Communications of the Association for Information Systems

Volume 17 Article 36

June 2006

What Can Be Learned from CMMi Failures?

David Gefen
Drexel University, gefend@drexel.edu

Moshe Zviran
Tel Aviv University, zviran@tau.ac.il

Natalie Elman
Tel Aviv University

Communications of the Association for Information Systems

WHAT CAN BE LEARNED FROM CMMI FAILURES?

David Gefen
LeBow College of Business,
Drexel University

Moshe Zviran
Natalie Elman
Faculty of Management,
Tel Aviv University
zviran@tau.ac.il

ABSTRACT

The software crisis has been around since 1968 when NATO first identified the problematic nature of software development. In recognition of the need to manage the process of software development many methodologies have been proposed over the years. A recent contribution to this rich set of rigorous software development methodologies is the Software Engineering Institute's (SEI) Capability Maturity Model Integration (CMMi) methodology. While the minimal previous research on CMMi has extolled it, learning how to implement CMMi successfully requires leaning also from its failures. And yet, despite apparent anecdotes, little is known on this topic possibly because of the reluctance of many companies to wash their dirty linen in public.

Based on a set of in-depth interviews accompanied with survey verification, this study examines the assessed effectiveness and efficiency of CMMi as implemented in several projects in a large high tech company in which only low levels of CMMi maturity were reached. In an exploratory manner this study shows the need to differentiate between the quality of the software product developed through CMMi and the quality of the process involved. The study also shows that whether the product is an off the shelf product or a customized one has a direct bearing of the quality of the product developed under CMMi methodology and the process itself. In particular, we discuss why some projects reach only a low maturity level of CMMi even though the organization as a whole might typically reach high maturity levels.

I. INTRODUCTION

CMMi is part of a general trend toward quality management in software development. Quality management is a domain which is well developed within the manufacturing and the service sectors because meeting customer needs and supplying excellent products can lead to competitive advantage. Within the software quality control sector, however, these quality concepts are still in the emerging stage, with several approaches and no agreement on a single method. Nonetheless, some generic guidelines for achieving quality have been developed and are widely acknowledged, including CMMi and the ISO 9000 group. These standards provide a framework for achieving high quality processes and, presumably through these, also high quality products (Ethiraj, et al., 2005; Gibbs, 1994).

Quality control is necessary because building a new software system is a costly endeavor. In part, this cost is due to the way the software development process is managed or mismanaged. Addressing the obvious need to better manage the software development process in order to reduce development costs and meet business objectives (Zviran, 1990), many methodologies have been proposed since NATO first declared the software crisis in 1968 and proposed the creation of the software engineering discipline as an answer to it (Gibbs, 1994; Niazi, et al., 2005; Pitterman, 2000). A primary contributor to this process of developing rigorous software development methodologies is CMMi (SEI, 2005), developed by SEI. SEI was created in 1984 by the DoD in response to the software crisis and headed by Gibbs himself (Budgen and Tomayko, 2005).

CMMi is claimed to improve the software development process (SEI, 2005). This claim is generally supported by a small number of industry anecdotes (Hoffman, 2004). Anecdotal industry reports also suggest a 20% to 25% reduction in the need to correct software defects after implementation (Anthes, 2004). Among the few companies that have reached level 5, only about 75 worldwide,[1] there is a reported significant decrease in the number of software errors (King, 2003). Correcting software errors is a significant proportion of the overall cost spent on correcting software errors. Estimates put the percentage of cost devoted to errors at between 35% and 50% of the programmers' time (Dunn, 1984) and more than 50% of the total development cost, making it the most expensive activity in software development (Capers, 1986). Software quality is theorized to be directly related to the effort and time devoted to software development (Halstead, 1977). CMMi is also claimed to reduce as much as 60% of the effort in supporting operational systems because of fewer emergency activities (Anthes, 2004).

While these claims intuitively make sense, there is practically no research on companies where CMMi did not achieve these objectives. Understanding this topic remains therefore a needed topic, as highlighted in a recent interview with 25 professionals about CMMi (Niazi, et al., 2005) and is highlighted by the slowness in which the industry is in adopting CMMi.[2]

This paper discusses why some projects do not attain a high level of CMMi maturity, even though the projects in their companies usually do attain a high level of maturity. Understanding why this happens is especially important in the case of outsourcing projects because vendors need to convince prospective clients of their capabilities (Ethiraj, et al., 2005) and their compliance with internationally accepted quality control measures (Pries-Heje, et al., 2005).

This exploratory study steps into this void by analyzing a set of projects in a large high tech company where CMMi was at a high implementation level and yet the technical staff of the particular projects studied were not very pleased with it. In this set of interviews we studied why only low levels of CMMi maturity were reached. The software developers' assessments of how CMMi affected the software development process was examined through interviews and then supported by a survey. It is shown that CMMi effectiveness and efficiency of software development were determined by the contribution CMMi made to the quality of the software product, but surprisingly not by its contribution to the quality of the software development process

[1] SEI publishes the appraisals of participating organizations at http://seir.sei.cmu.edu/pars/pars_list_iframe.asp

[2] The CMM, and later the CMMi, model proposed by SEI has 5 levels of maturity progress. In the first level the process can best be described as chaotic with no planning. In the second level the process is repeatable. In the third it is defined. In the fourth it is managed. And in the fifth it is optimized. When CMM was first proposed 75% of the companies were at level 1 (Gibbs, 1994). By 2004 34.9% of companies were still at level 1, 38.2 at level 2, 18.5 at level 3, 5.5 at level 4, and only 2.9 at level 5 (Evans, 2004).

itself. Whether the product was off the shelf or a customized project also contributed to the perceived quality of the software development process and product.

II. LITERATURE REVIEW

WHAT IS CMMI

CMMi is a process improvement approach that provides organizations with the essential elements of effective processes. It can be used to guide process improvement across a project, a division, or an entire organization. CMMi also provides a point of reference for appraising current processes (SEI, 2005). Adapted to software development, CMMi, and its previous version CMM, deals with a key issue in software development projects, namely an often lacking well-established management framework to handle the software project (Jiang, et al., 2004).

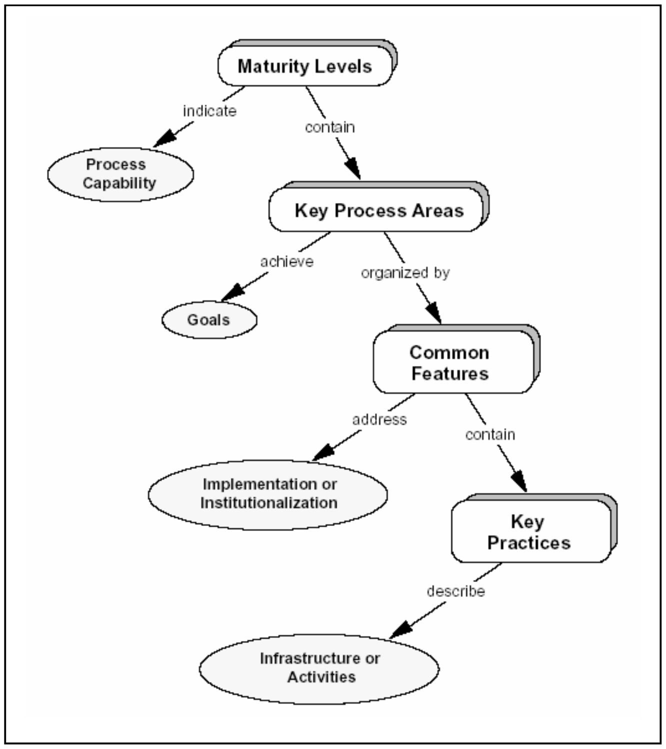
The CMMi model for Software Process Improvement (SPI) has been a significant part of an increasing effort in the last two decades to develop and implement such software development process strategies that can potentially reduce the risk of software development failure and increase the quality of the developed software products. One of the first steps in this direction was the development of the CMMi for Software methodology which provided software organizations with guidance on how to control their software development and maintenance processes (Paulk, et al., 1993). The CMMi methodology is based on four related streams of knowledge which relate to software development: systems engineering, software engineering, integrated product and process development, and supplier sourcing. The CMMi methodology consists of a framework that generates multiple integrated models, training courses, and a method of internal assessment and external evaluation termed "appraisal" (Rassa, et al., 2002).

The approach that has guided the development of the original CMM in the early 1990s was that continued process improvement should be based on small evolutionary steps, rather than on the introduction of revolutionary innovations (Paulk, et al., 1993). As a de-facto standard for process assessment and improvement, CMMi is used to identify key elements of effective processes and to evaluate the maturity of software processes in an organization. Accordingly, CMMi prescribes an evolutionary move toward process improvement and organizes this wide range of activities into a gradual progression through five steps of maturity levels: initial, repeatable, defined, managed, and optimizing (Paulk, et al., 1993). These maturity levels represent an ordinal scale that defines the maturity of an organization's software development process.

At the *initial* level of maturity, the software process is characterized as an ad hoc process which may be chaotic. Few processes are defined at this stage and success depends mostly on individual effort. There is no systematic management of the software development process at this step and the transfer of experience from one project to another is haphazard. At the *repeatable* level, there are rudimentary project management processes in place. Because these methodologies call for a systematic approach to managing software development, the organization is able to repeat previous successes with similar applications. At this level, knowledge transfer and experience are transferable and so the organization can build on the experience gained in previous projects toward making the next software development projects more successful. In organizations that achieve the *defined* level, the software process is standardized and documented, additionally all projects use an approved, tailored version of this process. This more rigid application of methodology allows further knowledge and experience transfer among software development projects by making the knowledge more specific to a particular development methodology. The *managed* level involves, in addition to the steps taken by the defined step, also the collection of detailed measures of the software process and product quality, which in turn are quantitatively controlled. At the highest level of maturity, the *optimizing* level continues process improvement made possible through quantitative feedback. Table 1 describes the maturity levels of CMMi.

Table 1. CMMi Maturity Levels

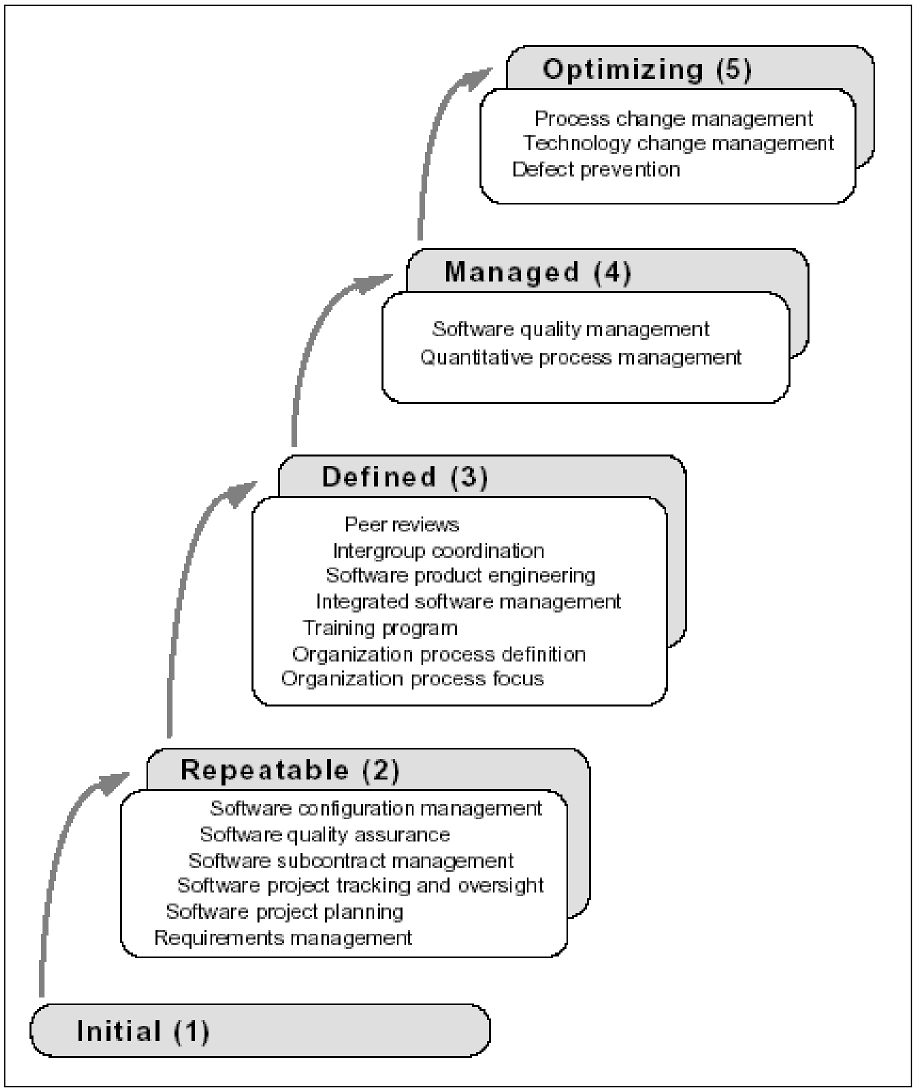
Maturity Level	**Description**
Initial	• Achieving rudimentary predictability of schedules and costs • Establishing process control, process improvement is not yet possible
Repeatable	• Rigorous project management of costs, schedules, commitments, changes, goals
Defined	• Process documented and standardized • Tailored standards for each process • Advanced technology can be usefully introduced
Managed	• Process understood, measured, controlled • Comprehensive process measurements and analysis
Optimizing	• Foundation achieved for optimization of the process • Focus on rapid improvement and rapid technology updating



Adapted from Paulk et al. (1993)

Figure 1. The Structure of CMMi

The hierarchical structure of CMMi is shown in Figure 1. Each maturity level (except for Level 1) can be decomposed into Key Process Areas (KPAs) which identify the areas that should be addressed in order for an organization to achieve a specific maturity level. Figure 2 depicts the KPAs of each maturity level. Next, KPAs can be clustered by their common features, which include (1) commitment to perform, (2) ability to perform, (3) activities performed, (4) measurement and analysis, and (5) verifying implementation. Each common feature contains a set of key practices that describe the policies, procedures, and activities that are important to the effective implementation of a KPA. The key practices describe "what" is to be done rather than "how" the objectives should be achieved. Those practices describe general principles that are relevant for a wide range of organizations, projects, and applications. It is still the responsibility of the implementing organization to formulate specific procedures and activities that best fit its environment and characteristics.


Optimizing (5)
Process change management
Technology change management
Defect prevention
Managed (4)
Software quality management
Quantitative process management
Defined (3)
Peer reviews
Intergroup coordination
Software product engineering
Integrated software management
Training program
Organization process definition
Organization process focus
Repeatable (2)
Software configuration management
Software quality assurance
Software subcontract management
Software project tracking and oversight
Software project planning
Requirements management
Initial (1)

Adapted from Paulk et al. (1993)

Figure 2. Key Process Areas by Maturity Level

Although developed as a standard for U.S. government funded software projects, CMMi has become widely accepted in the commercial sector as well. CMMi is a well established standard for software development control, improvement, and evaluation (Xu, et al., 2003). In the decade since the formal introduction of CMMi, over 6000 companies and government organizations worldwide successfully implemented this methodology (Rassa, et al., 2002). It has become "one of the most popular means for improving software development" (Adler, et al., 2005, p. 215).

According to Adler et al. (2005) over 2000 organizations are using CMMi as of the end of 2003. During 2003 alone, SEI conducted over 450 appraisals in companies to assess their CMMi level.

III. RESEARCH ON CMMI

Most of the research on CMMi, which is surprisingly limited compared to the wide implementation of this model, has focused on the impact of CMMi on project and organizational performance measures. Research shows that this methodology assists in producing high quality software and increased productivity, while reducing the cost involved and the time invested (Niazi, et al., 2005). This conclusion is supported by Adler et al. (2005) who examined a very large software services firm at CMMi level 5. Adler et al. suggested that the potential of CMMi to improve development performance in quality, cost, and timeliness depends on four key success factors: strategic impetus, management commitment, broad participation, and organizational socialization.

In an attempt to empirically examine the effects of the CMMi maturity level on project performance measures, Harter et al. (2000) found that higher levels of process maturity are associated with significantly higher product quality, but also with increased cycle time and development effort. However, the reductions in time and effort resulting from improved quality outweighed the increases from attaining higher levels of process maturity. In another study, Gopal et al. (2002), focusing on key process areas rather than on the maturity level, found that quality-oriented processes significantly reduced rework and increased effort, while having an insignificant impact on elapsed time. Technical processes reduced effort but increased the elapsed time, probably due to the need for increased coordination between project members. Jiang et al. (2004) confirmed that software process management maturity is positively associated with project performance, both for software engineering issues of efficiency and effectiveness and for organizational issues of control, communication, and organizational knowledge. Goldenson and Gibson (2003), in a report prepared for the SEI, collected data from 11 organizations in different locations, sizes, and sectors that implemented CMMi, supporting the proposed ability of CMMi to positively impact performance measures relating to cost, schedule, quality, customer satisfaction, and return on investment.

However, CMMi is not without criticism. The most common criticism is that the higher levels of maturity require excessive documentation, which places a heavy burden on development efforts. Under the standardization and formalism of CMMi, developers may lose their traditional autonomy, which in turn may lead to a loss of motivation and creativity (Adler, et al., 2005). Moreover, the implementation of CMMi may require a considerable amount of time and effort (Jiang, et al., 2004). Herbsleb et al. (1997) found that the cost and time required for a SPI program exceeded the expectations of most of their respondents. However, their results also indicated an association between increased maturity level and improved performance.

IV. BUT WHAT HAPPENS WHEN CMMI IS NOT SO SUCCESSFUL?

What previous research shows then is a picture of a better software development process achieved through CMMi, but at a cost. This bright picture stands in contrast to the relatively low level of CMMi penetration. Moreover, the lack of previous research on relatively unsuccessful implementations of CMMi leaves a gap in understanding how to better manage CMMi implementation. This paper, in an exploratory manner, examines the case of one company that while many of its projects where on a high level of CMMi maturity, others were not. We focused on these low maturity level projects through interviews and corroborated these findings with survey research among the interviewed. Understanding the reasons why in some projects high CMMi maturity is not reached is necessary because the profit margin of many outsourcing vendors depends on their ability to provide high quality software in a standardized manner (Harding, 1993). CMMi provides one methodology for doing so.

To address this gap we obtained access to projects where CMMi was applied, but where the effectiveness of its implementation was of questionable value to the software development team.

These projects took place in a generally high CMMi maturity level company, which gave us access to factors leading to unsuccessful CMMi application on a project-specific basis while being able to exclude organizational polices which obviously favored high level CMMi maturity levels. Moreover, the participants in these projects typically also participated in other projects most of which were high CMMi maturity level projects, so it was not the lack of training of the personnel which led to these results either. The interviewees, 30 in number, were the software project managers in a leading high-tech company in Israel. This company is a large world leader in software development with branches around the world and with this branch in charge of developing high quality complex software. These low level CMMi projects were an exception to the rule in this company.

The technology industry in Israel has been the driving force in Israel's economy, particularly in the last two decades. Israeli high-tech companies have established a significant foundation in the high-tech industries worldwide. As a case in point, Israel ranks among the leading countries globally in the number NASDAQ-listed high-tech companies. Our exploratory study took place in one of the major key players in the Israeli high-tech industry.

The projects were large multi-million dollars projects. The managers were interviewed regarding the respective software projects each was currently managing. The average duration of each interview was two hours. All 30 respondents had an academic degree, 73% were males and 27% females; 3% were 20-30 years old, 24% were 31-40 years old, 36% were 41-50 years old; and 37% were above the age of 50. Having gained this qualitative understanding through exploratory interviews, we augmented it with a survey so we could statistically assess the conclusions. The insight gained from the interviews is discussed next. The quantitative questionnaire and its analysis follow.

In assessing CMMi, the interviewees acknowledged the need for effective implementation of CMMi in more than by name alone if its prescribed benefits were to be derived from it, but also highlighted the need to distinguish between the types of software being developed when assessing this impact. The need to achieve high level CMMi maturity levels was a declared company policy. The interviewees also highlighted the need to distinguish between the quality of the software and the quality of the software development process itself. The effective implementation of CMMi, i.e. the effect it had on the organization, in their view had a direct effect on creating a better software product, as well as on improving the quality of the development process. This is to be expected as it is what CMMi is explicitly designed to achieve. The interviewees also indicated that effective implementation should lead to an increased efficiency and effectiveness of the development and testing phases in the software development process, i.e. meeting budget and schedule targets. This effect was over and above the improvement in the software development process itself. Possibly, this was because of the way CMMi creates a different managerial mindset through its emphasis on quality and feedback loops. However, in contrast to the implied benefits on CMMi (SEI, 2005), they did not think that the improved process of developing the software in itself resulted in improved software quality in these specific projects. Rather, it seems there were types of software development projects where, because of their internal complexity and the associate need to improvise on the run, were better managed in a more flexible development environment than is advocated by CMMi. These development methodologies such as RAD and XP, which were used in these projects and which explain why in this generally high maturity CMMi company these specific projects were relatively low on the CMMi scale, might actually provide an advantage over the more constrictive CMMi model. RAD and XP allow for better handling of contingencies and unexpected events which are typical when handling complex algorithms and poorly defined requirements.[3] The unleashing of productive

[3] XP, or Extreme Programming, is a relatively new software development method which empowers the developers to respond dynamically to customer requests, thereby circumventing a lot of red tape which is prevalent in many other software development control methodologies. XP emphasizes team work with integrated customer and developer teams. It emphasizes courage. More details on XP are available at http://www.extremeprogramming.org/

innovative minds through RAD and XP through the empowerment they give the developers combined with the time and effort saved by not adhering to elaborate control mechanisms are at the core of this phenomenon. In other words, it might be advantageous to apply CMMI in certain types of projects while more advantageous to apply more iterative methodologies in others. CMMi is based on cycles of improvement based on feedback. But what happens then when the project in case is totally different from past organizational experience? In these cases breaking out of the mold by the type of small scale iterative and incremental development activities which characterize RAD and XP may be more advantageous.

The interviewees also made a distinction between the overall efficiency of CMMi in saving time and money during development, and its effectiveness in changing the way the company developed its software. The interviewees were mostly neutral about the efficiency of CMMi. Applying CMMi did not add nor did it detract from meeting budget requirements and deadlines. In contrast, the interviewees were mostly negative about the effectiveness of CMMi in being worth its price through improving how the processes were done in these projects. This would correspond to the observation about CMMi being an effective tool when developing the kind of software the company is used to developing because of its ability to reuse learnt experiences and lessons from previous projects, but being, as in the case of these projects, a liability to some degree when these lessons are not really applicable to the specific type of software developed in the project. This may explain why the projects investigated were mostly at a low level of CMMi and their commitment to CMMi was rather low. Hence, many of the interviewees indicated CMMi as a necessary burden, rather than as a prescribed panacea. Nonetheless, the interviewees acknowledged the improved software quality and the improved quality of the software development process itself brought about through the application of CMMi. This is interesting because CMMi was viewed as a necessary nuisance probably because CMMi was not fully embraced in these projects, but instead was regarded as another external quality requirement forced on the development teams. In sum, it was the necessity of a dynamic iterative development process, which was quite different from what the company usually did, which led to the relatively low level of CMMi maturity in these projects in contrast to most other projects in the company which were on a high CMMi maturity level being able to utilize knowledge acquired from previous projects and programmer experiences.

It is interesting to note that in previous research, where CMMi was mostly regarded as an excellent solution, its application was accompanied with a gradual cultural change in the companies. In contrast, in the interviewed projects it was an external addition which was not accompanied with any cultural change in these specific projects. It was applied because the customer demanded it. The application was not accompanied with any major cultural shift. Programmers and managers need to meet deadlines. If the methodology assists them in doing so, then it will be used. If it hinders them, then it will be used but primarily because of customer requirements. That nonetheless CMMi had the effects it had, despite the unfavorable impression it made on the managers, actually supports previous research, as well as the recommendations of SEI that implementing CMMi be accompanied with an appropriate cultural shift (SEI, 2005) and shows how even in these cases it can have a positive impact on the quality of the software and its development process.

Another topic which came up in these interviews was the need to distinguish among types of software development projects. The interviewees indicated that CMMi was applied differently depending on the whether the software was being written from scratch or being adapted. It was apparently not a fit-all standardized solution. To begin with, software in general is not a uniform product, and, therefore, neither is software development. Perhaps the most telling classification of software is between off the shelf software, on the one hand, and customized software, on the other. The elaborate control mechanisms which are an integral part of CMMi would seen on the face of it to be most appropriate for building large customized software, while quite an excess of controls when it comes to the much less demanding task of implementing off the shelf software where the necessary development is mostly limited to setting parameters in an iterative manner with the customers by and making minor adjustments to an existing and tested software package. This distinction was not identified in previous research on CMMi and yet came up as an important aspect in the interviews. While not really a surprise, it does confirm the conclusion discussed earlier. CMMi, as a methodology, requires a certain cultural attitude of ongoing improvement and learning. When previous knowledge can contribute to the success of the software development project, as was more the case with customized software, its advantages in terms of efficiency are higher. And, when previous knowledge does not contribute appreciably to the success of the software development project, as was more the case with off the shelf software, its advantages in terms of efficiency are lower. Accordingly, the type of the software being developed also affects how efficiently CMMi is being implemented to meet budget and time constraints.

V. QUALITATIVE VERIFICATION

In order to provide some statistical validity to these conclusions, we went back to the interviewees and requested that they complete a questionnaire. The next section discusses this questionnaire and the model it analyses which formalized the conclusions drawn above. Put formally, the conclusions drawn based on the interviews suggested several hypotheses.

- H1: Installation effectiveness of CMMi should improve software quality.
- H2: Installation effectiveness of CMMi should improve the software development process.
- H3: Installation effectiveness of CMMi should improve CMMi efficiency.
- H4: Improved software development process through CMMi should improve software quality.
- H5: Improved software development process through CMMi should improve CMMi efficiency.
- H6: In projects where CMMi is implemented, software quality will be improved more in projects with customized software.
- H7: In projects where CMMi is implemented, the software development process will be improved more in projects with customized software.
- H8: In projects where CMMi is implemented, CMMi efficiency will be improved more in projects with customized software.

In retrospect H1 through H5 can be explained on a rational basis. The criteria for CMMi installation effectiveness was the assessment of the project managers that CMMi was worth its cost and did not hinder other needed activities by needlessly utilizing resources. Recall, in these projects previous knowledge was less applicable than in many other projects where overall the CMMi maturity level was high. This was because CMMi is intended to improve the process of software development and make it more efficient by learning from the past (SEI, 2005).

H6 through H8 can also be viewed as a rational response. If CMMi improves the process of software development and the software product itself, then it should do so more frequently and efficiently in those projects where there is more software development. To be precise in this case, CMMi should have more of an impact in the case of customized software than off the shelf software because there is more development in customized software.

The questionnaire items are shown in Appendix 1. The PPV and PSQ items were on a five point scale anchored at Very Good, Good, No Difference, Bad, and Very Bad. The PEFCT and PEFNC items were on a five point scale anchored at Strongly Agree, Agree, No Difference, Disagree, and Strongly Disagree. The questionnaire was pre-tested before it was administered. The respondents were also asked to state if the project was a custom built IT or an adapted off the shelf one. The mean and standard deviation of each item are also shown in Appendix 1. Item PEFCT3 was reverse coded before the mean and standard deviation of the PEFCT construct were calculated.

VI. QUALITATIVE DATA ANALYSIS

The data were analyzed with PLS Graph Version 3.0 Build 1126. PLS is especially suited for this type of analysis because it can handle small samples and is aimed at exploratory research (Chin, 1998). The factorial validity of the data were established by showing that the correlation among each pair of constructs is smaller than the square root of the Average Variance Extracted, shown in Appendix 2, and by showing the loading of each item on its assigned construct is much higher than on any other construct, Appendix 3 (Gefen and Straub, 2005). The factor validity was revalidated with a Principal Components factor Analysis, Appendix 4. The research model is shown in Figure 3. The path coefficients and explained variance are shown in Table 2. PLS item Loadings are shown in Appendix 1. A single asterisk in Table 2 means significance at the .05 level and double asterisks means significance at the .01 level.

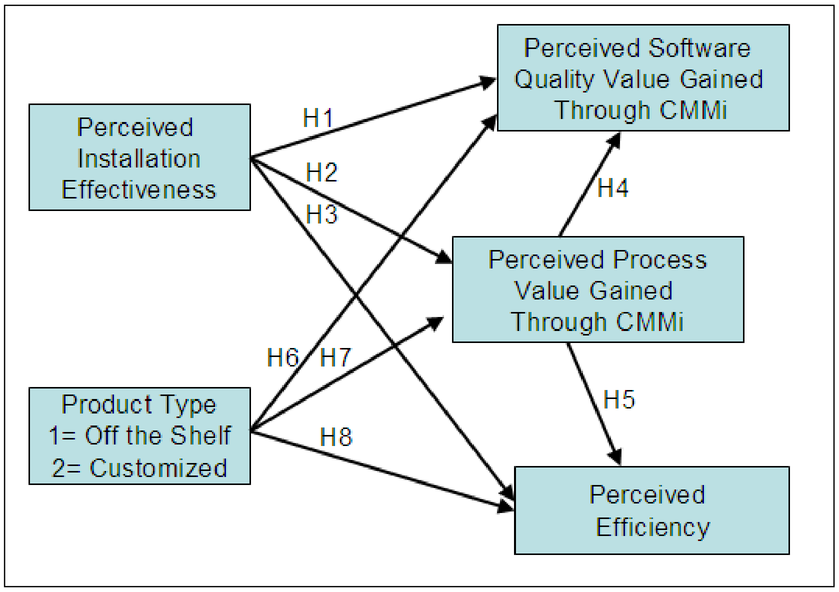

Perceived Installation Effectiveness
Product Type 1= Off the Shelf 2= Customized
Perceived Software Quality Value Gained Through CMMi
Perceived Process Value Gained Through CMMi
Perceived Efficiency
H1
H2
H3
H4
H5
H6
H7
H8

Figure 3: Research Model

Table 2. Path Coefficients and Explained Variance

From / To	Explained Variance	Perceived Installation Effectiveness (negatively worded)	Product Type Customized or Off the Shelf	Perceived Process Value Gained Through CMMi
Perceived Software Quality Value Gained Through CMMi	31%	H1 -.37**	H6 .30*	H4 n.s.
Perceived Process Value Gained Through CMMi	18%	H2 -.30*	H7 .26*	
Perceived Efficiency (negatively worded)	33%	H3 .42**	H8 -.36*	H5 n.s.

n.s. = not significant

The model was also rerun as a saturated model in which all the other paths were included. No other path was significant. Especially noteworthy, Product Type had no effect on CMMi Effectiveness.

VII. DISCUSSION AND CONCLUSION

Software development companies compete for lucrative, job-creating contracts on the basis of industry maturity, labor skills, technology infrastructure, and government support (Pries-Heje, et al., 2005). The acceptance of software process standards and methodologies are key determinants of competitiveness in this market, which is why methodologies such as CMMi are spreading rapidly among software developers worldwide. The benefits of these techniques are well known and provide a credible explanation for why this rapid diffusion is occurring (Ethiraj, et al., 2005; Niazi, et al., 2005). And yet, as Brooks famously put it in his seminal paper (Brooks, 1987), there is no silver bullet in software development. Both the process and the product are complex beyond simple solutions. As previous research has shown, CMMi is in some cases a good methodology for improving the software development process and through it also the quality of the software product itself, although SEI has published case studies on failing projects (Hissam, 1997).

As in many other proposed solutions, CMMi is not a standalone solution. It is part and parcel of a culture of how software should be developed, and as such its impact will depend on the appropriateness of the methodology and the culture it creates to the task. At the core of CMMi is a continuous learning process, presumably assuming that what applied in the past can apply in the present. This is typically true in software development processes and, hence, the quality control and improved practices CMMi brings about are generally beneficial. But, occasionally, as was the case with the 30 projects studied here, this is not the case. Sometimes, the task at hand is either too different from past projects or too simple as not to be worth the overhead that applying lessons and quality control practices learnt in the past entail. This was the case in these 30 projects. Their low maturity level of CMMi in these projects, as opposed to the generally high maturity level in most other projects is indicative, not of failure, but of resourcefulness. Methodologies are only tools. They should not be applied blindly. Sometimes software development is a straightforward process which, by resembling past projects, can benefit from learning from past applications and applying ever increasing quality and quality control accordingly. But not all software development projects are of this nature. Knowing when to apply a methodology and when not to, is a sign of good management. This is what we saw in this company, contrary to what may otherwise have seemed the case.

The interviews suggested, and the data supported, the need to consider the type of software when assessing the impact of CMMi. Methodologies are born within a context. CMMi was created for complex software development processes. But CMMi is no panacea. When it is applied to the simple task of adapting off the shelf software, then its effect was significantly weaker.

LIMITATIONS

Obtaining data on what seems to be failed projects is problematic. Companies typically prefer to keep their dirty linen private. Bearing this in mind, the rather small sample size, while a statistical limitation, is actually quite large considering how practically difficult it is to come by such data. Nonetheless, from a statistical point of view, there is a need to replicate this study with a larger sample to give statistical credence to the conclusions. Having said this, however, the statistics are in place primarily to lend support to the insights gained through the interviews.

Related to this need for additional research, is the need to redefine what is meant by unsuccessful software development projects. On the face of it, these applications of CMMi might have been considered a failure when examined against the CMMi maturity level scale. But when examined considering the applicability of the methodology to the specific tasks, these 30 projects might actually not even be considered failures. Not adhering too closely to CMMi directives, and thus achieving only small benefits from it, may actually have been the right decision to make in these cases.

IMPLICATIONS

Analyses, surveys, and case histories of CMMi implementations reveal some common mistakes made by organizations in planning and executing process improvement programs. Some of these mistakes are due to lack of experience or knowledge; some are the result of cultural "blind spots", a lack of discipline, or optimistically wishing the world didn't work the way it actually does (Hefner and Tauser, 2001). But, sometimes, these methodologies simply do not fit the task and are forced on the software development process by customers.

While any implications drawn on a small sample, whether in interviews or survey data, should be taken with a measure of salt, some conclusions in this exploratory study do stand out and are worthy of additional research and consideration when managing projects with CMMi. First, the findings confirm that CMMi is not a miracle solution, if only because other factors, such as the type of the software being developed, also contribute to the success of both the software development process and the resulting product. Second, nonetheless, installing CMMi successfully is crucial as it directly affects the quality of the software and its development process, at least as it is judged by the project managers who developed it. Third, and this is perhaps somewhat surprising as the other two conclusions could have been derived from previous research and SEI papers, improving the software development process does not seem to automatically have a significant effect on the quality of the software itself. At least in the case of these 30 projects this seems to have been because the application of CMMi, while beneficial in other projects, did not provide enough benefit to justify its cost in terms of additional time and managerial activities.

Also worth noting, product type, as shown in Appendix 2, is highly correlated with the efficacy of CMMi, which is not surprising considering CMMi is designed for complex projects and so, in the case of off the shelf software, it has less of an impact. On the other hand, product type, as shown in Appendix 2, is not significantly correlated with the effectiveness of CMMi installation. This may be the result of the alternative project management methodologies that the project managers had and the way these contributed to the overall successful software development process.

Along the same lines, outsourcing companies should, as many do, require quality control from their vendor. CMMi is a leading methodology in achieving such quality control. But, like other tools, it is important to know when to apply it and when not to apply it. Blindly requiring the vendor to adhere to CMMi when the nature of the software does not justify it, may actually add an unnecessary burden, and throw a wrench in the software development process as evidenced in these projects. Improving software development is not a matter of methodology alone. It is a matter of matching culture with task.

CONCLUSION

CMMi is a good methodology, but it is not a silver bullet. CMMi as described by previous research is an excellent methodology when applied correctly. As this study tentatively shows, even when CMMi is implemented in a less than perfect manner, the quality of this implementation is still of significant impact on several crucial aspects of successful software development projects.

EDITOR'S NOTE: The following reference list contains the address of World Wide Web pages. Readers who can access the Web directly from their computer or are reading the paper on the Web, can gain direct access to these references. Readers are warned, however, that

1. these links existed as of the date of publication but are not guaranteed to be working thereafter.

2. the contents of Web pages may change over time. Where version information is provided in the References, different versions may not contain the information or the conclusions referenced.

3. the authors of the Web pages, not CAIS, are responsible for the accuracy of their content.

4. the author of this article, not CAIS, is responsible for the accuracy of the URL and version information.

REFERENCES

Adler, P.S., F.E. McGarry, W.B. Irion-Talbot, and D.J. Binney. "Enabling Process Discipline: Lessons from the Journey to CMM Level 5," *MIS Quarterly Executive* (4:1), 2005, pp. 215-227.

Anthes, G.H. "Model Mania," *Computerworld* (38:10), 2004, pp. 41-45.

Brooks, F.F. "No Silver Bullet; Essence and Accidents of Software Engineering," *Computer* (20:4), 1987, pp. 10-19.

Budgen, D., and J.E. Tomayko. "The SEI Curriculum Modules and their Influence: Norm Gibbs' Legacy to Software Engineering Education," *Journal of Systems and Software* (75:1-2), 2005, pp. 55-62.

Capers, J. *Programming Productivity*, McGraw Hill Book Company, New York, NY, 1986.

Chin, W.W. "The Partial Least Squares Approach to Structural Equation Modeling," In *Modern Methods for Business Research*, G. A. Marcoulides (ed.) London, 1998, pp. 295-336.

Dunn, R.H. *Software Defect Removal*, McGraw Hill Book Company, New York, NY, 1984.

Ethiraj, S.K., P. Kale, M.S. Krishnan, and J.V. Singh. "Where Do Capabilities Come From and How do They Matter? A Study in the Software Services Industry," *Strategic Management Journal* (26:1), 2005, pp. 25-45.

Evans, M.W. "SPMN Director Identifies 16 Critical Software Practices," www.iceincusa.com/CrossTalk_Mar01.htm, 2004.

Gefen, D., and D.W. Straub. "A Practical Guide to Factorial Validity Using PLS-Graph: Tutorial and Annotated Example," *Communications of the Association for Information Systems* (16:5), 2005, pp. 91-109.

Gibbs, W.W. "Software Chronic Crisis," *Scientific American* (77:September), 1994, pp. 86-95.

Goldenson, D.R., and D.L. Gibson. "Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results," 2003.

Gopal, A., T. Mukhopadhyay, and M.S. Krishnan. "The Role of Software Processes and Communication in Offshore Software Development," *Communication of the ACM* (45:4), 2002, pp. 193-200.

Halstead, M.H. *Elements of Software Science*, Elsevier Scientific Publishing Company, New York, NY, 1977.

Harding, E.U. "IS explores multisourcing: trend toward selective use of third parties - MIS - includes related article on temporary outsourcing partnerships - Field Report," *Software Magazine,* http://www.findarticles.com/p/articles/mi_m0SMG/is_n9_v13/ai_13922287, 1993.

Harter, D.E., M.S. Krishnan, and S.A. Slaughter. "Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development," *Management Science* (46:4), 2000, pp. 451-466.

Hefner, R., and J. Tauser. "Things They Never Taught You in CMM School," *The 26th Annual NASA Goddard Software Engineering Workshop*, 2001, pp. 91-95.

Herbsleb, J., D. Zubrow, D. Goldenson, W. Hayes, and M. Paulk. "Software Quality and the Capability Maturity Model," *Communications of the ACM* (40:6), 1997, pp. 30-40.

Hissam, S. "Case Study: Correcting System Failure in a COTS Information System," *SEI COTS, http://www.sei.cmu.edu/cbs/monographs/case-study-correcting/case.study.correcting.pdf*, 1997, pp. 1-14.

Hoffman, T. "Checking Out CMM," *Computerworld, http://www.computerworld.com/managementtopics/management/project/story/0,10801,95457,00.html*, 2004.

Jiang, J.J., G. Klein, H-G. Hwang, J. Huang, and S.-Y. Hung. "An Exploration of the Relationship between Software Development Process Maturity and Project Performance," *Information & Management* (41:3), 2004, pp. 279-288.

King, J. "The Pros & Cons of CMM, http://www.computerworld.com/managementtopics/management/project/story/0,10801,87882,00.html," *Computerworld*, 2003, pp. 50.

Niazi, M., D. Wilson, and D. Zowghi. "A Maturity Model for the Implementation of Software Process Improvement: An Empirical Study," *The Journal of Systems and Software* (74), 2005, pp. 155–172.

Paulk, M.C., B. Curtis, M.B. Chrissis, and C. V. Weber. "Capability Maturity Model for Software, Version 1.1. CMU/SEI-93-TR-24," 1993.

Pitterman, B. "Telcordia Technologies: The Journey to High Maturity," *IEEE Software* (17:July–August), 2000, pp. 89–96.

Pries-Heje, J., R. Baskerville, and G. Hansen. "Strategy Models for Enabling Offshore Outsourcing: Russian Short-cycle-time Software Development," *Information Technology for Development* (11:1), 2005, pp. 5-30.

Rassa, R.C., V. Garber, and D. Etter. "Capability Maturity Model Integration (CMMI): A View from the Sponsors," *Systems Engineering* (5:1), 2002, pp. 3-6.

SEI "Welcome to the CMMI® Web Site," *CMMi,* http://www.sei.cmu.edu/cmmi/, 2005.

Xu, Y., Z. Lin, and W. Foster. "Agile Methodology in CMM Framework: An Approach to Success for Software Companies in China," *Proceedings of the GITM*, Calgary, Canada, 2003.

Zviran, M. "Relationships between Organizational and Information Systems Objectives: Some Empirical Evidence," *Journal of Management Information Systems* (7:1), 1990, pp. 65-84.

APPENDIX I. QUESTIONNAIRE ITEMS

Code	Item	Mean (Std)	PLS Item Loadings	PLS Composite Reliability
	Please state how the value was on each of the following characteristics after applying the CMMi?			
Perceived Process Value		**3.67 (.64)**		**.95**
PPV1	Meeting project timetable	3.83 (.81)	0.980	
PPV2	Meeting project budget constraints	3.52 (.63)	0.930	
Perceived Software Quality		**3.62 (.52)**		**.92**
PSQ1	Functionality	3.73 (.58)	0.887	
PSQ2	Reliability	3.63 (.56)	0.932	
PSQ3	Maintainability	3.50 (.63)	0.845	
	How would you characterize the progress of process improvement since the adoption of CMMi?			
Perceived Installation Effectiveness (negatively worded)		**2.10 (.80)**		**.88**
PEFCT1	The CMMi was well worth the money and effort we spent, it had a major positive effect on the organization.	3.97 (.77)	-0.840	
PEFCT2	Because of the CMMi we have neglected other important issues facing the organization.	2.37 (1.129)	0.780	
PEFCT3	Nothing much has changed since the CMMi adoption.	1.90 (.96)	0.887	
Perceived Efficiency (negatively worded)		**3.10 (1.12)**		**.93**
PEFNC1	Process Improvement is taking longer than we expected.	3.23 (1.19)	0.965	
PEFNC2	Process Improvement is costing more than we expected.	2.97 (1.13)	0.958	

APPENDIX II. CORRELATION WITH THE SQUARE ROOT OF THE AVERAGE VARIANCE EXTRACTED IN THE DIAGONAL

	PEFCT	PEFNC	ProductType	PQS	PPV
PEFCT	0.842				
PEFNC	0.441	0.962			
ProductType	-0.073	-0.394	1.000		
PQS	-0.403	-0.554	0.407	0.889	
PPV	-0.332	-0.276	0.291	0.274	0.955

APPENDIX III. PLS CONFIRMATORY FACTOR ANALYSIS

	PEFCT	**PEFNC**	**Customized or Off the Shelf Software**	**PQS**	**PPV**
PEFCT1	**-0.84**	-0.48	0.31	0.42	0.21
PEFCT2	**0.80**	0.33	0.21	-0.20	-0.39
PEFCT3	**0.89**	0.30	-0.04	-0.37	-0.27
PEFNC1	0.46	**0.97**	-0.37	-0.56	-0.32
PEFNC2	0.38	**0.96**	-0.39	-0.51	-0.20
Product Type	-0.07	-0.39	**1.00**	0.41	0.29
PQS1	-0.35	-0.54	0.46	**0.89**	0.40
PQS2	-0.32	-0.44	0.43	**0.93**	0.19
PQS3	-0.41	-0.49	0.16	**0.84**	0.11
PPV1	-0.45	-0.41	0.31	0.54	**0.96**
PPV2	-0.25	-0.16	0.03	0.14	**0.81**

APPENDIX IV. PRINCIPAL COMPONENTS FACTOR ANALYSIS WITH OBLIMIN ROTATION

	Component				
	1	2	3	4	Communalities
PQS2	.986	.046	.089	.058	.945
PQS1	.852	.026	-.075	-.155	.825
PQS3	.714	-.163	-.012	-.110	.700
PEFCT2	.201	.902	.011	.092	.788
PEFCT3	-.173	.864	-.089	-.182	.814
PEFCT1	.131	-.681	-.027	-.185	.669
PPV2	-.141	.017	.976	-.031	.906
PPV1	.337	-.091	.704	-.055	.823
PEFNC2	-.011	-.015	-.042	.947	.912
PEFNC1	-.083	.049	-.026	.898	.929
Eigenvalue	3.517	2.949	2.027	3.028	

ABOUT THE AUTHORS

David Gefen is Associate Professor of MIS at Drexel University, where he teaches Strategic Management of IT, Database, and VB.NET. He received his Ph.D. from GSU and a Masters from Tel-Aviv University. His research focuses on psychological and rational processes in ERP, CMC, and e-commerce implementation. David's interests stem from 12 years developing and managing large IT. His research findings have been published in MISQ, ISR, IEEE TEM, JMIS, JSIS, DATABASE, Omega, JAIS, CAIS, and JEUC.

Moshe Zviran is Associate Professor of Information Systems in the Faculty of Management, The Leon Recanati Graduate School of Business Administration, Tel Aviv University. He received his B.Sc. degree in mathematics and computer science and the M.Sc. and Ph.D. degrees in Information Systems from Tel Aviv University, Israel, in 1979, 1982, and 1988. He held academic positions at the Claremont Graduate University, California, the Naval Postgraduate School, California, and Ben-Gurion University, Israel. His research interests focus on the management of the information resource and information systems security. Prof. Zviran's research has been published in: MIS Quarterly, Communications of the ACM, Journal of Management Information Systems, Communications of the AIS, IEEE Transactions on Engineering Management, Information and Management, Omega, The Computer Journal and other journals. He is also co-author (with N. Ahituv and S. Neumann) of *Information Systems for Management* (Tel-Aviv, Dyonon, 1996) and *Information Systems – from Theory to Practice* (Tel-Aviv, Dyonon, 2001).

Natalie Elman was a M.Sc. student at the Information Systems Department, Faculty of Management, Tel-Aviv University.

.