NURTURING THE ACORN: HELPING A SMALL SOFTWARE COMPANY ONTO THE CMM LADDER

DAVID S. BOWERS

Department of Computing, University of Surrey, Guildford, Surrey GU2 7XH

d.bowers@eim.surrey.ac.uk

+44 (0)1483 879635

fax: +44 (0)1483 8796051

Keywords:

CMM, Software Process Improvement

Stream:

Practice

Topic:

Project Management and Configuration Management?

ABSTRACT

We report on an interaction between a University and a small software development company within the framework of a Teaching Company Scheme. By exploiting the peculiar environment offered by a TCS, the University was able to help the company introduce measures to improve their software development process. Not only have these measures moved the company from level 1 to level 2 of the Capability Maturity Model; they are doubtless also responsible, at least in part, for the company's survival. The fundamental features of the environment which supported this success are discussed, and it is suggested how the approach might be applied elsewhere, either within or independently of a funding framework such as TCS.

1. INTRODUCTION

The Capability Maturity Model (CMM) [1] has become well known as a means of driving Software Process Improvement (SPI). Although derived initially in the context of large-scale aerospace control software projects, it is equally relevant to a wide range of software development environments [2]. Most successes reported in the literature, however, describe the application of CMM in large organisations: see, for example, [3], [4]. CMM is not a "quick fix", but all the reports affirm that the large improvements in software quality justify the cost[5].

In surveys of a wide range of organizations, even the smaller enterprises that respond are hardly "small" in any absolute sense; in [6] it is noted that, "25% [of the organizations] had fewer than 54 software engineers". That same study noted common criticisms of CMM, including that, "CMM-based SPI will be counterproductive, will cause the organization to neglect important non-CMM issues, and will cause the organization to become rigid and bureacratic, making it more difficult to find creative solutions to technical problems." A related criticism cited is that, "CMM-based SPI causes organizations to become risk-averse." The authors note that, "there is some limited evidence that it may be more difficult to apply the CMM … in small organizations".

The last two points are particularly pertinent for VERY small software companies. Such companies can be built on creative, innovative developments, which are, by their very nature, high risk. Sometimes, the innovations are the "good ideas" of an individual, on which the company has developed a niche market. The problem for such companies can be not, how to progress up the CMM ladder, but rather, how to get their foot onto the first rung without compromising the balance between creativity and discipline[7].

We report in this paper a Teaching Company Scheme between the University and one such (very) small enterprise. During the Scheme, the development section of the company grew from, essentially, a single programmer with a couple of assistants to a managed team of some 9 developers. This transformation depended heavily on the introduction of a number of standard techniques corresponding to level 2 of the CMM; without these techniques, the survival of the company – let alone its growth – might have been rather uncertain.

As presented, our experiences constitute a single longitudinal case study which revealed deficiencies in the software processes of the company [8], and suggested appropriate intervention to meet the long-term goals of the company [9]. Participant observation [10] was combined with consultancy to gain understanding of the development environment [11]. Using an interpretivist approach, we have been able both to draw specific conclusions concerning the need for, and methods for introduction of, techniques to support the repeatability criteria of CMM, and also to abstract some facets of the teaching company environment which may prove valuable in other organisations. [12].

Thus, the most important message of this paper is that the framework of a Teaching Company Scheme provides an excellent mechanism for software process improvement. This approach could even be generalized to situations where grant support is not available.

This project was first reported, somewhat superficially, in [13].

2. THE PROBLEM

Background

When the company approached the University, they were about to launch a second major software product. Both products had been developed by a talented programmer, their principal developer. There were two other developers in the "team", but both were assistants to the principal developer rather than independent team members. Management of the development process was one of several responsibilities of the Product Manager.

It was anticipated that the new product would satisfy an emerging niche market. The range of facilities required of the product was large, and the company (or, rather, the Directors) realised that the task was "far too large" for their principal developer alone. Hence, the initial goals for the Teaching Company Scheme proposal were for two major areas of the product to be developed by Teaching Company Associates (TCAs).

The Teaching Company Scheme, as funded, was for two Associates, each of whom was to investigate and develop, during their two-year appointments, distinct components of the new product. A tacit requirement was for the development process within the company to be improved, so that the Associates could, indeed, develop their components independently of the principal developer. In reality, the most significant deliverable from the Scheme was the paradigm shift required to allow the transformation of the development section from a principal developer with assistants to a coordinated development Team.

It is this paradigm shift, which was associated primarily with the first Associate, which forms the focus of this paper.

The Starting Point

When the first Associate joined the company, the development process was dominated by a perceived requirement to synchronise releases with two software exhibitions, which occurred in the Spring and Autumn. The year was split into two cycles, each comprising three months of innovation, followed by three months of preparations for the next release. The latter of these phases was characterised by the "individual efforts and heroics" attributed to immature (level 1) software organizations [6]. The inclusion of features in successive releases tended to be whimsical, rather than planned, and fluid right up until the release was actually burned onto a CD. Testing was as thorough as possible, but not as disciplined as it might have been. Further, the process, which depended critically on the principal developer, was interrupt-driven, with fixing of bugs frequently taking priority over development. Documentation was limited, and bug fixes could introduce further bugs.

There seemed to be a complete lack of a formal change control process, and the selection of which bugs were to be fixed (first) appeared to be determined primarily by which customer shouted loudest.

The company, including the principal developer, was aware of the problems, but, like so many in their position, could not see how to address them: they were just too busy trying to keep their heads above water.

As a corollary, there was "no time" for the new Associate to be familiarised systematically with the existing product. With little technical documentation, he was left to explore the raw code, and to seek help – from the principal developer – when needed, adding to the interruptions for the latter. The Associate was asked to enhance the product, but his modifications tended to have unexpected side-effects. It was always "quicker" for the principal developer to sort problems out himself, rather that "waste" time explaining how it should be done.

The problems will be familiar to anyone who has observed the early growth of a software team; many such problems are described in Fred Brooks's classic [14]. The existence of the problems is no reflection on anyone; they are almost inevitable. Indeed, the very existence of "Level 1" in CMM implies that they are widespread. However, for the company to grow – or even survive – it was essential that they should be overcome.

CMM levels 1 and 2.

Of the five levels of the Capability Maturity Model, only the lowest two are relevant to this study.

Level 1, the "Initial" level, of the Capability Maturity Model is described thus[1]:

> *"The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics."*

and level 2, the "Repeatable" level, as:

> *"Basic project management processes are established to track cost, schedule and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications."*

Each of the 5 CMM levels – except the first – is split into a number of "key process areas" that indicate the focus required for a company to improve its software process. For level 2, the key areas are:

Requirements management
Software project planning
Software project tracking and oversight
Software subcontractor management
Software quality assurance
Software configuration management

For an organization to claim to be at level 2 – almost by definition, the lowest level of maturity in which even a small ***team*** can operate successfully – the goals specified for each of these areas must be achieved.

The company, as described above, was clearly failing to meet any such criteria. This does not mean that their products were necessarily bad, but, rather, that the process by which they were developed could have been better. The aim of CMM is to suggest how to improve the process, so that subsequent developments will be more effective and require fewer "heroics" than currently. A consequent improvement in quality should also be observed.

Applying a mechanism such as CMM is not zero-cost. In [15], it is noted that, "the successful implementation of the CMM guidelines is often not accomplished without significant organizational change", and in [16] it is suggested that the required investment will be forthcoming only if it is justified by the business goals of the organization. In the case of our example company, survival would seem a fairly dramatic business goal!

Further, CMM is a framework rather than a strict recipe. It is intended to benefit the enterprise, rather than be yet another regulatory burden. It is suggested in [2] that strict adherence to the levels is less important than attaining the benefits which CMM can bring, using the components of CMM to address perceived problems.

CMM is not the only framework for software engineering: ISO 9001 is a well-known alternative. Both share a concern for quality and process management, but they are not equivalent. The two models have been compared in studies such as [17]. In [18], it is noted that, "although an ISO-9001 compliant organization would not necessarily satisfy all of the level 2 key process areas, it would satisfy most of the level 2 goals and many of the level 3 goals." Given that CMM has 5 levels in all, it would seem that ISO-9001 may be a minimum, rather than ideal, framework, in addition to suffering from a "compliance" philosophy rather than one focussed on benefit.

It should be noted, incidentally, that the company had recently acquired ISO 9000 registration for its sales division, but had decided that the bureaucracy involved in seeking the corresponding recognition for its development activities was likely to outweigh any potential benefits.

3. THE APPROACH

A qualitative approach is appropriate where, as in this case, the primary concern is with processes [10]. Furthermore, where there is a clear need to change the organisational processes, Action Research is particularly relevant [19], and participant observation may well be the most effective way to learn just how much needs to be improved[10], and for developing the appropriate quality strategies [20]. Such an approach inevitably introduces biases [12, 21], but such effects are more usefully regarded as beneficial results of critical research [22]. These direct results are validated by the extent to which they are accepted by the company involved [10].

Data presented in this paper were derived from a combination of formal reports made by the Teaching Company Associate to the Company, direct observations by the Author, and discussions between the Associate and the author.

Options and Constraints

To recap the problem, the company was totally dependent on one individual, who had become interrupt-driven and, hence, less productive. Because bugs were fixed "on the fly", side-effects were introduced, and this led inevitably to significant reworking. As the demands grew, it became less and less feasible to develop the team which was clearly needed. The company was very much at level 1 of the CMM. Although the problems had been recognised, the management were unable – because of those very problems – to do much about them. Such problems are not restricted to software processes; see, for example, [23]

In a situation such as this, there are basically three possible strategies:

- to impose a "proper" project management framework, and force everyone to conform;
- to encourage evolution, or
- to persuade the proverbial leopard(s) within the company to change their spots.

The first could well have resulted in everyone leaving. The last would have been tantamount to throwing out both baby and bath-water, since the company depended on the flair and independence of the principal developer, which might have been destroyed were he to have been persuaded to operate in a more "conventional" mode. The second strategy, to encourage evolution, is consistent with that proposed for small companies by the authors of CMM [7], and was the only viable option.

There were also a number of constraints. Foremost amongst these were the commercial pressures. The company was small, but growing rapidly. To maintain this performance, it was imperative that external deadlines – the two software exhibitions each year – be met. The company had to keep ahead of its competition, so each release had to be innovative. Missing an exhibition might have destroyed the company. However, by its very nature, software development at level 1 of the CMM is inherently unpredictable, and makes forward planning extremely difficult.

A second set of constraints concerned personalities. One cannot simply tell a successful innovator to do things differently. On the other hand, the prospective team members have to feel satisfied with what they are doing, and not forever subordinate to the principal developer. Further, they need to be persuaded to work as a team, even though the principal developer might seem to "get away" with a less disciplined approach.

Finally, everybody was far too close to the problem – indeed, totally embroiled in it – to see how to deal with it.

The Teaching Company Scheme Recipe

Teaching Company Schemes (TCS) attract substantial government funding to promote technology transfer between academia and industry. Within a scheme, one or more recent graduates are employed on 2-year fixed-term contracts as *Teaching Company Associates* (TCAs). During that time, they are employees of the academic partner – usually a University – but they work in the company, following the company's agenda. They are supervised jointly by industrial and academic supervisors, the latter acting also as a consultant for the project.

The Associate attends several relevant training courses. These address crucial skills such as working in a team and project management; the skills gained in these courses were invaluable for this project.

The Associate, being an employee of the University, is in a curious position. He works in the company, and yet is, to some degree, insulated from it; in this particular project, this facet of the scheme was probably crucial.

When a scheme is proposed, a project is identified for each Associate. In this case, the project chosen – a significant addition to the company's product – could be achieved only if there were an effective development team; there was no intention to repeat the "mistake" of having yet another product developed by one individual.

The final ingredient in the recipe is the role of the academic supervisor, which can vary between different types of scheme. In this case, the role was that of an external observer who was ready to ask awkward questions. A traditional consultant's role may not come easily to academics, but this particular role certainly does!

How it worked

The Teaching Company Associate (TCA) was asked to develop a major new facility for the Company's recently launched product. He started, and did his best. However, he very soon encountered problems.

First, he faced an extremely steep learning curve. Not only was the existing product poorly documented – for the usual well-known reasons – but the Associate was unfamiliar with the language in which the product had been developed. Despite this, he was expected to produce something tangible within a few months.

When the TCA encountered difficulties, the principal developer could always solve the problem. In fact, it was always quicker for the developer to do it himself. The problem was that the developer was always too busy. At about this time, other members of the development "team" were starting to leave, apparently because they were experiencing similar problems.

It was at this point that the academic was able to contribute, not to the substance of the project itself, but to the manner in which it was being pursued. All involved "knew" that some basic project management techniques were needed, but everyone was "too busy" to be bothered with them. Furthermore, external suggestions, made directly by the academic, would probably not have been well received.

By helping the TCA to understand the basic causes of his problems, and then to discover, *for himself*, the "standard" techniques which could be brought to bear, some of these techniques could be introduced *from within the team itself*.

For example, one of the first problems encountered was tracing bug fixes – had they been done, by whom, in which release, and so on. Once he had demonstrated how much time was being wasted by bugs being investigated, and possibly fixed several times, it was easy to introduce a single point of reporting with a bug trace database.

A similar problem was coordination of effort. Everybody was always busy, but it was not always clear with what. Some people had tasks piling up, whilst others were waiting for colleagues to complete tasks, but with no idea of when completion was due. A simple task allocation chart, shaded to show progress through tasks, was readily introduced once the problem had been verbalised.

Similar approaches were taken to problems of poor documentation (leading to attempts at module specs and even code reviews) and "feature creep", whereby the development team were under pressure to include the latest "bright ideas" into an impending release, rather than consolidating on a set of agreed features; the latter problem, once recognised, led to some rudimentary product planning.

Not all of the techniques the TCA suggested were adopted. Some techniques failed – initially. Others were seen as a waste of time. However, what really mattered was not that all the "correct" techniques had been applied, but, rather, that he was initiating a culture change. And culture change is (very) difficult.

A Long-Term Relationship

A major strength of a Teaching Company Scheme is that the interaction takes place over an extended period – typically two years. Unlike "standard" consultancy, wherein a consultant appears, asks questions, pronounces and then disappears, this was a long-term partnership. Further, the academic is not working full-time in the company– he visits about once a week – so he is not drawn into the politics – or the emotions – of the problems. In essence, the academic is providing a drip-feed of objective external appraisal.

After several weeks of interaction, the TCA had changed his focus. What became important was not that *he* develop the required new facility, but that the *team* develop it – a clear example of "egoless programming"[24]. In order for the team to be able to develop the new facility, it had to function *as a team* rather than as a set of individuals. Hence, the mission for the TCA became the introduction of better working practices – in fact, moving the company towards level 2 of CMM.

Furthermore, to introduce change to the team, the TCA needed to be part of the team, but protected from being regarded as a troublemaker. Being a University employee protected him, so that he had the confidence to be persistent.

Potential Pitfalls

This project could easily have gone horribly wrong. If just one "improvement" championed by the TCA had disrupted development so that a release deadline was missed, and the whole scheme would have been in jeopardy. Alternatively, the TCA could have upset his colleagues – or they might just have got tired of being nagged.

The risk of any of these occurrences was minimised because the TCA was so deeply involved himself in the production and maintenance process; indeed, he had overcome the learning curve surprisingly quickly, so that he was a respected member of the department. He was also very persuasive. This is probably the most significant point: it was often by sheer force of personality that he was able to persuade his colleagues at least to *try* different ways of working – a clear example of a participant observer changing his environment [12].

It was always possible that some of the techniques introduced TCA might be abandoned when he left the department; indeed, acceptance by the client provides the primary validation of the approach [10]. Some have, indeed, been given a lower priority. However, during the project, the company recognised the need to employ a project manager, to build on the changes first introduce by the TCA.

In retrospect, despite the many opportunities for failure, nearly all seem to have been avoided, at least thus far. What this needed was sensitivity, on the part of all concerned, to what the TCA was attempting to achieve, and when that could – or could not – be allowed to interfere with the development schedule.

4. THE OUTCOMES

Achievements

The company is still in business, and, indeed, the "acorn" has grown into a healthy, medium-sized oak.

Several "standard" techniques have been introduced for managing the development and maintenance of the company's products. These address all the key areas of CMM level 2, apart from subcontractor management, which was not directly applicable. Moreover, the team did develop and deliver on time the required new facility, retaining the company's position as a market leader. During the period, turnover increased by more than 50%, largely from sales of the new product.

By the end of the project, the development section of the company had grown from a principal developer with a couple of "assistants" to a team of 10 developers plus a manager; that growth has continued subsequently. The team is able now to work in parallel not only on several features but also on features for more than one release.

Version releases, although still not without drama, are no longer a last-minute panic a few days before an exhibition. In part, this is due to the fact that the features to be included are planned rather than whimsical, and also because the entire team – since they are all familiar with the whole product – can assist with the integration and testing stages of a release. With a more organised approach to releases, coupled with basic quality assurance procedures, the incidence of "bugs" is (relatively) lower – and there is also a procedure for tracking them.

Finally, the company is no longer losing developers. What is more, the principal developer is still there, still innovating, and working *with* the team, rather than independently of it.

5. CONCLUSIONS

Having seen that the company seems to have achieved (most of) CMM level 2, we consider now what factors promoted success, and how that success might be translated to other enterprises.

First, there was determined commitment from the company. Although they had never articulated their real problem, it was clear that they knew what it was, and were looking to the programme to address it. Furthermore, they accepted input from an "unworldly" academic – but possibly because they knew that input was needed.

A second major factor was the willingness of the team to embrace change. Old habits are always hard to break, and the change from individual to team work is notoriously difficult. Nevertheless, the team did not just tolerate the TCA's suggestions; they actively followed them, even when the benefits were not immediately obvious.

One of the most important factors was the character of the Associate himself. The project would probably have had a totally different outcome had, say, an experienced C++ programmer been appointed rather than someone who not only had to face the steep learning curve but also both recognised that it should not have been there and was prepared to do something about it. In his refocused role, the Associate benefited from his stubborn determination and his ability to interact with all those involved.

It is possible that the success of the project was due, in part, to the fact that software process improvement was never stated explicitly as an objective – had it been, those involved might well have tried too hard. Many companies have allegedly "adequate" project management procedures that are breached more often than they are observed.

Finally, the protracted drip-feed of observations from the academic was also significant. Being both outside the environment of the company and also able to walk away from it after each visit allows the academic to remain far more objective than (s)he might were (s)he to be more closely involved with the project or the company.

There should be many opportunities for applying this approach in other situations. The essential feature is for the introduction of a new team member – a contractor, perhaps – coupled with long-term external consultancy. Whilst such arrangements are supported admirably by Teaching Company Schemes, analogues, such as contractors and consultants, might work equally effectively.

As always, however, the most important factors are for the relationship between the company and the consultant (academic) to be sufficiently positive, and for the right person to be appointed as the Associate. As in this case, personality and the ability to interact with others can count for far more than technical ability alone.

6. REFERENCES

1. Paulk M, Curtis B, Chrissis M and Weber (1993), C, "*The Capability Maturity Model*", IEEE Software, **10**(4), pp.18-27.
2. Bamberger J, (1997) "*Essence of the Capability Maturity Model*", Computer, **30**(6), pp.112-114.
3. Arthur L, (1997) "*Quantum Improvements in Software System Quality*", Communications of the ACM, **40**(6), pp.46-52.
4. Billings C, Clifton J, Kolkhorst B, Lee E and Wingert WB, (1994) "*Journey to a Mature Software Process*", IBM Systems Journal, **33**(1), pp.46-61.
5. Fox C and Frakes W, (1997) "*The Quality Approach: Is It Delivering?*", Communications of the ACM, **40**(6), pp.25-29.
6. Herbsleb J, Zubrow D, Goldenson D, Hayes W and Paulk M, (1997) "*Software Quality and the Capability Maturity Model*", Comm. ACM, **40**(6), pp.30-40.
7. Paulk, M, (Oct 1998) "*Using the Software CMM in Small Organizations*", Proc. 8th Int. Conf. On Software Quality, Portland, Oregon, pp350-361,.
8. Darke, P, Shanks, G and Broadbent, M, (1998) "*Successfully completing case study research: combining rigour, relevance and pragmatism*", Info Systems Journal **8**(4), pp273-289.
9. Lee, A, (1999) "*Rigor and Relevance in MIS research: beyond the approach of positivism alone*", MIS Quarterly, **23**(1), pp29-34.
10. Vinten, G, (1994) "*Participant Observation: A model for organizational investigation?*", Journal of Manament Psychology, **9**(2), pp30-38.
11. Davenport, T and Markus, M, (1999) "*Rigor vs. Relevance Revisited*", MIS Quarterly, **23**(1), pp19-23.
12. Walsham, G, (1995) "*Interpretive Case Studies in IS Research: Nature and Method*", European Journal of Information Systems, **4**(1), pp74-81.
13. Bowers D, (Nov. 1999) "*From Prima Donna Coda to Harmonious Ensemble*", in "Business Information Technology Management: generative futures", Proc. 9th Annual BIT Conference, Manchester.
14. Brooks F, (1975) "*The Mythical Man-Month: Essays on Software Engineering*", Addison-Wesley, Reading, Massachussetts, 195p.
15. McGuire E, (1996) "*Factors affecting the quality of software project management: An empirical study based on the Capability Maturity Model*", Software Quality Journal, **5**(4), pp305-317.
16. Reiblin S and Symons A, (1997) "*SPI: 'I can't get no satisfaction' – directing process improvement to meet business needs*", Software Quality J., **6**(2), pp.89-98.
17. Saiedian H and McClanahan L, (1997) "*Frameworks for quality software process: SEI capability maturity model vs. ISO 9000*", Software Quality J., **5**(1), pp.1-23.
18. Paulk M, (1993) "*Comparing ISO 9001 and the Capability Maturity Model for Software*", Software Quality Journal, **2**(4), pp.245-256.
19. Benbasat, I and Zmud, R, (1999) "*Empirical Rsearch in Information Systems: the Practice of Relevance*", MIS Quarterly, **23**(1), pp3-16.
20. Boon, S and Ram, M, (1998) "*Implementing Quality in a small firm: an action research approach*", Personnel Review, **27**(1), pp20-39
21. Yin, R, (1994) "*Case Study Research – Design and Methods*", Sage, p.80.
22. Klein, H and Myers, M, (1999) "*A set of principles for conducting and evaluating interpretive field studies in Information Systems*", MIS Quarterly, **23**(1), pp67-94.
23. Ragsdell, G, (2000) "*Engineering a paradigm shift? An holistic approach to organisational change management*", Journal of Organisational Change Management, **13**(2), pp104-120.
24. Weinberg, G, (1971), "The Psychology of Computer Programming", VNR, p56...