

Changing Culture:

Should you? Could you? How? Why?

D R A F T

C. C. Shelley

June 2009

Oxford Software Engineering
9 Spinners Court, 53 West End, Witney, Oxfordshire, England, OX28 1NH
shelley@osel.netkonect.co.uk

ABSTRACT

The evolution of software development cultures is briefly described. A simple method for characterizing and analysing an organization's software culture, within the context of its host organization, and with respect to the products or services it delivers is described, and proposed as a prerequisite to any attempt to change a software culture. It is suggested that culture change is risky and potentially damaging and should not be attempted - but if it is to be undertaken there is only one rationale. Finally, a set of tools that can assist in cultural change are then identified and briefly described.

Introduction

All software is developed within a software development culture. The culture¹ exerts an influence on development practices and the software itself. It is thought that by changing² the culture the development activity and the software itself will, in some way, be improved. However it is not always clear what 'improved' means. And attempting to change a software culture has an uncertain outcome:

"It should be borne in mind that there is nothing more difficult to handle, more doubtful of success, and more dangerous to carry through than initiating changes in a state's constitution. The innovator makes enemies of all those who prospered under the old order, and only lukewarm support is forthcoming from those that would prosper under the new. Their support is

¹ Culture = 'norms and values' or 'behaviour and expectations'

² Assumed to be for the better by those responsible for initiating the change, although this is not always a view shared by those affected.

lukewarm partly from fear of their adversaries, who have the existing laws on their side, and partly because men are generally incredulous, never really trusting new things unless they have tested them by experience. In consequence, whenever those who oppose the changes can do so they attack vigorously, and the defence made by the others only lukewarm. So both the innovator and his friends come to grief"³.

Machiavelli: The Prince, Part VI

Software Cultures

Software has been developed for more than fifty years. Over that time software development cultures have evolved and diversified. Early programmers were a part of a small computing elite of scientists and engineers working in a 'white coat' culture. This soon expanded and split into a software engineering culture – mainly within the computer manufacturers and producers of embedded systems - and a DP (data processing) culture - now IT - within the commercial computer using organizations (the scientific dimension being marginalized to computer science, actually a branch of applied mathematics; and not to be confused with programming done by scientists – mostly modelling and simulations).

³ He then follows with some advice: *...But to discuss this subject thoroughly we must distinguish between innovators who stand alone and those who depend on others, that is between those who to achieve their purposes can force the issue and those who must use persuasion. In the second case they always come to grief, having achieved nothing; when, however, they depend on their own resources, then they are seldom endangered.*

<

Along with these two principal strands a hacker sub culture has developed, exemplified by the Unix (and now Linux) community and the open source and free software communities. Members of this sect can be identified by their delight in recursion and recursive, or subversive names: e.g. GNU, PHP, Subversion (a CM tool!)

The development of cheaper distributed computing enabled by mini, and later micro computers, and computer networks, has diluted or marginalized the engineering culture and has disseminated wider, heterogeneous, 'de-professionalized' *developer* cultures.

Describing Cultures

When discussing the cultures within which software is developed the scope should be identified before it is described. Specifically, are we interested in the culture of the developers (and/or analysts and testers) themselves, the culture of the teams they work in or the host department or organization? Each is liable to influence the other in some way and the degree of alignment between them may be important.

The culture of the host organization will influence the nature of the software development culture from two perspectives – firstly because the developers are immersed in the culture of the host, and secondly because the organization's requirements of the software being developed will influence the expectations and behaviour of the developers.

What sort of culture would you expect in a computer manufacturer, a bank, an insurance company, a defence contractor, a pharmaceutical company or a telecoms company? These organizations function in completely different ways and with different norms and values. Developers in these organizations will be immersed in this culture. What is the 'best' (most appropriate) culture for these developers? Should it be aligned with the culture of the host organization, or the (non functional, performance) requirements of the software being produced or maintained, or both, or something else altogether?

To deal with this some way of describing, communicating and reasoning about a culture; its scope, current status and desired status is required.

Ways of describing cultures in general terms are well known. They usually involve some form of scale or matrix with characteristics describing different types of organization. Constantine describes four organizational paradigms: Closed (which is stable, secure, less flexible, top down decision making, less innovative, and accepts traditional authority), Open (innovative, collaborative, sharing, negotiating, adaptive, consensual and problem solving), Synchronous (harmonious, aligned, quietly efficient,

slow to change, effortless co-ordination, smooth operation, visionary leadership), and Random (independent initiative, creatively inventive, individualistic, less stable, autonomous, informal freewheeling). DeGrace and Stahl describe two different software cultures; the Greek (less formal, small scale programs, OO methods, under managed projects, minimal documentation and standards to enable uniformity), and Roman (more formal, structured methods, large programs, CASE tools, strictly managed projects, large quantities of documentation and standards used to induce conformity).

These taxonomies are useful for characterizing a software culture but more is needed to provide a tool to understand a culture for someone attempting to change it. Such a tool should enable a culture to be described and communicated (reasonably) clearly, completely and objectively (that is, not imply values) *before* change is attempted. (It would appear reasonable to understand what is to be changed, what the 'baseline' is, before it is changed.)

The following simple two step process for developing an understanding and mapping of a software culture is proposed:

Step 1: For the culture within a particular context a list of relevant attributes or characteristics is built where 'relevant' means they are either known to be present, or of interest but absent, or observed in similar situations, or desired, or to be avoided. Characteristics may be described by a single word, for example, 'consensual' or 'ambitious', or 'siloes', or a simple phrase or sentence, for example 'good internal communications', or 'limited staff turnover'. While these words or statements should be reasonably clear they need not be definitive. They are intended to suggest characteristics of interest or value and may need exploration and interpretation.'

The attributes can be structured to enable them to be reasoned about. A simple hierarchy: *values > attitudes > behaviour* helps to provide an organizing principle to help reason about the relationships between the attributes. So a particular value (social attributes) may lead to one or more attitudes that in turn lead to or explain several behaviours (in general, technical attributes):

```
value1
  attitude1
    behaviour1
    behaviour2
    behaviour3
  attitude2
    behaviour4
```

...
etc.

So values or attitudes can be used to predict behaviours, and behaviours can be considered (perhaps) as symptoms of attitudes and values. So, for example: “This organization believes <a> which suggests that behaviour will be performed, or if introduced may be found useful and valued”. Or alternatively : “ ...which suggests that behaviour <c> will not be found and if we attempt to introduce it will be rejected.”

(Note: while it is natural to consider this relationship one directional; that values inform attitudes that direct behaviours this may not always so. On a occasion strenuous and persistent efforts to change behaviours may modify attitudes that influence values – suggesting that bottom up approaches to changing behaviour, e.g. training, *may* provide a means of cultural change. Also, use a structured list with caution. The construction of the structured list may be *wrong*, i.e. an attitude presumed to lead to a behaviour may *not* – it only appears to.)

This structuring may also be used to identify consistency of values, attitudes and behaviour (or inconsistencies) of the culture 'as is' or 'to be'.

It is important that, so far as possible, the list of attributes have no value suggested or implied, no suggestion of 'better' or 'best', and certainly no scoring should be possible. The intent is a cultural 'mapping' tool, not a means of evaluating cultural 'goodness'.

This process of building a list is as important as the list itself. It is an opportunity to explore the culture. Standard, 'off the shelf' lists should not be used, but may be used to seed ideas (see attributes and cues list⁴).

The list building is then repeated for the wider context within which the culture is being evaluated. So, for instance, when a software culture is to be characterized then so will the host organization's culture, within which it exists. Or if individual software developer attributes are to be characterized then so will the teams within which they work.

Similarly the characteristics of the cultures within, or properties of software created by the culture being evaluated will be characterized. So for instance if the team culture is being evaluated then the software being produced or maintained by that team should also be evaluated (of course software should already be characterized by its 'ilities', or non functional or supplementary requirements).

In this way the culture being described is bracketed by its surrounding context, and the context it itself surrounds.

(No attempt is made to ensure consistency or traceability of attributes between lists. While this may have some value the constraints and overheads this may impose are thought to outweigh the value. And the establishment of consistent sets of attributes may also lead to standard lists and the beginnings of a scoring system, which is not wanted.)

Step 2: The culture is then examined 'as is' and each of the listed attributes marked as either 'present' (more or less uniformly) or 'not present', or 'mixed'. Each of the attributes a can commented too, and is recommended as it can give a richer picture of the culture. This is repeated for the contextual culture and the enclosed culture or software attributes. The lists are being used as checklists to provide a set of nested frameworks for describing the cultures. They are best completed by several individuals working independently then reviewing their results - similar to the Delphi process. If additional attributes are identified during this process these are added to the list.

When completed the lists describe the culture, its context, and cultures it hosts or the systems it is responsible for. These can be compared and analysed for alignments, conflicts and to identify areas to be addressed to enable the culture to be modified to better meet the host organizations need *by producing software and systems that better meet the host organization's needs* – and not simply align a culture to that of its context (e.g. align a software development culture to that of the host organization). These lists can then be used as the starting point for developing a set of cultural 'to be' lists.

As an optional third step the checked lists can be used as a baseline and developed (rechecked) to describe the desired 'to be' culture

(And there may be alternative uses of the list building and checking process. Rather than building and completing a set for 'as is' and 'to be', lists can be built for comparison, for example building and checking lists for 'as is', and 'as they are' (and we'd like to be), or 'as is' and 'the way we were' (when we were really good).)

Changing Cultures: Why?

It is well known that “organizations get the software they deserve” and “software = team” (attributes) so why attempt to change a culture? Software cultures emerge unconsciously. Why not leave well alone and allow them to evolve naturally?

⁴The author's personal list

It would seem clear that to change a culture requires the destruction of some elements that were at least understood, perhaps accepted and maybe valued, by those immersed in it and, especially those successful and prospering in it. And what ever the change, experiencing it is, for many, at best uncomfortable and at worst unacceptable as uncertainly, and doubt disrupt and displace customary norms and values.

There has to be a very good reason for doing it. It has been assumed here that the objective of describing a software culture is a prerequisite to enable and direct changes to that culture. However any consciously undertaken change presumes a change for the better; an improvement of some sort. But this may not, perhaps, is usually not, what happens. Mere *technical* changes to working practices, say the introduction of a new method or tool are problematic enough even when they are aligned with the prevailing culture and/or the technical requirements of development. Changing the culture itself is at best difficult, and with ideas of what is good or bad directed only by personal experience and preference rather than a real understanding of the organization's context, history and character it is important to think carefully about even attempting cultural change. Mapping a culture may provide some understanding, and that may indicate that change is too uncertain, or is certain to fail.

The only acceptable reasons to undertake cultural change are, firstly, because the current culture is clearly inhibiting or damaging software development and it is necessary to replace or realign it *so that software developers and others are better able to produce and maintain software systems that better meet the organization's needs*, whether the software is for use by the host organization, or is a product of the host organization, and, secondly, it is cost effective and feasible to do so.

The first reason should be obvious, the second is not. Many attempts to change cultures are destined to fail because of poor understanding of the context and the robustness of the prevailing culture, and the difficulties of cultural change.

The best advice when knowingly⁵ considering undertaking cultural change is 'Don't'.

Changing Cultures: How?

Assuming the advice above is ignored there are a number of tools for understanding culture and inducing change. They tend to be either quite direct, even brutal – e.g. remove the individuals or disbanding the teams that do not embody the required cultural norms, or rather indirect, seeking to influence a culture at one remove by undertaking technical activities and seeking some second order effects. (This second order effect also implies that these methods can be inefficient and expensive, with sometimes considerable wastage.)

These tools are diverse and used in widely differing contexts. They are probably best described as patterns (after Alexander, 1977) which are basically statements of recurring problems and then a statement of the core of the solution “...in such a way that you can use the solution a million times over without ever doing it the same way twice.” - very non procedural and unmethodical; ideally suited to solving rather difficult to describe problems.

Patterns and Antipatterns

This section lists some of the patterns (or pattern sets) and antipatterns (a solution that doesn't produce the outcomes expected). That can be used to understand cultures and induce cultural change. Each item is classified as either a patterns (P) or antipattern (A), or both (P/A) where it can cause problems. And is classified as a pattern for providing understanding (U), or change (C), or both (U/C).

1. *The cultural mapping tool described above* (P, U). Used discreetly⁶ provides a picture of current and required cultures and their contexts.

⁵ Cultural change is often, perhaps usually, performed unconsciously by those that by simply expressing and acting out their beliefs and values establish a software culture. Whether or not this is the best culture for a given context is not always clear as those establishing the culture are not aware of what they are doing or how it aligns to the delivery of software or systems that best meet the host organization's needs. The author has asked managers and developers in many organizations, both large and small, why things are the way they are – why practices, models, assumptions and beliefs are set in a particular mould. The usually response is that that is the way they have always been. Very rarely is a consciously and competently modelled software culture encountered.

⁶ Individuals are often very wary of methods or tools being used on them, and are ingenious in removing themselves as objects of study or distorting the method or the information it delivers.

2. *Senior Management* (P, C). Senior management (and other influential people) are an excellent SPI tool and an excellent aid to cultural change too. But they can be unaware of this. Use them to exert influence. Often all that is required is for them to say what they want. Provide them with objectives and means to monitor, and be seen to monitor, progress to objectives. Appropriate reward systems can help too - but these are difficult to get right.

3. *Remove leaders and influencers who do not embody desired cultural attributes* (P, C). Remove those that do not fit the required culture, and introduce or promote those that do. Make sure the reason for the change is known by others.

4. *Disband teams who do not embody desired cultural attributes* (P, C). As for 3. Teams are important; the units of production for software. Remove those that exhibit the wrong cultural behaviours and build teams with terms of reference and individuals aligned to the required culture.

5. *Reorganization* (P, C). Necessarily done by new leaders a reorganization can also provides the uncertainty (and occasionally, chaos) to enable a change of culture to take place. (Although the 'ghost' of the old organization may continues to operate too.) Organizational structures (hierarchical, flat, network, star etc. may also suggest or enable certain values.

6. *CMM & CBA IPI* (P, U) CMM provided a valuable framework for understanding organizations. Nominal organizational 'levels' actually characterize organizational types. The assessment process is a powerful diagnostic tool enabling the ways of working to be examined and improvements proposed. When used well it also provides an impetus for change.

7. *CMM Common Features* (P, C). The CMM's 'common features' are a set of artefacts and activities that direct and maintain an organizations ways of working. These 'institutionalization' tools are divided into three categories: commitment, ability, measurement and analysis, and verification. These tools are have been largely lost in the more recent versions of CMMI.

8. *Mimicry* (P/A, C). Mimicry is used widely used means of transmitting culture. This can work well in software development as a tool for cultural change where the change is either technical or somewhat limited. It requires a leap of faith - 'just do this and see what happen' - many repetitions (trial and error), and an ability to evaluate the results. This has been described by Parnas:

"The bad news is that , in our opinion, we will never find the philosopher's stone. We will never find a process that allows us to design software in a perfectly rational way. The good news is that we can fake it. ... " (Ref 1. Parnas & Clements).

However it can also lead to sterile and damaging imitation if the changes required are subtle, extensive or are in conflict with the host organizations culture. Problems will also occur the approach to change does not lend itself to trial and error, or ongoing learning. The wholesale adoption of 'agile' practices and frameworks such as CMMI are especially prone to this. This 'cargo cult'⁷ imitation is widespread in the software community.

9. *Education and Training*: (P, C&U). While training is listed here for its obvious role in changing behaviour, and 'education' has a role in changing attitudes too, but that is not the primary reason they are listed here. Rather they can act as a diagnostic tool and a catalyst for change. When large numbers of people are to be trained it is an opportunity for the trainer to find out about the working practices, and culture of those being trained. A dialog, based, around the topic being taught, provides an excellent learning opportunity to find out why things are the way they are. It is notable that those working within a particular organization and culture often have little idea of the reasons they work the way they do.

In some instances training can act as a powerful catalyst, validating a practice already well known by those being taught, but not used because it is not habitual in the organization. The training signals that the practice is acceptable. With this signal, the activity can now take place - it is now acceptable to expend time and effort performing it.

10. *Skeptics* (P, U&C). When attempting change skeptics will emerge, either as vociferous critics or quietly observing from the sidelines. These people are worth seeking out and attempting to engage with. Their experience, beliefs and opinions are often valuable and should be taken

⁷ "...In the South Seas there is a cargo cult of people. During the war they saw airplanes with lots of good materials, and they want the same thing to happen now. So they've arranged to make things like runways, to put fires along the sides of the runways, to make a wooden hut for a man to sit in, with two wooden pieces on his head for headphones and bars of bamboo sticking out like antennas- he's the controller- and they wait for the airplanes to land. They're doing everything right. The form is perfect. It looks exactly the way it looked before. But it doesn't work. No airplanes land. So I call these things cargo cult science, because they follow all the apparent precepts and forms of scientific investigation, but they're missing something essential, because the planes don't land."

- Surely You're Joking Mr. Feynman?, Richard Feynman

seriously. They may be privy to information that change leaders are not, and may be right in their skepticism. Try to work with them to explain and justify proposed changes, accept modifications to the changes and reach consensus on a way forward.

11. *Kotter's Eight Stage Change Process* (P, C). This process was developed to remedy eight 'errors' when making change. In particular step one 'establishing a sense of urgency' (or precipitating a crisis) is often important if the desire to change is to be taken seriously. Without it day to day operational matters dominate people's thinking. The remaining steps: *create a guiding coalition, developing a vision and strategy, communication the change vision, empowering broad-based action, generating short term wins, consolidating gains and producing more change, anchoring new approaches into the culture* are about developing sharing and sustaining the changes.

12. *Performance Targets* (A/p⁸, C) These can work very well indeed, but usually cause damage. HP's '10X' performance targets for its software development (requiring a carefully defined ten fold improvement in quality over five years) was a notably successful programme of improvement driven by performance targets. But this is the exception. Poorly thought out performance targets will lead to dysfunctional behaviours, undermining development activity and hiding real performance. Typical dysfunctional performance measures are the achievement of, say, CMMI ML3 by a given date, or year on year productivity improvements. Or both together – several instances of both these targets being inflicted on software developers a have been observed by the author.

13. *RPI* (P, C). Rapid Process Improvement (Ref 2. RPI) is an approach, supported by a set of tools for making change in organizations. Most of these changes are technical in nature. RPI is a form of SPI (Software Process Improvement) which has a high technical content, but as software development is people intensive there is often a cultural element to it too: aligning or modifying technical practices to the prevailing culture, 'selling' technical practices to those unfamiliar with them, etc. The RPI approach has a number of characteristics: it is focussed on problem solving, not unspecified 'improvement', desired results are specified and measurable, it is driven by results and learning from them, which are delivered by ongoing small increments of work, work is time-boxed, work may use models or frameworks but not driven by them, work is inclusive and integrated into day to day work,

work is structured by the use of methods and tools, it supports local ownership and accountability, and is opportunistic. Many of these characteristics were present in SPI, but have been largely lost with the rise and maturing of comprehensive software development frameworks. Tools include methods for exploring and defining processes, problem solving, fixing process problems, performing post implementation reviews (the tools have been developed and acquired and are not necessarily original – just useful), defining measures, etc. Many are procedural which makes them potentially easy to understand, but appear to be unfashionable. They may be recast as 'patterns' (or have their procedural origins obscured) to make them more acceptable to contemporary software organizations and teams.

14. *Hawthorne Effect* (P, C). The Hawthorne effect is usually considered as an oddity and a confounding factor when trying to understand how to improve ways of working. Consider using the Hawthorne effect as a tool for change in its own right. Close observation and genuine interest in ways of working, with a real concern with improvement and perhaps nominal changes or placebos can trigger disproportionate improvement, steered by observation of the results and careful selection of the 'treatments'.

(While not strictly a tool for cultural change the author has observed 'mixed cultures' that have produced striking results. A mix of military and academic staff in one organization produced an odd, and occasionally comical environment but also a formidable cultural 'alloy' with an impressive capability. But such cultural alchemy is expensive, time consuming and risky, and unlikely to be undertaken by any but the largest or most ambitious or determined organizations.)

Closing Remarks

Software development cultures have developed over time to accommodate the industries and professions within which they exist, and the performance requirements required of the products or services they deliver. Changing these cultures may be unwise. It is difficult and risky with uncertain outcomes. However if it is to be undertaken then the culture before and after change should be well understood. And it should only ever be undertaken if it is known to be feasible and the new culture will provide better products or services to the host organization – where 'better' is understood explicitly and specifically and measurably.

⁸Listed as a small *p* because performance targets are usually poorly formulated and cause damage.

“How easily men could make things much better than they are, if they only tried together.”

Churchill to Clementine - 1909

References

- 1 David Lorge Parnas, Paul C. Clements, A Rational Design Process: How and Why to Fake It, IEEE Transactions on Software Engineering, Vol SE-12. No 2, February 1986.
- 2 RPI: The Rapid Process Improvement tool set, www.osel.co.uk/rpi/rpi.htm