AP-M Appropriate Process Movement

Process Configuration for CMM Level 0

Hüseyin Angay

Karabash Ltd.

Member of the Appropriate Process Movement

http://www.aptprocess.com

Abstract

Capability Maturity Model introduces the five levels of maturity for an organisation's software development processes. These are Levels 1 to 5. They have, however, left out Level 0. They are probably afraid that somebody might actually try to achieve it if they discovered that it was there. Of course, we are not SEI. So, for those who would like to go where even fools fear to tread, here is an explanation of what constitutes CMM Level 0. For the really foolhardy, we even added some techniques for getting there.

Copyright Note

Sensible Note

In case you've had a long day and all this appears to be genuine advice from a professional engineer, please get some sleep and read this piece in the morning, when you will feel better and less gullible.

We would love to hear from you if you have more tips or if you would like to correct any point.

We have no connection with the manufacturers, distributors or the retailers of Wonder Millennium Potato Peelers. If you would like one, the Liverpool Lime Street station is the place to go.

This is, of course, meant to be humorous. Tell us if you do or don't find it funny and why, but not if you are mad enough to take it seriously.

Process Configuration for CMM Level 0

Process configuration is all about challenge. You set the next challenge; you achieve it; you move on. If you are the selfish type, you can treat process configuration as a gravy train, place yourself in the conductor's seat and have a job for life, steering your organisation through the journey that never ends. Or, you can take the selfless route and go in a blaze of glory, having achieved the ultimate challenge – that of reaching CMM Level 0 accreditation.

Let's face it. Everybody's at CMM Level 2, at least. Ask anybody in the industry, and they'll tell you that they are doing the same things over and over again – this may not be your definition of repeatability, but it's certainly mine. CMM Level 3 is no challenge at all, either. Ask any organisation that issued the memo saying that they will reach CMM Level 3 by Christmas and managed to do it – you will not be short of candidates. Having reached Level 3, it's no great shakes to move up the ladder to Levels 4 and 5. You just issue more memos, one to get to Level 4 by Christmas in two years' time and another to do level 5 in four years' time – by Christmas again, naturally[1].

On the other hand, CMM Level 0 is the Fight Club of the software development world. It's that point where you have sunk so low that you've actually come full circle and became more than superhuman – well, ok, super-organisation. So, how about it? Have you got the guts?

What is CMM Level 0, then?
Well, Level 2 is Repeatable. Level 3, Defined. Level 4, Managed. And Level 5, Optimised.
If you are none of these, you are Level 1.
So, how do you get to Level 0 in the first place, since if you do this stuff you are at some Level 2-5, and if you aren't doing any of it, you are at Level 1. And presumably, if you manage to do all that AND also manage to implement the air traffic control system for Europe, say, with a crew of six (one of whom is solely dedicated to making the teas and fetching the coats), you must be at Level 20 or something. But Level 0?

[1] I know a lot of companies announce their exec bonuses at Christmas, but this is bound to be just coincidence.

Think of Level 0 as the platform 9¾ of the Process Configuration gravy train station. The difference between Levels 1 and 0, in a nutshell, is one of intention and direction. Organisations are at Level 1, because they fail to achieve the qualities that make their processes repeatable, defined, managed and optimised. To get to Level 0, they must not *fail to* do these things; they must *manage not to* do any of these things even when everything appears to be in place for them to be at Level 5. And that subtle difference, dear reader, is the quantum gap between the *failed* process configuration engineer and the *ultimate* process configuration engineer. The former will get nothing but contempt, whereas the latter will get the recognition that infamy brings. The choice is yours:

> You can do your best and still fail to lift your organisation above Level 1 and be a contemptible failure.
> You can take your organisation to some higher level and become moderately or even very successful.
> Or, you can take your organisation down to Level 0 and become (in)famous.

Do you feel big enough for the challenge?

Assuming that you do, here is how you make a process:

- unrepeatable even with the best of intentions;
- undefined, even though it looks well defined;
- unmanageable even by the best managers;
- and so far away from the optimum that it's not merely sub-optimal but infra-optimal.

CMM Level 2 – Repeating the unrepeatable

Repeatability is based on the assumption that if you repeat your successful practices, you should consistently do better than re-inventing the wheel every time. This is a tough one. It is quite inevitable that despite your best efforts, there will be some individuals, or, perish the thought, groups of individuals who will want to repeat the odd practice that helped them get things done. You can reduce this tendency by making sure that they have no chance of getting things done in the first place, so that they can't learn anything from it, but they will occasionally get things right by pure fluke if for no other reason. The strategy for this is quite involved. You must convince them that this is an isolated fluke so that they won't expect it to work again. If they want to try it again, make sure that whatever they do will contradict the defined process, so that they can never do it. If they

This is, of course, meant to be humorous. Tell us if you do or don't find it funny and why, but not if you are mad enough to take it seriously.

are really sanguine about it, insist that they must thoroughly document how they did it, at which point either they will give up or they will do such a good job of documenting it that it will become part of the process monolith documentation. Either way, we won't have a practice to follow or we will have it so well documented that we will never be able to follow it – especially if we mandate that it fits perfectly well with the documentation (see Level 3 for this).

As to the ones who learn from their mistakes, they are obviously smart so you'd think that would make them harder to crack. Of course not! The smarter they are, the harder they fall. Make sure that they thoroughly document how they did things. While they are busy doing that, make their previous project turn out to be a great success and have them lionised. Nobody's going to turn round and say, "Oh, by the way, you know that great success story that you rewarded me for so handsomely? Well, it was a great failure actually." In fact, it's likely that they'll actually believe that it was a great success: Learning from your mistakes is great, but not making them in the first place is even better. We have excellent denial mechanisms to bolster up our self esteem. When other people start to stroke our egos, these mechanisms go into overdrive and create an unreality field around us, which affects us and all those involved. This is how myths are made. At this point, you can take the documented mistakes, which now turn out to have been the best decisions for the situation, and make them part of your process. Two birds with one stone: you've enriched the process with even more problems and quietened the people who might have taught everybody else how to avoid the pitfalls. If people follow the same advice and fail (as they are bound to), they obviously haven't followed it properly, or they would have done well just like the original inventor of the idea.

If you thought learning from mistakes was all there was to it, it's not surprising that you're reading this, instead of getting on with real work. To completely miss the Level 1, you also need to address the requirements management issues.

Now, we know that more is better with metrics. With matrices, there is only *more* – *less* doesn't make it even into the runners-up roster. When you think requirements, you must think matrices. You should have matrices of related requirements, matrices of requirements priorities, matrices of requirements and features, matrices for everything. And to have matrices, you must have traceability. Some

This is, of course, meant to be humorous. Tell us if you do or don't find it funny and why, but not if you are mad enough to take it seriously.

people believe that the aim of traceability is to know why some software features are there – you look up the requirements attached to the feature and you have some idea. If that were really the case, can you explain why nobody's bothering to do it right? No, the real aim is to be able to supply the matrices with the cross-references that they need. Matrices are hungry beasts. You make one up and it looks sparse. So, you supply the cross-references to make it less sparse, but then, while looking for cross-references, you end up finding even more rows and columns and you've got a sparse matrix again. The more you feed them, the more they grow and the hungrier they get. And that's exactly what we want.

If you want a fat beast of a requirements repository, encourage as many types of interdependencies as you can. Then make sure that everyone finds as many of those as possible. Invent statistics that give the optimum quantity of connectedness for systems[2], then make everybody try to get as close to that number as they can. Penalise those who exceed it or come under it. Then, just as they start to think they know what's going on, crank up the traceability targets yet another notch: Introduce audit trails for everything people do, from compiling code to making changes to the size of the buttons on the screen. Just as people start to groan under weight of the paper, introduce a paperless audit trail by integrating configuration management with some kind of work tracking tool. This may appear more efficient at first, but when you notice how many policy changes you can make on the fly without anyone noticing, you will appreciate the value of an automated tracking system. Then, produce statistics – and matrices, of course. Most non-IT people's eyes glaze over when you put metrics about LOCs, function points, etc. in front them. Start giving them metrics about requirements and system features, though, and they'll start believing that they understand it all. Of course they don't. But the vocabulary appears to make sense, the words appear to be in real English instead of some techie perversion of it, so it must be sensible. So, their eyes will still glaze over, but this time, with the peaceful sleep of recognition and not with the panicked shutdown of the non-numerate. These semi-recognisable metrics are also a good way of gaining a foothold for the even less productive but even more

[2] If you aren't numerically inclined or if you just can't be bothered, look it up on the internet. Somebody's bound to have come up with the figures. The world is full of crackpots – we have to give them a purpose in life or they'll go inventing cars running on water or something and destabilise the whole economy.

useful (for us, anyway) metrics of the pseudo-technical nature, which will be explored in Level 4.

How do we know we are doing things right over and over again? The easy method, of course, is to say, at the end of the project, "The system ended up on the users' desk, so it worked." Life is not that easy, however. There is always some stuck-up user who will say something along the lines of "Thanks, but I was expecting this to be delivered sometime in the last century." This is where project planning comes in. It would be wholly inappropriate to disappoint the users by delivering something on time or on budget, but even more inappropriate for them to be able to prove that you have missed these targets.

So, introduce project planning early in the project – preferably, before it even starts. Have detailed plans, where every single task is identified. Provide estimates for each task, to the nearest hour, if possible. We don't want to appear to be tracking every time someone sneezes, but we want to do it, all the same. Underplanning is the deadliest sin that an organisation could commit. We will be tracking the project progress against estimates (and configuration management will help us work out even how long people spend over their work) and we will be holding the managers responsible for failing to stick to these estimates. Soon, they will all be trying to make sure that every one of the tiniest tasks in their projects are identified up-front and accounted for. Since the plans are made early enough in the project, there is naturally no way of getting the figures right. But the cure is simple and every manager will soon discover that they can just overestimate everything to make up for the unforeseen tasks. This is when you spring your trap: Chart the budgets and the projects for the next few years, which will no doubt show an astronomic potential overspend. Wouldn't it be wiser to subcontract everything, if this were the case? Torn between the desire to inflate the figures and the fear of being outsourced, the managers are now putty in your hands.

A threat never realised is a threat soon forgotten. So, you will need some subcontracting activity just to keep people's minds focused. But subcontractors can be a problem. They are quite likely to bring in ideas alien to your process, such as *iteration*. They also tend to use dangerous words like ‘agile’. Counter with baselines and lawsuits. One sure trick is to assign the wimpiest and most incompetent manager in the company to the management of the subcontractors. Then, put the manager under excessive pressure and imply that they are being led

by the supplier instead of leading them. The combination of timidity and the desire to be assertive should produce uncontrolled aggression. The manager will swing between letting the suppliers get away with blue murder and shouting at them for small failures. Hammer the suppliers for their failures, to the extent that they will lose money with every single job that they do for you. Sooner or later, all the reputable suppliers will be avoiding your calls for tender and you will end up with proposals only from the bottom-of-the-barrel lot that would tender for anything out of sheer desperation. Perfect. These guys will eat out of your hand.

Quality is such an important concept that it needs to be repeated to be appreciated.
Quality.
And again.
Quality.
Louder.
QUALITY.

For the quality assurance team, select the weaseliest characters in the organisation and import more if you can't find enough of them. They are easy to spot. Find out who manages to go on training courses more often than others but always avoids the events in the crummier hotels; who get their hands on corporate hospitality tickets for sporting events far too regularly; who always cluster around the coffee machine talking very quietly. Make this the cushiest job in the company – some good strategies: get them the airiest office space, new machines, regular training in posh locations and make them the only people eligible for overtime remuneration in the company. Remember, quality is far too important to leave in the hands of disgruntled employees; you can't have failed to notice that the countries who pay their policemen the least also appear to have high levels of crime and corruption. Spread the rumour that there will be more openings if the team is not up to the task. That way, nobody will be able to criticise the team openly because they will be hoping to be in it sometime soon. Except that the quality team will not be too keen to lose or dilute the perks of the cushiest job in the organisation. So, they will make sure that all the contenders for the job look far too incompetent to be in the cream team. This should be relatively easy, since they will be making the criteria for quality and they will be policing it. This is a great way of creating tension without anyone

This is, of course, meant to be humorous. Tell us if you do or don’t find it funny and why, but not if you are mad enough to take it seriously.

being able to complain. What are they going to say? "We don't want to produce deliverables of sufficient quality"?

Level 2 is thus conquered. Introduce these practices and your organisation is guaranteed never to reach Level 2, even though it will think it is already there.

CMM Level 3 – Defining the indefinable

Badly defining a process is easy. I'm a lousy magician – even a three year old can see that I hide the cards under my belt behind my back; even then, I end up pulling out the wrong one, anyway. It's much harder to *appear to be* a lousy magician – you know, the kind who saws the assistant in half and all this blood starts pouring out of the box, and he goes all white but then the assistant steps out of the box, unharmed. The archmagician, of course, is the one who really kills the assistant, but still manages to make her look alive.

Now, let's mull over this for a moment. How can we kill a process and still make it look alive? Well, it's easy to do in principle but hard to execute: We document it to death.

Now, the military are the masters of this game. They have manuals to cover every eventuality, from someone shooting himself in the toe to the outbreak of global nuclear war. Then, they learn the manuals. By heart. Then, they practice it and then practice it even more until the opportunity arises to do it for real. But, your average software person isn't like that. In fact, they'd rather not have any manuals at all. We could blame it on a liberal upbringing by misguided parents or we could be thankful that they are like that, because that's our key to *process misdefinition*. It doesn't matter how badly your process fits together as long as it appears to fit together. The software developers won't follow it anyway – everybody knows they won't. So, who can blame you if they don't follow yours, either?

Remember, processes are like movies: The important part is not reality; it's verisimilitude. If it appears real, that's good enough. In fact, just as the epic on the cinema screen looks more real than actual reality, processes look more real on the computer screen. So, produce massive tomes of process manuals and make sure that it all appears to fit together. Then drop them in front of the development staff and make them stick to the process. This is where the fun starts, because

their natural liberal tendencies will ensure that they won't follow the manuals. They can't help it. They'll do their own thing and they will fail, the poor chaotic creatures. But you've got your backside covered, because even if they followed the process, it wouldn't work anyway – except, you could still blame them with not following the process and everybody would believe you. Perfect.

So, who defines these processes and who documents them?

Here is the clever part. In order to distance you from any blame if things go not just conveniently wrong but *really* wrong, you delegate the work to the Process Engineering Team. This has the added advantages of reducing the amount of work you do, increasing your influence (you now have minions) and driving your productivity through the roof (ten monkeys type ten times faster than just one monkey; it will still be gibberish, but who cares – in fact, buy them dictation software while you're at it).

Dedicate a group to the definition of the process. Make this group's conditions even cushier than quality assurance. By the way, you have the perfect individuals for this team: They are currently in the quality assurance team. As soon as they move to their new job, make them aware that if you can move them to this position, you can move them out, too. Also imply that everybody wants to be in the team and there are quite a few good candidates amongst those who replaced them in the quality assurance team. When they hear this, they will be producing documents and ideas like crazy in order to prove that the group is not under-resourced – who wants another half a dozen people sharing their slice of the pie? They'll also do their best to produce a byzantine process that will be so difficult to follow that all the minions trying to follow it will look utterly incompetent, which will ensure that those minions cannot be promoted to the process team. And since the process team oversees the quality assurance team's processes, too, they will manage to make even the quality assurance people look incompetent. This can then start a lovely chain reaction where the quality assurance will pick on the software developers and so on.

Lest you get accused of being some religious fanatic, you must allow the project teams to tailor the process to their own needs. Whatever you do, don't use this as an opportunity to voice your doubts about people's capability to tailor the process. After all, you've empowered

This is, of course, meant to be humorous. Tell us if you do or don’t find it funny and why, but not if you are mad enough to take it seriously.

them to control their development. Remember, the process is so big that they cannot have a hope to acquire enough knowledge to tailor it. Convince everyone that, before they can tailor the process, they must know how to do it successfully in the first place. This will take them a year or two, during which the process will have changed somehow. On top of that, publish the process on-line using some kind of content management system. This makes it look very slick, indeed. It also means that anyone wanting to tailor the process documentation to their own needs must first learn how to use the content management system. If you pick a system with loads of features and get it from a company that has another couple of fierce competitors, you can be sure that the feature set will grow faster than any techie who also has to manage a project could ever hope to cope with. Sooner or later, they'll give up.

The size of the process brings us to the training issue. Can we expect anyone to be able to cope with the process without any training? Hopefully not. If they can, you've definitely got it wrong. Back to the beginning, then.

Since this is an internally defined process, the training needs to be delivered internally, too. And since the process evolves to meet the organisation's needs, schedule regular refreshers.

Internally written and delivered courses have so many advantages:

- They are perfectly tailored for the organisation by the people who know the organisation and its needs.
- They save money, because we don't pay anything to an external provider.
- They work wonders for the career progression of the trainers, especially if they were never trained in giving training before.[3]
- They are the perfect vehicle for the resident office comedians.
- If you get the right hotel (let's face it, in-house facilities are never good enough for such important occasions), everybody will have a good time and go back to work smiling. At a time when the corporate entertainment budgets are being squeezed dry, you should see it as your holy duty to bolster up the staff morale – nobody else will do it for you. And more importantly, you'll

[3] Try to avoid external "train the trainer" courses. They are such a distraction and they play havoc with the morale of the process tailoring team, who may get the notion that they are being less than honest in their endeavours.

This is, of course, meant to be humorous. Tell us if you do or don't find it funny and why, but not if you are mad enough to take it seriously.

never be stuck for your week-end breaks, again – hotels are very generous to clients who bring in big delegations during the week.

Having sorted out the tailoring and training needs, you need to turn your attention to the day-to-day running of the projects. Such a drag, when done badly, I know, but it doesn't need to be. Convince your organisation that most of their management problems come from having large teams. Make the teams smaller in order to reduce the management overheads. Nobody will complain. Staff will see that you are working towards higher agility, and because smaller teams create more management positions, it will enhance their career progression prospects. Team leaders won't complain, because from running a single team, they will be promoted to running several teams with managers under them – suddenly, they are further up the hierarchy. The higher management will not object, because the hierarchical baobab tree underneath them will now be even bigger. For the benefit of the oddbods who read too many books for their own good, convince them that smaller teams encourage flatter management structures. If they fail to get the joke, change the corporate vocabulary to use words like 'support' instead of 'manage'. It will confuse the hell out of everybody (is this 'support' as in 'work for' or 'support' as in 'manage'?), but will sound contemporary, liberal and touchy-feely, so no one will dare whinge anymore.

So, what do smaller teams achieve? Apart from improving the morale of the whole organisation (everybody now believes that they got something for nothing), it also creates excellent scope for inter-team and interdepartmental dependencies. You now have a whole new playground where

- system architecture can be defined along team boundaries, which will lead to a proliferation of components;
- feudal attitudes will make the documentation spread across many servers, domains, cupboards, desks and waste bins – creating an excuse for another workstream to rationalise the documentation needs of the organisation;
- deeply held enmities will resurface, especially if the team divisions accentuate role divisions[4];

[4] It is essential that splits run along the lines of 'analysts' team', 'designers' team', 'architecture team', 'developers' team' in order to create centres of excellence for various disciplines. You can then create separate project teams and lend the people

This is, of course, meant to be humorous. Tell us if you do or don't find it funny and why, but not if you are mad enough to take it seriously.

- the opportunities for meetings, memos and management week-ends multiply by ten-fold;
- the need for improved communication techniques (instant messaging, desktop conferencing, electronic whiteboards and other corporate toys) goes through the roof.

If the organisation is dead against a proliferation of small teams, all is not lost. You can achieve a lot of dissent with only two or three teams. First, devise a team structure along architectural divisions, making sure that the teams are populated by proponents of competing technologies. A Microsoft vs. Somebody else's solution split is a strategy that never fails. Then, find developers who make a career out of incompetence and failure. Plain failed developers are no good; those who have built a successful career on a series of failed projects are the ones you need. Failing to deliver anything working is not good enough – they also need to be able to make sure nobody else around them delivers, and still come up smelling of roses. Now, if you had only one team that did that, this would look like your average failed project. But if you have two or more teams who collaborate in the failure, it's a different story. Remember, we have competing technologies here. We also need to arrange things so that both technologies will be used in the same system. This will ensure that there will be plenty of scope for either architectural feature being able to encompass the functionality of the other. In most organisations, the teams that control most of the architecture control the projects, too. So, tell each team independently that whoever gets its solution out first will call the shots. Now, they'll nicely work against each other, to the point of sabotage (like changing interfaces on the fly and then telling everyone about it in a thousand page document so that they have no hope of finding out what's really happened). If you set the goals small enough and iterate, there will be many architectural features assigned to one technology or other, haphazardly.

When your team rivalries reach maturity, it will be time to introduce integrated management of projects. Don't consider projects in isolation. They are all interrelated. When you have big team rivalries, this allows the fighting to transcend project boundaries and to spread

---

from these centres of excellence to the project teams. Now, everyone's working for at least two managers, and everyone also brings all the role rivalries with them, but they then take the project rivalries back to their centres of excellence, as well. Soon, nobody will be able to tell friend from foe. Yes, like chips, matrices go with everything, even management.

This is, of course, meant to be humorous. Tell us if you do or don’t find it funny and why, but not if you are mad enough to take it seriously.

to all other projects, too. The more architectural features the projects share, the more they will be affected by these fights. And since the disputed territory is now bigger and sweeter for the rival teams, they will fight even harder.

So, we managed to drive a wedge between groups and we fragmented those groups further, too, so that there is plenty of enmity even within groups. The last step is to ensure that all remaining vestiges of solidarity are driven out of the organisation, even at individual level. Your problem is that, having driven apart the various groups within the organisation, you will have actually driven the members of those groups even closer together. This is not acceptable. Small groups that gel together really well can do a lot of damage to your plans. But how do you turn them against each other? Enter peer reviews!

We are human. We are fallible. We can conquer that fallibility because we are also a social being: We have colleagues who can help spot our mistakes so that they can be corrected before they become problems. At least, this is what you tell everyone. There are a few right ways of doing peer reviews and, thankfully, there even more wrong ways of doing them. Peer reviews can be a stressful affair – except when the peers involved get on well, at which point they risk becoming constructive. Now, here is the dilemma: If you have peer reviews between groups and if the groups are already at the feuding stage, there is little to be gained from a negative review, except to flame the feud a little further. Good, but not good enough. On the other hand, if you have the review within a group, they are quite likely to let each other off lightly, so you've got nothing to show for the effort. How do you resolve this? Enter review guidelines!

Everything we produce in the organisation should be the best we can produce. Only if we strive for absolute excellence could we hope to compete in today's cutthroat marketplace. Unless we achieve excellence internally, we can never hope to achieve excellence with the products and services that our customers will receive from us. So, you must produce the most stringent quality guidelines possible for all the deliverables within the organisation. The guidelines will be so stringent that nobody could ever hope to get everything right. But that's ok. Process improvement is a journey. We have our targets and we strive to reach them. The peer reviews' goal is to point out the pitfalls on that road towards perfection. It's ok to get it wrong, as long as we know where we got it wrong, so that we can do better next time.

This is, of course, meant to be humorous. Tell us if you do or don’t find it funny and why, but not if you are mad enough to take it seriously.

Except, of course, it will still hurt when you tell someone that their code or document or model scored four out of ten on the excellence scale. And if it's one of their best friends who tells them this, it will hurt even more. And just to make sure that the best friend doesn't go easy on the victim, we will make another group also do a peer review on the same deliverable and give them scope to get really agitated if the same checklist produces different results.

The beauty of this is that, once someone receives a bad review, when it is their turn to do a review, they are more likely to be negative towards their former tormentor. Go through the cycle a few times and see if there is anyone left on speaking terms in the organisation.

What should go in the review checklists?
For a start, a document not elaborated to the finest level of detail will obviously be scored low – you can't build a system from insufficient information. And one that goes into too much detail would be scored low, too – we don't want to get into analysis paralysis.
You'd also expect everything to have been reviewed and approved by all the departments involved with the production of related artefacts. And since they are all fighting amongst themselves, there will be very little chance of that approval being granted in a hurry.
Just for a laugh and added bitterness, you can introduce automated tools that will go through the deliverables and produce a lot warnings about anything from grammar to the use of *goto* in code. Remember the old *lint*? You gave it a hundred lines of C code that you were really proud of and it came back with ten thousand warnings. That's about the level you should aim for.

CMM Level 4 – Managing the unmanageable

Metrics are a matter of life and death for any organisation. They are an objective and qualitative way of indicating where we are and where we are heading and a way of helping us decide whether this is the direction that we really want to be heading. At least, they could be, if we let them, which we naturally won't.

One thing most people fail to understand about metrics is that they are numbers. If they did, their lives would be easier and ours more difficult. So, maybe it is a good thing that they don't. This point about numbers is important, because it shapes our attitudes about metrics in particular and statistics in general. Invariably, we either imbue them

with some kind of mystical power over reality[5] or we completely ignore their meaning because they look just too obscure. These two effects are the ones we are going to exploit. We are going to create metrics that are easy to collect, so nobody can complain that they take too much time; we will ensure that they become the main source of information about what is going on and we will destroy their credibility – in that order.

The first step is to pick your statistics. They must be easy to collect so that nobody can complain about the excessive administrative burden that they place on the development staff, but they must also be able to produce a lot more figures with still not much effort. Figures like the number of lines of code produced are perfect for this purpose. Think about it: All you need to capture is the number of lines someone produces in a unit of time, be it program code or user guides. You are of course using configuration management tools, so, it is relatively easy to track these figures when people check files out and check them back in. You can then cut them any way you want – LOCs per team per week, LOCs per project, LOCs per iteration per person... It's a statistics fest. It's the equivalent of the all-you-can-eat deal at the pizza joint – you've had enough, but you'll still have another pie chart, then another one and yet another one; it's all free anyway, so who cares.

The second step is to convince the organisational stakeholders that these figures are the solid facts about your projects and that they should be believed above all else. After all, people always lie about the amount of work they are putting in and about their productivity. If we can obtain subjective figures, that are independently collected and universally available, we will know the Truth about the projects. Now, tie all this to statistics available for productivity in other organisations and other industry sectors. Show how your figures compare with theirs and show how you can make the figures even better. Suddenly, you have their attention.

The final step is to destroy all credibility about these figures – at least, amongst the development staff. If you collected the figures in a pseudo-scientific fashion in the first place, you will have no trouble with this. For instance, you could write the scripts to count the lines of

---

[5] Anyone who's watched a cricket match on television knows that the statistics about the game, players, weather and other trivia takes more screen estate than the actual game, players, weather and other trivia do.

This is, of course, meant to be humorous. Tell us if you do or don't find it funny and why, but not if you are mad enough to take it seriously.

code in a shoddy way, so that it counts everything, including comments and line feeds. Software people are already suspicious of metrics anyway. Given the evidence in front of them, they will be convinced that it's all a façade and that it doesn't matter what the metrics are. But since the management now view these as the gospel truth, the management will have to be appeased. In this case, the development staff, all the way from the most junior to the CTO will make their utmost to lie with the figures and make them look as good as possible – especially if their rewards are tied to it. (I mean, who's going to get hurt if I have another few hundred lines of comments in my code? It's just some nonsense figure anyway. While I'm here, I might as well increase the spacing between my functions – makes them more readable.)

There you have it, then. Your developers are now spending their lives producing dross and getting rewarded for it, while the people holding the purse strings are seeing continuous improvement. Everybody's happy.

Now, we are collecting our metrics and we are feeding a bunch of guys apparently doing quality assurance, whatever that may mean to them. Why not combine the two? Why not, indeed? After all, this will give us quality management for free.

All we have to do is combine the metrics that we collect with the metrics that the quality assurance guys collect. So, we can have defect rates against productivity. We can also have all sorts of defects figures against projects, teams and individuals. Remember that we are aiming for zero defects – that's Six Sigma, if you want to impress your peers[6]. We will not get there until we know how badly everyone is doing. So, collect as many defect metrics as you can. That's quality management.

Of course, quality is not just about the product. It's also about how you produce things. One school of thought claims that if you get your process right, the results will be right, too. So, if they stick to the process we have configured so far... Perish the thought!

---

[6] If you like name-dropping like this, it is worth choosing busy moments so that they can't ask you to explain it in depth. It will leave the impression of a really knowledgeable person in their minds without you having to do much work to prove it.

This is, of course, meant to be humorous. Tell us if you do or don't find it funny and why, but not if you are mad enough to take it seriously.

When it comes to managing adherence to the process, metrics lift their blessed ugly little heads again, clamouring for attention. You see, you can actually collect metrics on process adherence.

How many change requests have been raised last month?
~34~.
And how many changes were there to the code base during the same month?
~83~.
What? We are making unauthorised changes? This has to stop!

How many lines of code do we have?
~A couple of million.~
How many use cases?
~Ten.~
What? So, we've got two hundred thousand lines of code per use case? Haven't you read section 28 of the manual about the traceability discipline?
[At this point, you have them by the metaphoricals. The developers, having delivered a thousand line of code daily for the last six months, cannot possibly admit to be producing mostly empty source code. The only way out is to admit that they hadn't been following the process to the letter.]

How many features have we delivered last month?
~Two.~
Two? Two? At this rate, we'll have delivered it all sometime during the next decade!
[Your metrics will wisely omit the amount of time people spend collecting the metrics and following the process. Besides, the developers have been inflating their productivity figures so much – since you told them it's all nonsense to keep the accountants happy, anyway – that nobody's going to believe they are spending a lot of time doing the process instead of doing real work.]

Now you have the attention of the development staff. They are failing publicly. And they are failing because they haven't been following the process, apparently. If they don't do everything in their power to stick to the process from now on (or at least, give the impression of sticking to it), they are ripe for sacking, anyway. If the recruitment policy has been favouring the jobsworths who will do what they are told (you

have done your work there, haven't you?), you need do not much more to convince everyone that what you say goes.

Like the man who markets the Wonder Millennium Potato Peelers outside the Lime Street station would say, "But that's not all you're getting for your money." Because metrics are the best fun you can have with a bunch of consenting (ok, only barely, but still consenting) adults without breaking the law. By this stage, the consenting adults aren't so sure, of course, but they are still tagging along. Not tagging along would imply that they haven't bought into the process, which would imply that they aren't following it, which would imply that everything's their fault. They are being carried along and they know that they'll fall and get hurt very badly if they stop running. So, they'll keep on running. Trust me on this. Now for the fun part, combine peer reviews and metrics. Simmer gently.

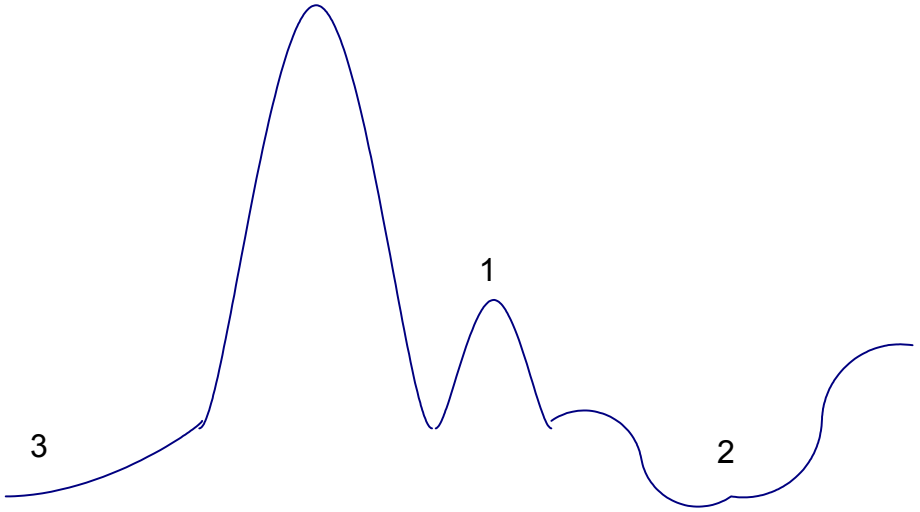
Think about it. Your stakeholders are constantly after better ways of measuring the performance of their suppliers – in this case, the software folk. They are getting wind that this lines-of-code business isn't all it's cracked up to be and the figures aren't all that reliable. But then, you breeze into the boardroom with a folder full of figures from the peer reviews. We are auditing ourselves now, and just to prove that we aren't pulling our punches, look at these figures: They are all below average. But that's ok, because we are on a journey. It will get better. And just to make sure that it gets better, we will introduce performance related bonuses that are tied to these figures. Naturally, we can't have an open-ended bonus pot – that would be mad. So, we decided to have a fixed bonus pool to be shared amongst the staff according to their performance figures. If Jack does better than Jill, Jack gets paid better than Jill. See them fight up the hill.

CMM Level 5 – Infra-optimisation

The journey is nearly over. By this point, the staff should be so browbeaten that they will consent to anything. They would even be prepared to do some real work for a bit of light relief. So, when you talk about optimisation and how it will make their lives better, they'll all sit up and beg.

If we want to reach Level 0, of course, we could hardly afford to optimise the process for good. What we aim for is something well below optimal. Sub-optimal wouldn't do. It has to be *infra-optimal*.

What is *infra-optimal*?
To have an infra-optimal process, things have to be so bad that everybody (including the tea boy) should believe that things could be better, much better. Nothing new with that, of course, so we have an additional criterion: Things must be so bad that changing anything would make things even worse, not better. So, even if everybody's unhappy with the situation, nobody can make things better, allowing you to claim everything's optimal without anyone being able to challenge the claim.
This sounds like a paradox. How can it be as bad as it can get and still have room to get worse? Not so difficult, actually. Look at the following curve.



It has a minimum and a maximum, but it also has other local minima and maxima. If you gave this to a fool and asked him where the most sub-optimal point was, he would show you the obvious minima: 2 or, maybe, 3. But think about it: When you're at the point where things cannot get any worse, they can get only better. Sooner or later, somebody will catch on and start on some improvement programme that will show positive results straightaway. Once they get the improvement bug, organisations can never shake it off and carry on improving things. Even when they get stuck in temporary ruts, like one of the local minima, they look at their past history of improvements, take heart and carry on. That's not quite what we want, is it? We want them to believe that things are as good as they

This is, of course, meant to be humorous. Tell us if you do or don’t find it funny and why, but not if you are mad enough to take it seriously.

can be, even though things aren't anywhere near as good as they could be.

So, the wise fool would have found a very low local maximum with steep drops either side. Say, 1. That way, if you move in either direction, things get even less optimal, which discourages everyone from trying out anything. Instead, they cluster on their tiny little peak of mediocrity, always dissatisfied, but also afraid to fall off. This combination of sub-optimisation and the threat of worse to come is collectively termed infra-optimisation.

There are many techniques for keeping people in their place. Here, we will concentrate on the corporate techniques because they are legal and much more effective than the corporeal ones. One such technique is making defect prevention everyone's business.

If somebody finds a defect, they should document it. This is hurdle one. If you want the brownie points, you have to spend ages to document the defect first. Could you be bothered? I didn't think so, either.

Having documented it, the most efficient method to deal with the defect is to try to fix it yourself. This is hurdle two. It stands to reason: If you remove bureaucracy from defect prevention, it will work more smoothly. But, since we are all responsible for our work and since the configuration management system we have in place tracks every tiny move we make, would anyone dare? Anyway, we replaced all the staff with jobsworths, having we? Would they lift a finger to fix someone else's defect? No, because everybody hates everybody by now, if you did your job well.

Having found the defect, they should identify source causes. The best way to do this is by committee. Committees are always more heartless than individuals when it comes to apportioning blame in generous doses[7]. So, start a weekly defect prevention meeting, where all the defects are aired and the responsible parties chastised. Within this sort of environment, everyone will try to keep their defects to themselves and will try to identify those of the others to the best of their ability. Fingerpointing is now the favourite sport of the staff.

[7] This is because committees cannot be blamed for things, whereas individuals can. I know people who would call this mob mentality, but I won't.

This is, of course, meant to be humorous. Tell us if you do or don’t find it funny and why, but not if you are mad enough to take it seriously.

In an environment like this, it is relatively easy to introduce practices that encourage more defects. For instance, you could ask everyone to comment their code to excess, in order to prevent misunderstandings. When the comments start to take over from code, the error rate will shoot up, too. The only way to prevent these defects is to improve the quantity of comments, of course. And so on.

The ideal defects are those that cause most fingers to point, but leave the real source of the problem unidentified. That way, everyone has the opportunity to blame somebody else and also the opportunity to go paranoid with the thought that the balance of power may change any time, leaving all fingers pointing at them.

At this point, the quality management team can become the defect prevention team, too. But having a central team on the job is a little too draconian, and not very efficient. So, there should be a defect prevention commissar (well, call them tsars, if you want; this seems to be more socially acceptable these days) in every team. In fact, let's have a defect prevention tsar for the company, who assigns commissars in every team. How much more Kafkaesque can we get? The team members should take turns to sit in the commissar's seat, in order, apparently, to appreciate the importance of quality and to share the heavy burden. The real reason, of course, is to give everyone a chance to be hated by everyone else in the team.

Identifying the defects is not enough. We should also revise our process every time we discover defects. That's our excuse to make the process even bigger and more complex, but who cares?

Our final technique is the introduction of measurements to defect prevention activities. If defect prevention is good, the more of it that teams do, the better. Measure how much defect prevention teams are doing and reward them for this activity. Soon, everybody will be busy with defect prevention, defect hiding and defect metrics collection. They may not have time to do anything else, but they will be very busy and, on paper at least, very successful.

This is the end of the journey, where we delivered the whole organisation onto the hamster wheel of continuous process

This is, of course, meant to be humorous. Tell us if you do or don’t find it funny and why, but not if you are mad enough to take it seriously.

improvement and defect prevention. Now, I know hamsters make lousy pets because they sleep all the time, they get naughty and bite your finger if you pet them too much and they don't live long. But look at their faces when they are running on that wheel. They could almost be happy. And if they don't like it, why are they on that wheel all the time?[8] So, how could you deny your organisation that once in a lifetime opportunity to get on the wheel of change?[9]

Finally

Yes, there you have it.
You thought it would be easy, didn't you?
In fact, it's a lot more work to be a successful failure than most would believe.
Should that stop us? Of course not!

This is the bit where you are probably expecting, "Aah, but there is a better way. If you do this and that, your processes will flow smoothly and you will achieve CMM Level 17." Be prepared to be disappointed – or should that be relieved? Because it's not going to happen. Even if it weren't just too corny an ending (which it is), it still would have been wasted effort because just about everybody who is anybody has already written a book or six about it.

Instead, you might like to consider this.
Suppose you managed to take an organisation to CMM Level 0. What next? Well, you've lied, you've cheated, you've manipulated large numbers of people as if they were mere pawns in some game, you have no conscience, feel no remorse – and, since you haven't been hung, drawn and quartered for your efforts, you've turned out to be very good at all this, unlike the majority of the population. Tell us: Have you ever considered a career in politics?

---

[8] Maybe because they aren't smart enough to get off it once it gets going. And they get back on it in the first place, because they are too stupid to remember what happened the last time they did that.

[9] Don’t worry. They will bite. Because they don’t remember what happened last time they took the once in a lifetime opportunity to get on the wheel of change.

This is, of course, meant to be humorous. Tell us if you do or don’t find it funny and why, but not if you are mad enough to take it seriously.

Acknowledgement

It turns out that there is a similar paper entitled "The Capability Im-Maturity Model (CIMM)" by a Capt. Tom Schorsch of U.S. Air Force, which has been around for a few years (the paper, that is. The USAF must have been around longer than a few years, I'm sure.). That paper was in turn based on Finkelstein's "A Software Process Immaturity Model", from way back in '92.

This paper (if you can call it a paper), on the other hand, is based on nothing but the firm conviction that life can get a lot nastier than it already is if we carry on believing that processes are about bits of paper and not about people.

This is, of course, meant to be humorous. Tell us if you do or don't find it funny and why, but not if you are mad enough to take it seriously.