduction,
- qualitative—studying, in Sects. 4–7, a situation in its natural setting using, in Sects. 7 and 8, quantitative and qualitative data,
- holistic—studying, in Sect. 6, a single instance of the phenomenon of interest at a global level for its overall effect, and
- flexible—in which the parameters of the study are defined during the study and can change, as is reported in Sects. 7 and 8.

Because some of the data gathered for the study are qualitative, there is a need for triangulation, i.e., ascertaining that the qualitative data are consistent. If different people are making similar observations about what was happening during the proactive requirements reuse, then the qualitative observations can be said to be triangulated. Moreover, it should be that the qualitative data as a whole triangulate the quantitative data.

For a holistic case study, there are problems generalizing from the study. However, since all the resources of the study are applied to a single instance, and that instance can be explored thoroughly to understand all the variables that come to play in the outcome better than would be possible if more instances were studied. This more thorough study provides a better basis for other studies building on the current study. Also, as Robson [58] observes on Page 139, an *extreme case study* that is studying an instance that had a successful outcome despite all that was mitigating against its success, provides results that can be understood in the sense of "if it can work here it will work anywhere".

Finally, the case study of this paper is an after-the-fact case study. That is, the work had been done with no notion that it would be the subject of research. It was done in the normal course of MMD's conducting its industrial, profit-making business. Only after the work was done, and the dramatic reduction in time to market was noted, did the authors realize that the work should be the subject of a case study. The studied project was chosen after the fact precisely because it had done proactive requirements reuse, that reuse clearly saved a lot of development time, *and* the data to quantify the savings could be collected from the project's RM tool that keeps all project artifacts. Thus, the case study plan given below is an after-the-fact description of the study rather than a true plan laying out work to be done. While the benefits of having a planned case study are gone, there is less chance for the results to be biased by the researchers' desired outcome and there is no chance for a Hawthorne effect.

The after-the-fact plan for this case study would be [58]

Objective: determining if proactive reuse at the requirements level saves resources in the subsequent system development,
The case: studying the system artifacts, the data about time allocated to system development steps, and developer reactions,
Theory: using the prevailing folklore and the research described in Sects. 2 and 3 that show that reuse in general and reuse of requirements saves development,
Research question: asking research questions RQ1, RQ1.1, RQ1.2, RQ1.3,
Methods: analyzing the system artifacts, the data about time allocated to system development steps, and developer reactions, and

Selection strategy: the first author's noticing, after the fact, that the MMD project that she was helping to institute proactive reuse at the requirements level had cost much less than normal and then gathering data to see exactly how much was saved.

1.6 Plan for the rest of the paper

Consequently, Sect. 2 summarizes past work to quantify the effectiveness of requirements reuse to reduce time to market, and Sect. 3 summarizes past work to study RM empirically. Section 4 describes system development at MMD, in particular the organization of its projects. Section 5 describes RM and reuse at MMD. Section 6 describes (1) the MMD project during which the old opportunistic reuse evolved to systematic reuse and requirements-specification-based reuse and (2) the steps taken to achieve the evolution. Section 7 evaluates the effectiveness of the reuse in this MMD project, and Sect. 8 provides triangulation for the evaluation in the form of quotations from interviews of four participants in the studied project. Section 9 considers the threats to the validity of the conclusions from the case study, and Sect. 10 concludes the paper.

2 Past and related work, including empirical, on general and requirements reuse

There has been a lot of work since before the late 1980s on reuse in general software engineering and the closely related concept of product lines, to the point that there are survey reports describing methods for achieving reuse [32, 57], textbooks about setting up reuse programs in practice [11, 29, 54, 55], articles about reuse directed at practitioners [5, 6], a bibliography of literature on reuse in software engineering [40], and a nearly annual conference devoted to reuse in software engineering [25] (and a discontinued bi-annual symposium on the subject [64]).

There have been a number of empirical studies, mostly case studies, of reuse in software engineering exposing the benefits and drawbacks of, the impediments against, lessons learned from doing, and the costs and payoffs of general reuse [19, 23, 36, 37, 42, 49, 54, 59, 62, 65]. These studies show that in fact, there are many software reuse success stories in which the economic benefits are evident and substantial. For example, substantial costs were saved due to implementation of software reuse in the STARS demonstration project [42]. The first CBS that was developed with active reuse cost 43 % of a reference baseline and the second cost only 10 % [42] of the same baseline. Another example is the experience at Hewlett Packard described by Lim [37], where, by applying software reuse, the number of defects in a CBS development project was 15 % less than in past projects, and productivity was 57 % more than in the same past projects. Schach showed that under reasonable assumptions, the cost savings during maintenance as a result of reuse during development are nearly twice the corresponding savings during development. He is able to generalize his results to show that for an arbitrary software system development, these cost savings happen when maintenance accounts for more than 51 % of the total development budget and that the cost savings grow as the maintenance portion of the development budget grows [59]. What has been learned in these studies has been incorporated into a tool-supported method for predicting how much reuse will help [18] that is based on COSYSMO [69] that is in turn based on COCOMO [7].

Nevertheless, reuse is difficult to put into practice. As indicated by the textbooks, reuse must be carefully planned for [11, 29]. There is a certain amount of opportunism in conducting reuse, because reuse cannot happen unless there are artifacts with the correct functionality to reuse [60]. Careful planning is needed to make sure that the reusable artifacts exist and that they are written to *be* reusable [11, 29].

Why is reuse so difficult to achieve? First of all, it may be difficult to find a reusable artifact or pick among possible candidate artifacts [21, 43]. A mismatch between the architectures of the reused code and the using code can make the reuse more difficult and more expensive than just writing a new version of the reused code with the correct architecture [20]. In general, any time reused code is not exactly right for the new purpose, it may cost more to fix the reused code than it does to develop new code [67].

A large subset of the reuse work is concerned specifically with reusing requirements or some of their artifacts [12, 17, 34, 41, 43, 45, 48, 51, 61]. There is some work on requirements reuse from an unexpected quarter. As observed by Meth and Brhel [47], one possible purpose, sometimes stated explicitly, of tools for full or partial automatic requirements elicitation is to allow the elicited requirements to be easily reused [3, 10, 31, 66]. As with general software reuse, requirements reuse is more difficult than it may appear at first sight [43]. From observation of the practice of reuse in software engineering, Fortune et al. [18] conclude that even though requirements reuse is performed only occasionally, it has the largest potential payoff in terms of saved effort.

Requirements reuse suffers the same difficulties that make general reuse difficult. In addition, there may be too many degrees of freedom [14, 44]. One can consider adjusting the assets based on the requirements of the system to be developed. On the other hand, he or she can consider adjusting the requirements of the system to be developed based on the available reusable assets. Essentially, reuse of requirements is a wicked problem [56].

While there is a substantial body of work empirically evaluating general software reuse, there is not a whole lot of work that is evaluating requirements reuse. Daneva [14] describes *what* should be measured in order to determine how effective requirements reuse is in at least the ERP domain. Toval et al. [66] developed SIREN, a tool-supported method, for requirements engineering (RE) based on requirements reuse. In the concept-proving case study of the application of SIREN to an application with security requirements, 75.6 % of the final set of the application's requirements were reused from a requirements repository. Chernak [9] conducted a survey of mostly business analysts in IT organizations around the world. The respondents confirmed, among other things, that the main benefits of requirements reuse are reductions in time to market and development costs, when reusing requirements for releases, the average reported reuse rate was 45 %, with 16 % of the respondents reporting reuse rates of 80–100 %.

The STARS program report [42] made it clear that the organizational challenges of software reuse outweigh the technical ones. Among these organizational challenges is a reluctance of management to risk the used of new, unproved methods for software development. As a result, metrics are needed [5] in order to make "business decisions possible by quantifying and justifying the investment necessary to make reuse happen" [54]. Therefore, a main goal of this paper is to provide data that show how effective the requirements reuse in this project was in reducing time to market and saving money in the project. Thus, this paper adds to the small but growing body of empirical evidence of the value of requirements reuse in reducing time to market.

3 Past and related empirical work about requirements management

While the empirical work concerning requirement reuse may be sparse, there are lots of empirical studies of RM, some of it very specific, but much of it in the context of empirical studies of RE in general.

There are good summaries of empirical studies to validate the effectiveness of RE, its methods, and its tools, including those for RM, in general and in specific projects. These include "An Analysis of Empirical Requirements Engineering Survey Data" by Paech et al. [52] and the Past Related Work section of "Quantifying the Impact of Requirements Definition and Management Process Maturity on Project Outcome in Business Application Development" by Ellis and Berry [16].

Among the work described by Paech et al. are studies of RE methods and tools that comprise RM, i.e., those dealing with requirements volatility, requirements specifications, and reviews and inspections of requirements artifacts. Paech et al. [52, p. 34] conclude from the reviewed work that "It has been established that RE makes a difference for project success".

Ellis and Berry summarize (1) empirical studies in RE, (2) empirical studies in CBS development process improvement and maturity, and (3) empirical studies of RE effectiveness based on frameworks for estimation. The second group of these studies focuses mainly on CBS development process maturity, e.g., Capability Maturity Model (CMM) [53] and CMM Integrated (CMMI) [63]. One of CMMI's key process areas (a.k.a. KPAs) is RM. For a CBS development organization to be considered mature, it must have in place a consistently applied RM process, preferably assisted by tools. One of the points in the third group of studies is empirical evidence of the importance of RM to the projects that use them. For example, Boehm et al.'s [7] cost model weights the components of an RM process heavily in contribution to project cost reduction. Ellis and Berry [16] themselves show that for organizations developing large CBSs, high RM maturity is correlated positively with successful CBS development projects.

Quite a few of the empirical studies specifically concerning RM are focused on proving that particular RM measures are good predictors of stability and low volatility of requirements and reduced change requests [33, 38, 39]. Moser et al. [50] empirically evaluate an ontology-based method and tool for RM and find that the method and tool were more effective in identifying conflicts and in categorizing requirements with less effort than doing so manually.

Almefelt et al. [2] conducted an in-depth case study of RM for the design of a passenger-car cockpit in an organization in the automobile industry. The interviewed members of the organization found RM to be helpful and appreciate the benefits that they draw from it, in spite of its heaviness. Interestingly, the study explored how the organization dealt with what the authors call "commonality requirements", requirements that the cockpit under design shared with other organizations' cockpits, i.e., reuse of common requirements. "Comparing the actual cockpit configuration of the three first cars (three brands) on the platform, it appears that commonality at component level has not been fully adopted in the development project. This might be explained by the fact that the collaborative platform work took place late, i.e. after the actual concept development work. On the other hand, a view supported by information from the interviews is that the actual level of component commonality is well balanced considering the car brands' different driving factors and cost frames." That is, there is not as much requirements reuse as there might have been because (1) they started late and (2) in any case,

the necessity to maintain brand distinctions precludes much reuse.

Damian and Chisan [13] report on the results of an extensive 30-month, three-stage, explanatory case study, using questionnaires, interviews, and document inspection, of the RE process at the Australian Center for Unisys Software. During the case study, the Center was undergoing a concerted RE process improvement following the CMM, which requires RM. They note that the RE process improvement with RM improved the Center's CBS development. In terms of the effect of RM itself, they noted that[2]

- "Feature decomposition, sizing, and change management led to *more accurate estimates*";
- "Upfront test scenario definition, requirements validation, and peer-reviews led to *improved feature coverage*";
- "Enhanced feature understanding, change management, and project tracking led to *managed requirements creep*";
- "Change management and feature sizing led to *effective project scope negotiation*";
- "Traceability links, peer-reviews, and requirements validation led to *fewer defects*"; and
- "Feature decomposition, specification conformance, and team reorganization led to *reduced rework*"

Harter et al. [22] conducted a longitudinal study of the related CBS development projects of one organization that was undergoing CMM-based software process improvement and determined that "higher levels of software process improvement significantly reduce the likelihood of high severity defects". They found that process improvement is more beneficial for large or complex CBSs than for smaller CBSs. However, the benefit is reduced when the requirements for the CBS are ambiguous, unclear, or incomplete. Here again, to the extent that process improvement includes RM, RM contributed to the reported benefits.

This paper adds to the growing body of empirical studies of reuse in software engineering in general, of RM in general, and of requirements reuse in the specific. Of course, no case study, the current one included, is generalizable. However, when multiple case studies tell consistent stories of pay off, the pay off can begin to be believed in general.

4 System development at the organization

As mentioned, MMD is developing large-scale defense systems [26] each of which may incorporate hardware or software products that were produced by earlier projects. Sometimes, a new product is built out of products developed over decades all the while consuming thousands of person-years, or it was upgraded from an earlier product as technology was improved.

A defense system, as a CBS, normally requires 5–10 years of development to deliver in full. During that development time, many experiments are conducted both in laboratories and in the customer's environment in order to test the performance and the compliance with requirements of parts of the CBS already developed.

4.1 Project families

During the last decade, MMD has grouped its projects into families. Each project is developing a product, which is described by the project's product tree that contains as nodes the project's artifacts for the product. A *project family* is an ever-changing collection of completed and current projects whose product trees have a significant number of artifacts in common, enough in common that reuse at some level is feasible. When a project $P1$ in a project family reuses from another project $P2$ in the same family, then $P1$'s product tree shares at least one artifact with $P2$'s product tree. The purpose of this grouping of projects into families is to encourage reuse of artifacts in order to reduce time to market and to save money in the development of new products.

4.2 Organizational project family structure

Fundamentally, MMD's organizational structure is built around project families. Within a project family, there is a great deal of similarity between the products developed by the family's projects and between the customer requirements that drive the products' developments. When a customer comes to request a new product, the first step is to identify the project family whose products meet the most of the new requirements. Thus, any new product is based on other products produced by projects in its identified project family. Thus, reuse is highly encouraged by the project family structure.

As shown by the class diagram in Fig. 1, work in MMD is organized as a collection of *families*, each of which consists of a collection of *projects*, each of which *produces* one *product* that is a *solution* to a *problem* that a *customer* has.

In greater detail, as shown in Fig. 2, each project has a *customer* who wants one of its *problems* solved. So, the customer and MMD agree on a *contract* that directs MMD to form a project to *develop and deliver* a product that is a *solution* to the problem that the customer wants solved and that is *required by* the contract. This project is put in the

[2] In each quotation, the sans-seriffed items (the sans-seriffing not part of the original quotation) are those that are considered parts of RM.

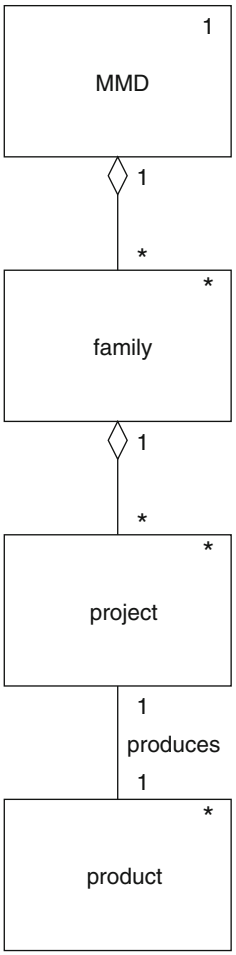

Fig. 1 Project family

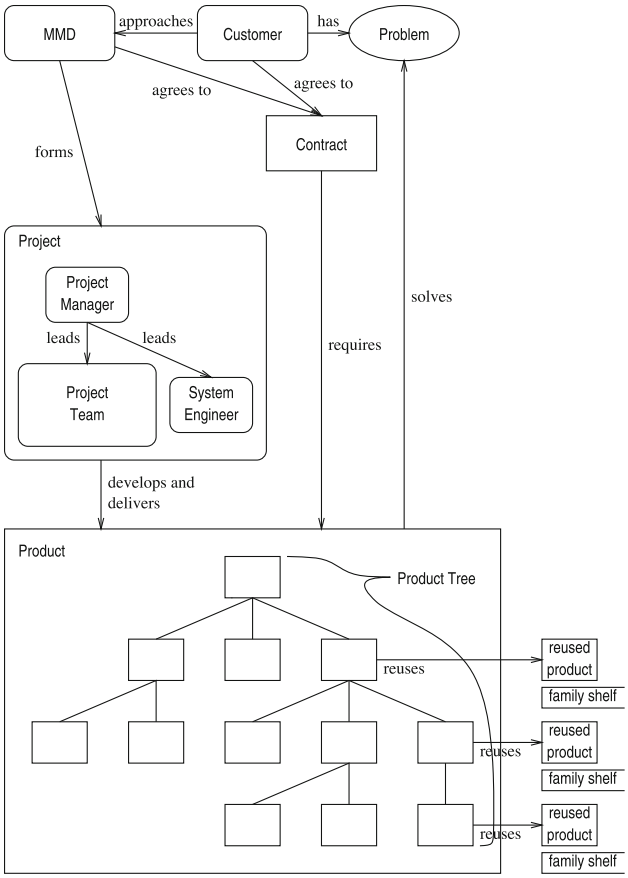

Fig. 2 A project

project family whose products have the most requirements in common with those of the new product to be developed.

Each project has a *project manager* who leads a *project team* in producing the desired product. If the product were to already exist, production could involve *immediate delivery*, but in practice, immediate delivery never happens. If the product does not already exist, then production involves *development*.

If a project develops a product, then the project's *system engineer* will decompose the product into a system of systems, using the DOD standard multi-level architecture framework of system, subsystems, and components. The class diagram in Fig. 3 shows that a product is *implemented by* a system, *S*, that consists of some number of products or of some number of systems that are called *S*'s subsystems. Each of these subsystems is decomposed in the same way. Any instantiation of this class thus is a tree in which the root and each nonterminal node is a system and each terminal node is a product. Thus, at the lowest level, each product is built of other, smaller, products, each of which is a mixture of hardware and software. The diagram in Fig. 2 shows in the product box a *product tree*, which is an object of the class product that is a solution described in Fig. 3. Figure 2 shows that at any level of the product tree, some but not all artifacts in the tree may be reused, and the place from which the reused artifacts come is a shelf of reusable artifacts belonging to the project family in which the project to develop the product resides. Thus, the reused artifacts are said to be *off the family shelf*.

As is shown in the class diagram of Fig. 4, the specification of a system of systems is itself a hierarchy in which

- the product that is a solution is specified by customer requirements specification (CRS) in the form of a contract,
- the highest level system is specified by a system/subsystem specification (SSS),
- each nonhighest level system is specified by a system/subsystem design description (SSDD), and
- each product is specified by a hardware specification (HWS) and a software requirements specification (SRS).

Thus, associated with any product tree is a specification tree, consisting of one SSS, a number of SSDDs, and a number of HWSs and SRSs. Each item in product tree or a

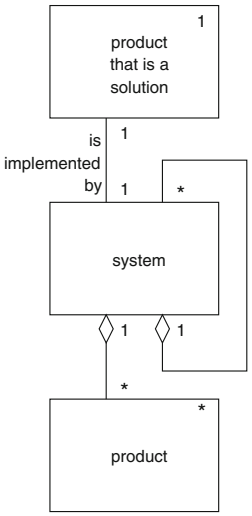

Fig. 3 Structure of a system of systems

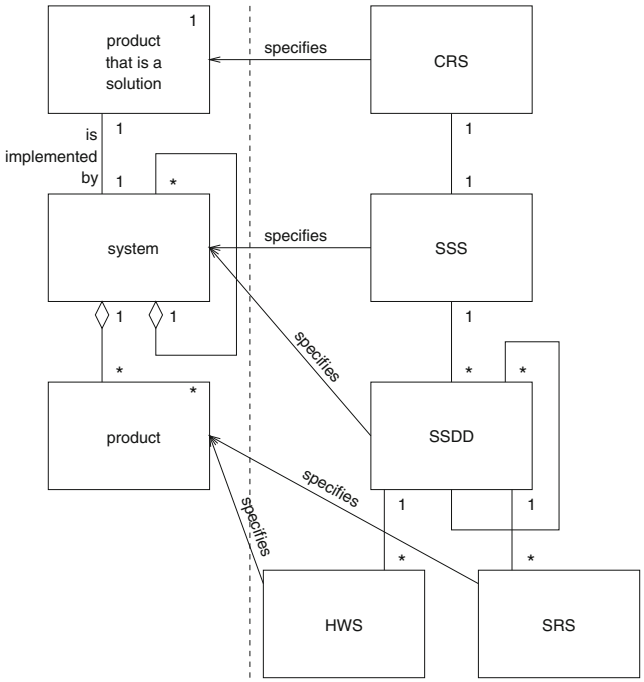

Fig. 4 Structure of requirements specifications of a system of systems

specification tree is what this paper has been calling an *artifact*.

4.3 Project management and system engineering stakeholders

In MMD, a product development project's team are the product's stakeholders. These stakeholders include the project manager, mentioned before; the program manager; the system engineer; software and hardware developers; and testers both in labs and simulations. Many a system engineer, hardware engineer, or software engineer works on the development of multiple projects in the same project family.

A project manager is responsible to review the contract for the project's product and to estimate the project's cost, effort, and schedule. A system engineer is responsible for writing the project's system and subsystems specifications and for keeping track of their implementations. Each software engineer or hardware engineer is responsible for implementing his or her own component's specifications.

5 RM and reuse at the organization

In the past, each development project at MMD maintained its own requirements documents using its own tool, usually only MS Word and in a manner that was potentially different from that of other projects. Reuse at MMD was confined to the lowest, code level, i.e., to using libraries of functions and data structures. MMD hired the first author as a consultant to help them initiate RM and requirements reuse and to do the training necessary so that RM and requirements reuse become the standard practice on MMD development projects. The result is what is described in this paper.

This section describes the context and the infrastructure at MMD supporting RM and reuse and the history of this infrastructure. It shows how the RM process, together with its supporting tool that provides a database in which to store all requirements and development artifacts, enabled requirements reuse.

5.1 Requirements management

Requirements management [35, 71] is particularly important to MMD for a number of business reasons:

- The main goal of all quality improvement efforts in IAI is to improve customer satisfaction.
- Achieving CMMI Maturity Level 5 is a key business goal for MMD. Thus, what is called "RM" in this paper is mandatory in all projects [63].

5.1.1 The beginnings of the requirements management process

When MMD began to consider implementing its RM process, it sought to implement a complete RM process far beyond just producing documents. The RM process would change the way stakeholders talk about CBS development, reviews, workflows, and in fact, would change the whole project culture.

The benefits of such an RM process would be to be able to

- use requirements as the backbone of all CBS developments,
- reduce development and delivery time for all CBSs,
- develop generic CBSs for multiple customers,
- resell existing CBSs to new customers, with a minimum of additional development, and
- reuse requirements of generic building blocks as the basis for reuse among products of a project family.

5.1.2 The RM process needs an RM tool

Motivated by a desire to achieve CMMI Maturity Level 5, MMD decided to deploy an RM tool. At the time, two projects were ongoing with two different customers to provide two large-scale defense CBSs, A and B, with minor requirements differences between them. In addition, MMD was negotiating a contract for developing another large-scale defense CBS, C that was a derivative of A and B but that would be operating in a different operational environment.

Thus, it was clear to one project family's system engineering group at MMD that an RM infrastructure is required and that a way to reuse contract requirements, requirements specifications, and all other requirements artifacts, as well as actual components, is essential for achieving the goal of saving project time, money, and personnel.

The expectations for the RM tool were that the tool would enable

1. easy capturing of requirements into the tool's database,
2. simultaneous work by many employees, including the one system engineering group that would lead all the projects, and
3. *most importantly*, easy reuse of requirements in any project without creating duplicates that would have to be carefully updated together.

The initial plan was to migrate only the contract requirements and the system requirements to the RM tool. However, after a while, it became clear that *all* requirements artifacts needed to be managed in the RM tool. It seems that software developers who had initially opposed the introduction of any RM tool began to appreciate the benefits of RM from the relative stability created by having the contract requirements and the system requirements in the RM tool.

The deployment of the RM process and tool happened in the four phases shown in Fig. 5:

1. The DOORS RM Tool (DRMT) was introduced as MMD's RM tool.
2. The RM process was defined, and the RM tool was customized with the consulting help of the first author.
3. The RM process and the RM tool were used in a new project, and they were found to promote reuse.
4. MMD assigned a Requirements Engineering Manager to be responsible for RM in MMD, as part of MMD's aim to improve its RE processes to be able to achieve CMMI Maturity Level 5.

The history of RM adoption at MMD thus amounted to a simultaneous and evolutionary adoption of an RM tool, an RM process, requirements reuse, and RE as a whole.

5.1.3 RM tool deployment and artifact migration

By 2006, IAI decided to use the DRMT from Telelogic, now IBM [24], as its RM tool. The system engineering group of one project family (the project family called PF in Sect. 6) in MMD had taken the lead and started managing, with the DRMT, project requirements specifications and related artifacts, including CRSs, SSSs, SSDDs, HWSs, SRSs, and algorithm design descriptions (ADDs). Since

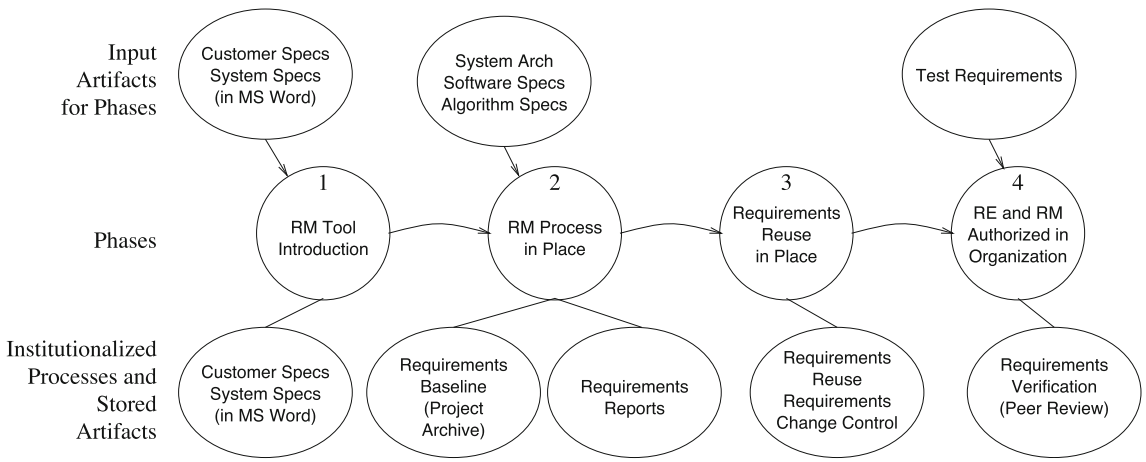

Fig. 5 Phases of the deployment of the RM process and tool

MMD, as part of IAI, was already committed to the MIL-STD-498 [68], specification templates already existed in the MMD installation of Microsoft (MS) Word for all of these artifacts, and all existing artifacts were available as MS Word documents. Equivalents for these templates were defined in the DRMT notation, and the pieces of the original MS Word artifacts were captured into the right places in artifact templates in the DRMT database. Fortunately, the DRMT is able to generate from its database contents all of the artifacts in MS Word format, because these artifacts needed to be provided as MS Word documents as part of the normal project deliverables.

Because of MMD's project family structure and the fact that each MMD employee may work on multiple product projects, the RM tool at MMD has to deal with multiple project families; multiple product projects within a family; multiple systems of systems; multiple sets of artifacts; multiple stakeholders; multiple subcontractors, including external consultants, and other IAI plants; multiple off-the-shelf products; and a matrix organizational structure of employees assigned to projects, in order to develop any fully operational large-scale CBS.

5.1.4 RM levels

A major innovation in the definition of the RM process at MMD was to define *RM levels* based on *the architecture of the CBS under development*. In any CBS that is a system of systems, there exist a number of SSSs and SSDDs, each on a different level of the CBS's architecture. Even the word "system" has to be better identified by its level in the architecture, since often, the development of one subsystem of one CBS is managed as a project on its own, in which what the project develops is referred to as "the system we are developing" even though it is a subsystem of another CBS. Since the artifact associated with a component at any architectural level *n* is a specification of the component's requirements or behavior, the artifact is said to be at RM Level R*n*.

The main RM levels in MMD projects are R1 through R5, as described in Fig. 6.

1. R1 refers to the customer's requirements specification (CRS) containing the contract document and all detailed technical annexes.
2. R2 refers to the system requirements specification in the form of an SSS.
3. Each of R3 and R4 refers to a system or subsystem design in the form of an SSDD or an ADD.
4. R5 refers to software and hardware components' requirements specifications in the forms of SRSs and HWSs.

An R3 system requirements specification that describes a major functionality may have derived functional, performance, and user-interface requirements expressed either as a subsystem design specification or as a collection of algorithm design specifications. Which form is used in any case is determined by the system engineer's doing the derivation according to his knowledge of the structure of the SSDD and ADD templates and of what must be specified. An R4 algorithm design specification or subsystem design specification can be refined at R5 into either a software component specification or a hardware component specification, according to how its implementation will probably be done.

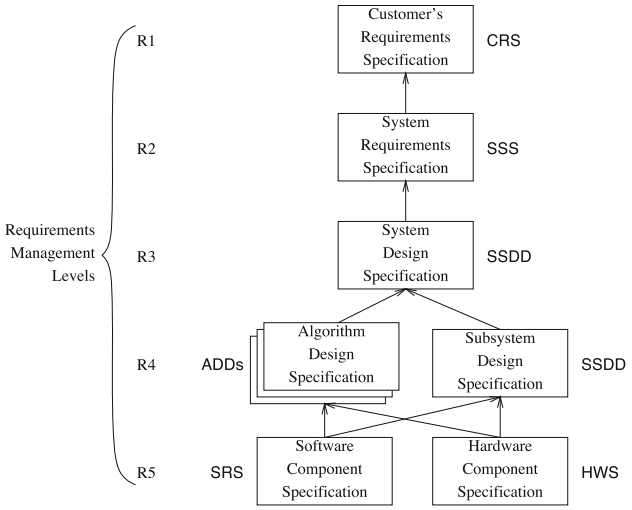

Fig. 6 RM levels

5.1.5 Requirements traceability

Requirements management traceability allows systematic analysis of requirements compliance, discovering inconsistencies between different but related requirements artifacts, and missing requirements. The objective of tracing is to help verify that all requirements in the customer's requirements specification, at RM Level 1, are covered by the other artifacts at RM Levels 2 and beyond, to help find inconsistencies among all these artifacts, and to help ensure that the test cases, developed later cover the requirements in the CRS.

A *trace* is a many-to-many relationship between building blocks at adjacent RM levels. These traces, which are illustrated by the arrows in Fig. 7, form the backbone of the development processes in any project family's projects, starting from customer's requirements specification of type CRS, at RM Level 1; through system specifications of type SSS, at RM Level 2, needed by system engineers for the system design and architecture; through system and subsystem design specifications of type SSDD, at RM Levels 3 and 4; and to software and hardware component specifications of type SRS and HWS, at RM Level 5.

Using advanced RM tools based on the DRMT database enabled MMD to create and manage the traditional product management documentation, i.e., SSSs, SSDDs, and SRSs. In addition, the RM process schema served as a vehicle to produce different project management artifacts that did not exist beforehand such as:

- the difference between projects in terms of their contract requirements and impact analyses,
- a variety of reports generated automatically from metadata offered by the DRMT itself, and
- requirements specifications of experiments to be performed with not-yet-delivered parts of the CBS, the experiments being derived from and traceable to any of a project's artifacts, enabling reuse of software components during the experiments.

5.2 Reuse of requirements

MMD's multi-project RM process was the biggest challenge to deploying the DRMT in MMD. Consider the project family PF depicted in Fig. 7. The project that was the first to use the DRMT, namely the project to develop product A, already had other derivatives for additional customers. It became clear that generic subsystems of A will be reused, which meant reuse also of the subsystems' specifications, but with some changes in the requirements. The only question was "In which level of RM will the reuse occur?"

As shown in Fig. 7, the contract requirements at Level R1 had to be specific for each customer, and they were captured into the DRMT. Each contract manager quickly learned the benefit of accessible requirements and preferred to use a copy of a similar contract and then rewrite it to suit his or her specific customer. This reuse method enabled the generation of a report of the difference between the original and new contracts. This difference report was very valuable in helping the project manager in planning his or her project's schedule and budget in the early stages of the project. When the project manager found requirements in common among two contracts, he or she counted on being able to save time and money by reusing the previously done development that arose from the common requirements.

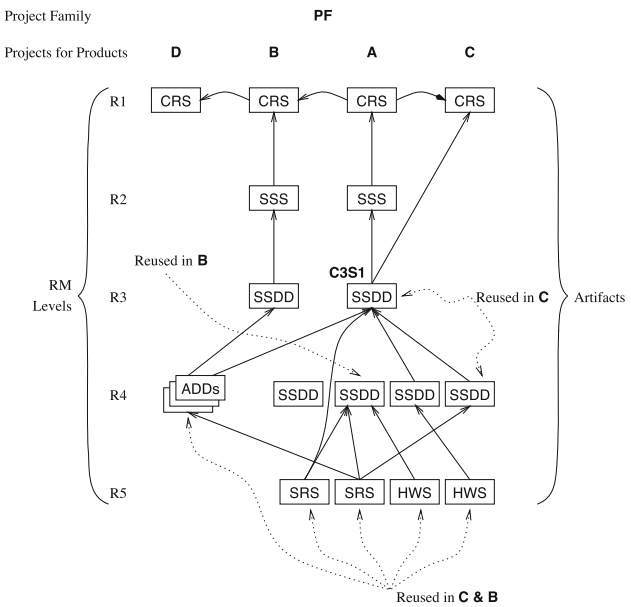

Fig. 7 Reuse within one project family

6 The project that led to systematic reuse

This section describes the MMD project in which the engineers learned to improve their opportunistic reuse to be systematic. The project was in the project family PF, depicted in Fig. 7, that already had two projects going, those to develop the products A and B, when a customer asked MMD to develop a product C. As this project started, the first author, as consultant, saw opportunities for more systematic reuse and took steps to implement that reuse. The section shows how the RM process and the storage of all the project's requirements and development artifacts in the DRMT allowed requirements reuse and how a change in the behavior of the project's senior engineers allowed the requirements reuse to become systematic.

6.1 Start of the project to develop product C

At the beginning of any project to develop a CBS, the project's senior system engineer does an architectural analysis for the CBS to determine its product tree. During the architectural analysis for C, as a result of the increased visibility of the requirements afforded by the DRMT, the senior system engineer realized that there was a terminology difference between components of A and C, as a result of their different problem domains. That A and C are from different domains is reflected in two customer-specific contracts at Level R1, and in the two different system specifications at Level R2, each of which providing a solution in the problem domain of its own customer's contract.

6.2 Opportunistically identified reuse opportunity

In the older product, A, a major subsystem in Level R3 was named "C3S1", while for the newer product, C, the same subsystem was named "C3S2". Fortunately, it was early enough in the development lifecycle of C, and the senior

system engineer for C decided to reuse the name "C3S1" from A, as a basic step toward reuse.

The project to build C had very tight deadlines, and any chance for reuse was appreciated very much. Subsequently, the ADDs of A were reused in a major way, basically because the same system engineers worked in both development projects and proactively searched for reuse opportunities in all component specifications relevant to C3S1. Since all RM artifacts at all requirements levels are requirement specifications, the search involved examining artifacts stored in the DRMT.

6.3 Toward systematic reuse with the help of unique naming of artifacts

One system engineering group normally supports all projects in one project family. This system engineering group is composed of at least the senior system engineers of the projects in the project family. The knowledge shared by the senior system engineers of the projects in a project family is clearly valuable in promoting reuse in the project family. As a result of the opportunistic reuse discovered in the project to build C, the system engineering group of PF decided to promote systematic reuse. This system engineering group observed that any subsystem that is generic enough to be incorporated into more than one product in a family needs to have the same name in all products that incorporate it, in order that it be identified as a generic building block and be reused. Therefore, whenever during any project, any subsystem is recognized as generic, it is renamed to something more generic and reuse promoting. Once a building block is named properly and it is thus announced as generic, the reuse of its requirements is straightforward, and that reuse may lead to reuse of the building block's substructure including its ADDs and their code.

So, the system engineering group of PF asked that its members actively collaborate to ensure that any concept has a unique name within the family. Furthermore, whenever in any project new functionality is identified, the system engineering group's members were to actively search for the highest level existing artifact that provided that identified functionality, even with a bit of tweaking. Finally, the system engineering group members began to systematically identify Level R4 and R5 building blocks that were candidates for being reused and becoming generic building blocks. Thus, reuse in MMD evolved from being opportunistic at the code level to being systematic at the requirements level [65].

6.4 Making a major subsystem generic

Once the system engineering group of PF decided to institute systematic reuse of requirements for projects of the family, it became clear that C3S1 was a major subsystem that could be reused in multiple products in the family. Indeed, it became clear that C3S1 was a system in its own right. The system engineering group implemented a process of trying to reuse all of C3S1's requirements artifacts. Eventually, most of C3S1's components were reused, incorporating requirements evolution while keeping compatibility between projects in the same project family.

Noticeably, in the project to develop C, 50 % of the ADDs of A were reused, enabling 100 % reuse, as-is, of A's SRS, thus allowing requirements evolution while maintaining compatibility between generic products in the same project family.

7 Evaluation of reuse during RM

This section evaluates the effectiveness of the MMD RM's support of requirements reuse for the project to develop C, using data from the DRMT database for the project family PF depicted in Fig. 7. The data for the evaluation are taken directly from the data[3] accumulated by the DRMT about the artifacts under its control. Below, the development time for any artifact is calculated by using the start and end time stamps for any work contributing to the development. The savings from the reuse of any building block is estimated as the building block's *known* past development time, minus the amount of time spent determining whether or not to do the reuse.

7.1 The data and their analysis

An evaluation of the projects in PF at the end of 2007 showed that

- the DRMT was used by the projects for three products in PF; two of the projects, for product A and product B, were in their production phases, and one project, for product C, was before contract signing.
- 30 engineers, from the three projects, used the DRMT for assessing the differences between the products A, B, and C, aiming for reuse,
- each of A and C happened to have eight major subsystems, of which five were very similar, suggesting the possibility of reuse and dictating that C should be in the same project family as A, and
- more than 70 % of the requirements in the CRS for these five similar major subsystems of A and C were identical, making significant reuse a real possibility.

[3] These data are confidential, and their actual values cannot be shown here.

At the end of 2009, another evaluation was performed. By this time, in the hopes of promoting reuse, the convention had been adopted to divide specifications at any level into *DRMT modules*, each about *one and only one* concept[4] of the described product tree. That is, each specification at any level is built out of modules, one for each concept about the product. By restricting each specification module to one concept, any module could be reused verbatim in any product with the same concept. These one-concept models are clearly candidates for becoming building blocks. Checking for a reuse opportunity was reduced to identifying concepts in common between pairs of products. If one of the pair was already implemented and the other of a pair was yet to be implemented, the identified reuse could potentially save all the development derived for the concepts in the new product.

In addition, the DRMT had become a production tool, used by all projects for logging elicited requirements, creating requirements specifications, reviewing requirements changes, generating auxiliary documents, etc., in short, for all RM. This universal use of the DRMT resulted in the availability of types of requirements artifacts that were not available previously, including V&V (verification and validation) matrices and traceability coverage reports, which can be generated automatically from other artifacts in the DRMT.

The first four results of the universal use of the DRMT are about normal usage of the DRMT for RM, and the remaining five are about requirements *re*use with the help of the DRMT.

1. The DRMT was being used by most of the projects in PF, each at its own stage of development.
2. More than 120 modules had been written with the help of the DRMT for the projects in PF and other projects during 2008–2009.
3. 160 engineers were assigned to PF and were using the DRMT,
4. Approximately 30 % of all specification artifacts released during the year were directly generated by the DRMT from the engineer-supplied specifications.
5. The projects for four products, A, B, C, and D, in PF are reusing Level R1 CRS requirements modules.
6. Each of products A and B has a major subsystem, C3S1 at Level R3. The Level R3 specification of A's C3S1 has 1,365 requirements and shares 840 of these with the Level R3 specification of C's C3S1, which has 932 requirements. Thus, about 62 % of the requirements for A's C3S1 are reused by another product in PF, and about 90 % of the requirements C's C3S1 are reused from another product in PF. This reuse is the first identified reuse discussed in the previous section in connection with Fig. 7. These details can be summarized by saying that A's C3S1 shares many requirements with C's C3S1 and that this sharing results in about 62 % of the requirements of A's C3S1 and about 90 % of the requirements of C's C3S1 being reused in PF. This summary, in turn, can be abstracted by saying only that about 62 % of the requirements of A's C3S1 and about 90 % of the requirements of C's C3S1 are reused in PF. In terms of Chernak's reuse rate[5] [9], the reuse rate of A's C3S1 is 62 % and the reuse rate of C's C3S1 is 90 %.
7. About 30 % of the Level R4 ADDs of A's C3S1 and about 55 % of the ADDs of C's C3S1 are reused in PF.
8. One hundred percent of the Level R5 SRSs of C were reused in PF, leading to the wholesale reuse of existing code of software components.
9. On average, about 70 % of each reused Level R5 SRS was reused as-is, i.e., with no change.

The amount of reuse of ADDs is amazing because in general, reuse of ADDs is very challenging since there are always details in one implementation of an algorithm that do not fit in other uses of the same algorithm.

Figure 8 shows a bar graph giving for each requirements level, for each product A, B, and C in project family PF, the reuse rate of the product's artifacts at the requirement level. The reuse rates for A and C are from the data calculated in this paper. The reuse rates for B are only estimated, by calibrating what was remembered about B against the available data for A and C. Note that, while the requirements reuse for C was systematic, the requirements reuse for B was only opportunistic until its system engineer decided late in the project to take advantage of the systematic reuse being done in the project to build C. Thus, B's reuse jumped from the purely opportunistically obtained rate of 50 % in Levels R1 through R4 to 90 % in Level R5.

7.2 Deeper analysis of the data

There has been a marked improvement in MMD's RM process over time. In 2007, projects for only three products were using the DRMT, mainly for assessing requirements differences between contracts. However, by the end of 2009, the projects for eight products were following the RM process, using the DRMT as the operational RM tool. Thus, the number of projects using the DRMT-driven RM process increased 260 % in about 2 years. The number of

[4] The one-concept-per-module calls to mind ontological approaches to RE [8, 50], which are just not used at MMD.

[5] The reuse rate of a requirements specification is the ratio between the number of existing requirements in the artifact reused from previous releases and the total number of requirements in the artifact.

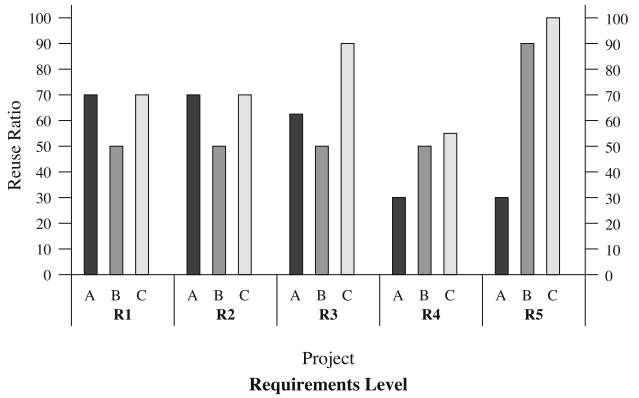

Fig. 8 Artifact reuse rates per product per requirements level

engineers using the DRMT increased over the same period from 30 to 160. This 530 % increase confirms that the DRMT-driven RM process and its infrastructure had become operational in MMD projects. According to the interviews described in Sect. 8, the number of engineers using the DRMT had increased to over 400 by the end of the summer of 2012, for a total increase, from the beginning, of 1,300 %!

By the end of 2009, as is required for CMMI Maturity Level 5, the projects for four products in PF were reusing requirements artifacts. On average, each one of the four reusing projects saved its development effort an amount of work equal to that of 45 % of work spent by the project developing the reused artifacts.

Managing the ADDs at the correct RM level, Level R4, enabled the reuse of more than 50 % of the ADDs of A's C3S1, resulting in 100 % reuse of C3S1's SRSs. Note that, one ADD may be implemented by more than one hardware or software component.

Determining the differences between the CRS for A and the draft CRS for C required about two days. The report of the differences provided information for schedule and budget estimation that was critical for the negotiation of the contract to construct product C.

In MMD, the development to full delivery of the typical CBS requires from 5 to 10 years, of which about 75 % is implementation of software from SRSs. We have found that the software implementation time of a reusing product was about 33 % of the estimated time. We have found also that in the PF project family, in which a typical product requires 5 years for development to full delivery, reuse of requirements from the product A reduced the time to market of each of its reusing products to 60 % of the estimated time to market. These data are consistent. If one adds to the 25 % of the development time that is not implementation time, about 10 % to account for searching for reuse opportunities and then adds 33 % of 75 %; the total is 60 %.

8 Qualitative triangulation

About two-and-a-half years after the period covered in the analysis in Sect. 7, we interviewed four MMD personnel who had been involved with RM and requirements reuse from the beginning, in order to confirm the conclusions of Sect. 7, to see if RM and requirements reuse were still being carried out, to see what the long-term benefits are, and to learn whatever else the interviewees wanted us to learn.

We conducted interviews by telephone in Hebrew with a system engineer specializing in algorithms, a program manager and system engineer, a head of a system engineering group, and an RM manager. In order to preserve promised anonymity while allowing the reader to be able to see which quotations are from the same person, in the lists below, these people are called "*P*1", "*P*2", "*P*3", and "*P*4", although not in the order of the list of their roles in the previous sentence.

The semi-structured interviews were guided by the five questions:

Q1. What do you think about requirements management and requirements reuse now?
Q2. Would you go back to the old way?
Q3. Is reuse worth the trouble?
Q4. How much do you feel that reuse has saved?
Q5. Any complaints?

A first draft translation into English was done by the first author, who is native in Hebrew and speaks English as a second language. The first draft translation was improved by the second author, who is native in English and speaks Hebrew as a second language. The improved translation was polished by the second author's older daughter, who is native in both English and Hebrew. The anonymized original Hebrew quotations and their final English translations can be found at http://se.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/originalQuotations.pdf.

In these quotations, any text enclosed in square brackets is the interviewer's assessment, from the context of what the interviewee is saying, what a pronoun or missing word refers to in the quoted text. Sometimes, without the bracketed text, the quotation might not be intelligible to the reader who does not work at MMD. Each quotation is marked with zero or more "Q*n*"s to indicate which questions the quotation addresses. This marking was achieved by merging with discussion each author's own marking. Some quotations address more than one question, and two address none. Some of the quotations address additional beneficial capabilities (ABCs) that have been discovered for RM and requirements reuse at MMD and that are being exploited. These quotations are marked with "ABC" Finally, note that in these quotations, because the topic of the interview was understood by all, it is clear that

"process" refers to the whole requirements-management-plus-requirements-reuse process.

The text below gives for each interview question all the quotations that address the question. The quotations that address more than one question show up more than once. The quotations for a question are grouped by interviewee.

Q1. What do you think about requirements management and requirements reuse now?

*P*1:

1. The requirements management system is up and running and is working!
2. It's obvious that we need [RM and reuse].
4. They continue to do reuse in [some] family of projects.
5. They now trace the [R4] algorithms to the [R3] system level. [This is what was at levels R4–R5].
6. The algorithms are now traced to shared tests [shared within a family], and they now want shared tests at the system level.
8. People complain loudly that they aren't given access to DOORS in time.
9. In a new project, a new guy came and immediately adjusted to working with the requirements management method on DOORS.
10. The new guy performs reviews on DOORS through discussion and asks for sharable editing.
13. In a huge project, they made a mess at the requirements levels. After the mess maker left, we saw all the problems in the [traceability of the] requirements, but they could fix them because of their distribution at the requirements levels.
14. Now, hardware requirements are entered [to DOORS], and you can trace them, because you can see the levels clearly.
15. The [RM] process document that we wrote was accepted [by senior management].
16. It's possible to summarize that the process is obvious!
17. Anyone who wants to know what's happening in the projects looks at DOORS.
18. Software engineers continue to manage requirements on their own.
19. Today, there are 400 users of DOORS and about 400 modules on DOORS [not 1-1].
20. There are six projects doing full requirements management with DOORS, and all sorts of project initiations too.

*P*2:

1. The requirements management and reuse have spread to all of the IAI and have become the standard at the IAI.
2. We started in the first project with requirements management, and we moved to the second project with reuse and then returned to the first project and improved the requirements.

*P*3:

1. Six years ago, the tool CORE was a pain. Then, I decided that in the project, we needed a tool that focuses on requirements management, and I chose DOORS. Today, in retrospect, this was the best choice I ever made in my entire career, that of the tool and of the way [RM process] to use it.
2. It [the switch to DOORS] wasn't easy, but today, people don't move without DOORS. DOORS serves as the source of the requirements, and all requirements changes must pass through it.
5. In the end, the tool was adopted formally by the entire plant [of IAI].
7. Requirements reuse of the same module happened in several projects, thanks to the attributes. There are 150 requirements in a module, and maybe 15 requirements differ between two projects. Sometimes, this [difference] is only in the parameters.
8. There is also reuse of tests, and that permits flexibility and order.

*P*4:

1. We did a good process that proved itself.
3. We are in the third project with requirements reuse.
5. We do test reuse by linking to the requirements, and that saves. It's not always possible to do test reuse because the conditions [of the tests] change.

Q2. Would you go back to the old way?

*P*1:

3. [Chief System Engineer of the Missiles Division] changed his perception, and now he's reviewing previous projects, and he's asking, "Why not work with DOORS and do requirements management?"
18. Software engineers continue to manage requirements on their own.

*P*2:

2. We started in the first project with requirements management, and we moved to the second project with reuse and then returned to the first project and improved the requirements.

*P*3:

1. Six years ago, the tool CORE was a pain. Then, I decided that in the project, we needed a tool that focuses on requirements management, and I chose DOORS. Today, in retrospect, this was the best choice I ever made in my entire career, that of the tool and of the way [RM process] to use it.

*P*4:

6. We will not go back [to without RM]. There is a struggle to keep [requirements reuse] also on levels R3 and R4, because that's important for system engineering to understand the impact on the higher levels [R1 and R2].

*P*2.2 indicates that RM and requirements reuse helped to improve requirements specification and not just reduce costs. *P*1.3 indicates that senior managers even revisit projects done the old way to enforce the new RM process in order to gain the benefits of the new RM and requirements reuse, and improved requirements specifications and reduced costs.

Q3. Is reuse worth the trouble?

*P*1:

*P*2:

3. Definitely, it [RM] was worth the trouble.

*P*3:

1. Six years ago, the tool CORE was a pain. Then, I decided that in the project, we needed a tool that focuses on requirements management, and I chose DOORS. Today, in retrospect, this was the best choice I ever made in my entire career, that of the tool and of the way [RM process] to use it.
2. It [the switch to DOORS] wasn't easy, but today, people don't move without DOORS. DOORS serves as the source of the requirements, and all requirements changes must pass through it.

*P*4:

3. We are in the third project with requirements reuse.

Q4. How much do you feel that reuse has saved?

*P*1:

11. Those who follow the method with DOORS don't talk about savings; it is obvious.

*P*2:

4. I don't know how to estimate how much we saved [with RM and reuse], but they brought order! Now, we know what are the requirements, what is up to date, what is in the current version, and what was in the previous version.
6. The measurements that I report in system engineering include: the number of TBDs and the number of requirement changes. In my opinion, the number of requirement changes in system engineering is not meaningful to me, because we are developing something new, and there will always be changes until we arrive at the solution.

*P*3:

3. It's hard to say what are the savings. There're savings in the order we now have, that is, no chaos. How can we measure how much we save because of the order and the lack of chaos. It's obvious that there's a substantial savings! All the [requirements] modules are there in one place, and it's possible to freeze them.
4. The tests are linked to the requirements. So I know that all requirements are covered [by tests], or at the time of linking of the tests, we identify requirements that I overlooked, and then, we know what we missed in the tests, and that's worth a lot.
7. Requirements reuse of the same module happened in several projects, thanks to the attributes. There are 150 requirements in a module, and maybe 15 requirements differ between two projects. Sometimes, this [difference] is only in the parameters.
8. There is also reuse of tests, and that permits flexibility and order.

*P*4:

4. The process saved a lot! It's difficult to assess. In fact, without the process, it's like doing the project from scratch. That is, so far, it's possible to say that we have saved three years of work.
5. We do test reuse by linking to the requirements, and that saves. It's not always possible to do test reuse because the conditions [of the tests] change.

These quotations about the difficulty of estimating the savings accrue from requirements reuse. Savings from one project compound into even bigger overall savings in later projects, in which the same requirements are reused again and again. Therefore, the estimate of savings given in Sect. 7 is actually very conservative, as the savings concern only one project in which some artifacts are reused. This estimate takes no account of later reuse of the same artifacts.

Q5. Any complaints?

*P*1:

7. There are complaints of deficiencies [missing requirements] in System SSDD required for traceability of system tests.
8. People complain loudly that they aren't given access to DOORS in time.

*P*2:

5. Requirements change control is heavy.

*P*3:

6. Missing is a modeling tool that's connected to the tests, in order to check for completeness and consistency.

*P*4:

2. Negative Lesson: It did not give us a method to distinguish between the systems in things that are not expressed in the requirements. Things that are not in the scope of the first system, but are required in the second system, we did not succeed to expose. We found defects.
5. We do test reuse by linking to the requirements, and that saves. It's not always possible to do test reuse because the conditions [of the tests] change.
6. We will not go back [to without RM]. There is a struggle to keep [requirements reuse] also on levels R3 and R4, because that's important for system engineering to understand the impact on the higher levels [R1 and R2].
7. The development people are not interested in needing all [requirements] levels.
8. There are change committees for requirements changes. It's difficult to manage this with DOORS.

Whenever people complain that they do not get quick enough access to a tool, as with *P*1.8, or that they need *more* capabilities in a tool, as with *P*2.5, *P*4.2, and *P*4.8 or *more* process, as with *P*4.6, these people are giving a sign that they *actually like* the tools and processes, and they wish that the tools and processes could do more. The complaint about deficiencies in traceability, as with *P*4.6, indicates that the engineers may have actually learned to *like* traceability from what it does at the lower level and now wish that it could be extended to the higher levels.

The first author, who was present at MMD during the case study and knows who was doing what, observes that only *P*2, with Quotation 5, and *P*4, with Quotation 8, complain about the heaviness of the change process with the DRMT. Neither *P*1 nor *P*3 make the same complaint, probably because they interact with the change control board (CCB) frequently and know what the CCB requires. *P*1 and *P*3 know that the tool helps them prepare submissions to the CCB that get approved the first time, because their submissions *show* that they have done their homework to determine the full impact of proposed changes. The DRMT makes that homework considerably more straightforward.

In spite of the complaints, the engineers at MMD nevertheless continue to practice serious RM and requirements reuse, probably because the engineers themselves benefit from the RM and requirements reuse in reduced work overall [4].

ABC. Below are the quotations that point out ABCs that arose as a result of the initiation and routinization of RM and requirements reuse.

*P*1:

6. The algorithms are now traced to shared tests [shared within a family], and they now want shared tests at the system level.
9. In a new project, a new guy came and immediately adjusted to working with the requirements management method on DOORS.
10. The new guy performs reviews on DOORS through discussion and asks for sharable editing.
13. In a huge project, they made a mess at the requirements levels. After the mess maker left, we saw all the problems in the [traceability of the] requirements, but they could fix them because of their distribution at the requirements levels.
14. Now, hardware requirements are entered [to DOORS], and you can trace them, because you can see the levels clearly.
16. It's possible to summarize that the process is obvious!
17. Anyone who wants to know what's happening in the projects looks at DOORS.

*P*2:

*P*3:

4. The tests are linked to the requirements. So I know that all requirements are covered [by tests], or at the time of linking of the tests, we identify requirements that I overlooked, and then, we know what we missed in the tests, and that's worth a lot.

*P*4:

These ABCs include (1) improved traceability, (2) improved testing, (3) improved visibility of artifacts and process, (4) easier recovery from damage caused by an departed employee's bungling, and (5) easier and quicker learning of the ropes by new employees. None of these had been anticipated, and they evidently happened to the pleasant surprise of the personnel.
With regard to the improved testing, writing test cases faces the same difficulties as writing specifications. Each can be written at the wrong level or at mixed levels. The result is that system tests often contain unit tests, and unit tests often contain system tests. The existence of clear requirement levels helps focus tests to the correct requirements levels and thus to the correct testing levels. In addition, the existence of the requirements levels and the tracing links between them helps using traces to determine test coverage. Both a requirement that is not tested and a test case that tests no requirement are easily noticed.

Two quotations were not classifiable, but are nevertheless very telling.

- *P*3: 9. It was great to work with you [Leah Goldin, the former consultant and current interviewer]. You understood quickly our needs and gave us the right answers in the form of an requirements management process. It's not easy to get into people's heads.
- *P*1: 12. What's my next [RE] project?

In the first quotation, *P*3 congratulates the first author for her successful technology transfer effort in getting serious, sustained RM and requirements going at MMD. *P*3 has been in the system development business for a while and *knows* how technology transfer efforts usually go! The second quotation shows the real enthusiasm toward RM that *P*1 had developed as a result of the successful introduction of RM and requirements reuse at MMD.

The interviews were focused on answering the following questions:

Q1. What do you think about requirements management and requirements reuse now?
Q2. Would you go back to the old way?
Q3. Is reuse worth the trouble?
Q4. How much do you feel that reuse has saved?
Q5. Any complaints?

Generally, people at MMD are happy with and continue to use RM and requirements reuse now. They would definitely not go back to the old ways particularly because of the benefits that they perceive RM and requirements reuse give to them. They feel that RM and requirements reuse *are* worth the trouble. While they cannot quantify the savings exactly, they *know* that they *are* saving ever-increasing amounts of effort as reuse compounds with each additional project. They do have complaints, but not enough to want to go back to the old ways. In fact, some of the complaints are in fact signs that people like RM and requirements reuse and that things are working better than they used to.

Thus, it appears that the impressions of *P*1, *P*2, *P*3, and *P*4 confirm the conclusions of the analysis of the case study data, even though they are hard put to quantify the savings. Moreover, the process improvements that began with the case-studied project have been sustained for the two-and-a-half years since the first author finished her consulting with MMD. The process improvements continue to benefit MMD. Nevertheless, there is a clear recognition that some problems remain unsolved and that some other problems may even be revealed by the new processes.

9 Threats

The threats to the validity of the conclusions are the same as in any other case study.

Construct validity: The speaker's intent in any quotation is understood accurately by the analyst and that the data used to analyze the savings in development costs, i.e., code size, money and time spent, and number of personnel involved, are correct for the purpose. Threats against construct validity are mitigated by the conversation of the interview and by general agreement among project managers that the data used are the right data for the purpose. Threats against construct validity are mitigated also by triangulation.
Internal validity: The assumptions of causality are correct, i.e., that the observed savings in development costs is *caused* by the increased reuse. Since reusing an artifact means that its development does not have to be done again, and its development time can be avoided in the reusing project. On the reasonable assumption that each reused artifact is big enough that its development time is more than the time to decide whether and how to reuse it, it is clear that at least some of any observed reduction in total development costs can be attributed to increased reuse. It is possible the mere act of trying a new approach could contribute to the developers' increased enthusiasm and increased productivity. However, the cost savings have been sustained over two years, well beyond when novelty would wear off.
External validity: The collection of interviewed users is representative of all users and what they say represents their behavior and that the studied projects are typical of all projects. This threat is specifically not mitigated, and

the conclusions are claimed to be true *only* for the described projects in the described organization. Each reader will have to determine him or herself whether the context of the study is similar enough to his or her own context that the conclusions seem applicable.

Reliability: The data being analyzed by a different analysts will not give different a set of conclusions. Threats against reliability are mitigated by feedback from the other authors of a previous version of this paper and the new coauthor for the current paper. Threats against reliability are mitigated also by the similarity between the conclusions and the conclusions of past, related work.

10 Conclusions

This paper describes the RM infrastructure at MMD of IAI, an infrastructure that allows MMD to systematically apply requirements reuse to a degree that yields real, measurable savings in development costs and real, measurable reductions in time to market for its products. The main elements of this infrastructure are at once technical, organizational, and behavioral:

1. Each product is developed by a project that resides in a family of projects that are developing related projects, for which reuse will be natural and feasible.
2. There is a single system engineering group for each family, consisting of the senior system engineers for all of the family's projects plus some at-large specialists who work their specialties for all projects in the family.
3. All requirements specification artifacts for one project family are stored in one DRMT for the family.
4. The DRMT for one family organizes the family's requirements specification artifacts into five requirements levels, each of which consists of all requirements specifications at the same level of refinement from (R1) CRSs through (R2) SSSs and two levels (R3, R4) of SSDDs, down to (R5) SRSs and HWSs.
5. The system engineering group of a family proactively looks for reuse opportunities among the requirements specification artifacts in the family's DRMT. When the group's members hear of a project needing requirements at least similar to existing requirements, they use their combined knowledge of the contents of the family's DRMT to find them. They isolate the needed requirements into a module with its own recognizable unique name that becomes standard throughout the family. Sometimes, the module is already there with its name already standardized. Sometimes, the module is already there, but its name is not yet standardized; the name is then standardized, possibly after undergoing a change to make it more easily recognized. Sometimes, the module is a part of an existing module; the part is now recognized and isolated as a building block in its own right.
6. In any potential reuse situation, the system engineering group aims for reusing the highest level artifact possible to gain the maximum amount of reuse and of savings.

By the analysis in Sects. 7 and 8 of quantitative data found in MMD's DRMT, this paper has provided evidence that for at least the paper's studied projects, institutionalized proactive requirements reuse pays off, thus answering positively Research Subquestion RQ1.1 from Sect. 1.4. Futhermore, the analysis of the project data from the DRMT has allowed calculating the amount of savings, thus answering Research Subquestion RQ1.2

Research Subquestion RQ1.3 is harder to answer. Nevertheless, it is clear that *all* of this technical, organizational, and behavioral infrastructure is necessary to achieve effective requirements reuse as part of an effective RM process. The newest concept in this list, and in fact in this paper, is that of the project family. It appears that most people associate product lines or product families with reuse. The second author's experience in her RE consulting business is that thinking in terms of product lines or product families is too limiting, in that it does not expose as many reuse opportunities as thinking in terms of *project* families.

A project can be developed based on one or more product lines, i.e., integrating reused products, assuming the products were à priori defined by the product lines. In a project family, the understanding that a new project belongs to some family is based on a project manager's or a system engineer's intimate knowledge of the project family and the new project, without à priori indication of what can be reused as is. Since each project is budgeted separately, all of a project's effort will be spent for the project's benefit; none of this effort will be spent providing, or even declaring the existence of, artifacts that can be reused by other projects or project families. Nevertheless, when a project makes its requirements artifacts both high quality and visible, these artifacts will be discovered and will be reused by impetus from the *reusing* project. Thus, RM in the context of project families promotes, in a natural way, the visibility of requirements artifacts at all levels of a product tree, from top to bottom. With such visibility, opportunities for requirements reuse arise naturally.

MMD's daily use of its DRMT-driven RM process enables sharing of requirements artifacts among the engineers working on projects to develop products in a project family of large-scale defense systems. This sharing makes

commonality among requirements artifacts readily apparent and exploitable. Moreover, MMD's engineers have been learning to exploit this commonality as opportunities for reuse of all kinds of artifacts. They have learned the benefits of reusing the highest level artifacts, i.e., those with the lowest RM level number, because the use of any high-level artifact implies the reuse of its derived lower-level artifacts. Over the two years reported in this paper, MMD was already reaping the benefits of the improved RM process, of the DRMT infrastructure, and of requirements reuse.

As confirmed by the quotations gathered in interviews two-and-a-half years after the conclusion of the analyzed project, this reaping continues now and is growing daily. Senior managers and senior system engineers are now focusing on full-scale project family deployment, over the breadth and depth of MMD. MMD is now doing RE in the fullest sense of the term, and not just RM. The MMD system development process has achieved CMMI Maturity Level 5. Requirements specifications and RM have become the backbone of the whole development process. Requirements specifications are now tightly coupled with design and testing.

As a result of the RM process and the supporting DRMT, requirements reuse opportunities become clear very early in any development project. Requirements change control is done online, because the infrastructure is there to support these activities. MMD is at the point in which the reuse of requirements specification as an integral part of RM starts to show a positive return on MMD's investment.

In retrospect, it appears that the RM process instituted at MMD *has* achieved, at least partially, all of the original goals listed in Sect. 5.1.1 for introducing an RM process.

- MMD now uses requirements as the backbone of all of its CBS developments;
- MMD has reduced development and delivery time for all CBSs in which requirements reuse has been found to be possible;
- MMD develops generic building blocks that can be used to help assemble CBSs for multiple customers;
- MMD is able resell existing CBSs to new customers, with considerably less than a full development project; and
- MMD does proactively reuse requirements of generic building blocks as the basis for reuse among products of a project family.

Therefore, in conclusion, the main research question,

RQ1: Does institutionalized, proactive requirements reuse pay off?

can be answered positively for at least the studied projects at MMD.

Acknowledgments Many thanks to the anonymous interviewees.

References

1. Goldin L, Matalon-Beck M, Lapid-Maoz J (2010) Reuse of requirements reduces time to market. In: Proceedings of the 2010 IEEE international conference on software science, technology & engineering (SwSTE), pp 55–60
2. Almefelt L, Berglund F, Nilsson P, Malmqvist J (2006) Requirements management in practice: Findings from an empirical study in the automotive industry. Res Eng Des 17(3):113–134
3. Ambriola V, Gervasi V (2006) On the systematic analysis of natural language requirements with CIRCE. Autom Softw Eng 13:107–167
4. Arkley P, Riddle S (2005) Overcoming the traceability benefit problem. In: Proceedings of the 13th IEEE international conference on requirements engineering (RE), pp 385–389
5. Barnes B, Bollinger T (1991) Making reuse cost-effective. IEEE Softw 8(1):13–24
6. Boehm B (1999) Managing software productivity and reuse. Computer 32:111–113
7. Boehm BW, Abts C, Brown AW, Chulani S, Clark BK, Horowitz E, Madachy R, Reifer DJ, Steece B (2000) Software cost estimation with COCOMO II. Prentice Hall, Upper Saddle River, NJ
8. Breitman KK, Leite JCSdP (2003) Ontology as a requirements engineering product. In: Proceedings of the 11th IEEE international conference on requirements engineering (RE), pp 309–319
9. Chernak Y (2012) Requirements reuse: the state of the practice. In: Proceedings of the 2012 IEEE international conference on software science, technology & engineering (SwSTE), pp 46–53
10. Cleland-Huang J, Settimi R, Zou X, Solc P (2007) Automated classification of non-functional requirements. Requir Eng 12(2):103–120
11. Clements P, Northrop LM (2001) Software product lines: practices and patterns. Addison-Wesley, Reading, MA
12. Cybulski JL, Reed K (2000) Requirements classification and reuse: crossing domain boundaries. In: Proceedings of the sixth international conference on software reuse: advances in software reusability (ICSR), pp 190–210
13. Damian D, Chisan J (2006) An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. IEEE Trans Softw Eng 32(7):433–453
14. Daneva M (2000) Reuse measurement in the ERP requirements engineering process. In: Frakes W (ed) Software reuse: advances in software reusability (ICSR), LNCS1844. Springer, Berlin, pp 211–230
15. Davis AM, Yourdon E, Zweig AS (2000) Requirements management made easy. Technical report, Omni-Vista, Inc. http://homepages.laas.fr/kader/Davis_et_al.pdf
16. Ellis K, Berry DM (2013) Quantifying the impact of requirements definition and management process maturity on project outcome in business application development. Requir Eng J 18(3):223–249
17. Finkelstein A (1988) Re-use of formatted requirements specifications. Softw Eng J 3(5):186–197
18. Fortune J, Valerdi R, Boehm BW, Settles FS (2009) Estimating systems engineering reuse. In: Seventh annual conference on systems engineering research (CSER). http://cser.lboro.ac.uk/papers/S01-10.pdf
19. Frakes WB, Succi G (2001) An industrial study of reuse, quality, and productivity. J Syst Softw 57(2):99–106
20. Garlan D, Allen R, Ockerbloom J (1995) Architectural mismatch: Why reuse is so hard. IEEE Softw 12(6):17–26

21. Glass RL (1998) Reuse: what's wrong with this picture? IEEE Softw 15(2):57–59
22. Harter DE, Kemerer CF, Slaughter SA (2012) Does software process improvement reduce the severity of defects? A longitudinal field study. IEEE Trans Softw Eng 38(4):810–827
23. Hoadley CM, Linn MC, Mann LM, Clancy MJ (1996) When, why, and how do novice programmers reuse code? In: Gray WD, Boehm-Davis DA (eds) Proceedings of the sixth workshop on empirical studies of programmers, Ablex Publishing, Norwood, NJ
24. IBM Rational DOORS (2011) Viewed 10 Aug 2011. http://www-01.ibm.com/software/awdtools/doors/
25. ICSR. International conference on software reuse. http://www.informatik.uni-trier.de/ley/db/conf/icsr/index.html
26. Israel Aerospace Industries (2012) MBT missiles division. Viewed 3 Aug 2012. http://www.iai.co.il/15901-en/SystemMissileandSpace_MBTMissiles.aspx
27. Israel Aerospace Industries (2012) MBT space division. Viewed 3 Aug 2012. http://www.iai.co.il/18660-en/SystemMissileandSpace_SpaceDivision.aspx
28. Israel Aerospace Industries (2012) Systems missiles and space group. Viewed 3 Aug 2012. http://www.iai.co.il/25467-en/Groups.aspx
29. Jacobson I, Griss M, Johnsson P (1997) Software reuse, architecture process and organization for business success. Addison Wesley Longman, Harlow
30. Jones C (1995) Patterns of software systems failure and success. International Thompson Computer Press, Boston, MA
31. Kaindl H, Śmiałek M, Nowakowski W (2010) Case-based reuse with partial requirements specifications. In: Proceedings of the 2010 eighteenth IEEE international requirements engineering conference (RE), pp 399–400
32. Kang KC, Cohen SG, Holibaugh R, Perry J, Peterson AS (1992) A reuse-based software development methodology. Technical report CMU/SEI-92-SR-004, Software Engineering Institute, CMU. http://www.sei.cmu.edu/library/abstracts/reports/92sr004.cfm
33. Khraiwesh M, El Sheikh A (2009) Empirical validation of requirements management measures. Int Arab J Inf Technol 6(2):196–203
34. Lam W, McDermid JA, Vickers AJ (1997) Ten steps towards systematic requirements reuse. In: Proceedings of the third IEEE international symposium on requirements engineering (RE), pp 6–15
35. Lauesen S (2002) Software requirements: styles and techniques. Addison-Wesley, Reading, MA
36. Lee N-Y, Litecky CR (1997) An empirical study of software reuse with special attention to ada. IEEE Trans Softw Eng 23(9):537–549
37. Lim WC (1996) Reuse economics: a comparison of seventeen models and directions for future research. In: Proceedings of the fourth international conference on software reuse (ICSR '96), pp 41–51
38. Loconsole A (2004) Empirical studies on requirement management measures. In: Proceedings of the twenty-sixth international conference on software engineering (ICSE), pp 42–44
39. Loconsole A, Börstler J (2005) An industrial case study on requirements volatility measures. In: Proceedings of the twelfth Asia-Pacific software engineering conference (APSEC), pp 249–256
40. LombardHill.com. (2012) World's largest reuse bibliography. Viewed 2 Aug 2012. http://www.lombardhill.com/biblio4.html
41. López Villegas O, Ángel Laguna M (2001) Requirements reuse for software development. In: RE 2001 doctoral symposium. http://giro.infor.uva.es/Publications/2001/LL01/Doc-Workshop.pdf
42. Loral Federal Systems and SWSC/SMX (1996) Software technology for adaptable, reliable systems (STARS), air force/STARS demonstration project experience report, version 3.1, vol I. Technical report, USAF Material Command, Electronics Systems Center
43. Maiden N, Sutcliffe A (1992) Exploiting reusable specifications through analogy. Commun ACM 35(4):55–64
44. Maiden N, Sutcliffe A (1993) People-oriented software reuse: the very thought. In: Proceedings advances in software reuse, selected papers from the second international workshop on software reusability, pp 176–185
45. Maiden NAM, Mistry P, Sutcliffe AG (1995) How people categorise requirements for reuse: a natural approach. In: Second IEEE international symposium on requirements engineering (RE), pp 148–157
46. McConnell S (1996) Rapid development: taming wild software schedules. Microsoft Press, Redmond, WA
47. Meth H, Brhel M (2012) Rise of the machines, the state of the art in automated requirements eliciation. Technical report working paper, enterprise information systems, University of Mannheim
48. Morais CGB, de Oliveira Basilio J, da Silva LF, de Barbalho TR (2010) Cognitio: um processo para reuso de requisitos. *Holo*, 4 (in Portuguese). http://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/view/488/369
49. Morisio M, Ezran M, Tully C (2002) Success and failure factors in software reuse. IEEE Trans Softw Eng 28(4):340–357
50. Moser T, Winkler D, Heindl M, Biffl S (2011) Requirements management with semantic technology: an empirical study on automated requirements categorization and conflict analysis. In: Proceedings of the twenty-third international conference on advanced information systems engineering (CAiSE), pp 3–17
51. na Moros B, Vicente-Chicote C, Toval A (2008) Metamodeling variability to enable requirements reuse. In: Proceedings of the thirteeth international workshop on exploring modeling methods for systems analysis and design (EMMSAD), pp 140–154
52. Paech B, Koenig T, Borner L, Aurum A (2005) An analysis of empirical requirements engineering survey data. In: Engineering and managing software requirements, part 3. Springer, Berlin, pp 427–452
53. Paulk MC, Curtis B, Chrissis MB, Weber CV (1993) Capability maturity model, version 1.1. IEEE Softw 10(4):18–27
54. Poulin JS (1997) Measuring software reuse. Addison-Wesley, Reading, MA
55. Reifer DJ (1997) Practical software reuse. Wiley, New York, NY
56. Rittel H, Webber M (1973) Dilemmas in a general theory of planning. Policy Sci 4:155–169
57. Robertson S (1996) Reuse lifecycle: essentials and implementations. In: Proceedings of the international workshop on systematic reuse
58. Robson C (2011) Real world research, 3rd edn. Wiley–Blackwell, Chicester, West Sussex
59. Schach SR (1994) The economic impact of software reuse on maintenance. J Softw Maint Res Pract 6(4):185–196
60. Sen A (1997) The role of opportunism in the software design reuse process. IEEE Trans Softw Eng 23(7):418–436
61. Shehata MS, Eberlein A, Hoover HJ (2002) Requirements reuse and feature interaction management. In: Proceedings of the fifthteenth international conference on software & systems engineering and their applications (ICSSEA). http://www2.enel.ucalgary.ca/People/eberlein/publications/FI_ICSSEA2002.pdf
62. Sherif K, Vinze A (2003) Barriers to adoption of software reuse: a qualitative study. Inf Manag 41(2):159–175
63. Software Engineering Institute (2010) Capability maturity model integration (CMMI) overview. Viewed 10 Aug 2010. http://www.sei.cmu.edu/cmmi/

64. SSR. Symposium on software reusability. http://dl.acm.org/event.cfm?id=RE121&tab=pubs&CFID=101038456&CFTOKEN=55915038
65. Tomer A, Goldin L, Kuflik T, Kimchi E, Schach SR (2004) Evaluating software reuse alternatives: a model and its application to an industrial case study. IEEE Trans Softw Eng 30:601–612
66. Toval A, Nicolás J, na Moros B, Garcia O (2001) Requirements reuse for improving information systems security: a practitioner's approach. Requir Eng J 6(4):205–219
67. Tracz W (1995) Confessions of a used program salesman: institutionalizing software reuse. Addison-Wesley, Reading, MA
68. US DoD (1994) MIL-STD-498, Military Standard: Software Development and Documentation. http://www.everyspec.com/MIL-STD/MIL-STD+(0300+-+0499)/download.php?spec=MIL-STD-498.025500.pdf
69. Valerdi R, Rieff J, Roedler G, Wheaton M, Wang G (2007) Lessons learned from industrial validation of COSYSMO. In: Proceedings of the seventeenth INCOSE symposium. http://web.mit.edu/rvalerdi/www/Lessons%20learned%20from%20industrial%20validation%20of%20COSYSMO%20Rev5.pdf
70. Weyuker EJ (1999) Evaluation techniques for improving the quality of very large software systems in a cost-effective way. J Syst Softw 47(2–3):97–103
71. Wiegers KE (2003) Software requirements, 2nd edn. Microsoft Press, Redmond, WA
72. Wikipedia (2012) Israel aerospace industries. Viewed 3 Aug 2012. http://en.wikipedia.org/wiki/Israel_Aerospace_Industries