The Wild-West Revisited

David R Pitts
Honeywell Building Systems Center Europe
Dave.Pitts@GERMANY.honeywell.com
Barbara H Miller
Honeywell IAC

Introduction

In 1985 Mark Spinrad and Curt Abraham published; "*The Wild West Lifecycle (WILI)*" [1] which, although written with a slightly irreverent tongue-in-cheek style, introduced a significant metaphor for the software engineering lifecycle. In this paper we will discuss briefly the importance of software metaphors in general, and then revisit and expand the Wild West analogy.

Software metaphors

Steve McConnell [2] in his excellent book gives a very good description of the importance of metaphors with respect to software engineering, and although he makes reference to the WILI model he does not expand upon it. The importance of the metaphor must not be underestimated. Dictionaries give a number of different definitions of the word metaphor. This in itself reflects an underlying problem of science, and ultimately the biggest problem of the use of the metaphor (but more on that later). The definition we prefer was found in a small pocket dictionary [3] and is: "Way of describing something by suggesting it has the properties of something else." When we wish to describe a new concept for the understanding of an individual, we more often than not use analogy (or, probably more correctly, metaphor).

Metaphors have a number of uses. A metaphor can be used to explain to someone something that could be part of their experience, but which they may have not yet experienced. A metaphor can be used to theorize about processes not possibly within the experience of any person; for example treatises on multi-dimensions make analogy to the three spatial dimensions of our experience and extrapolate from three to many dimensions. A metaphor may be used to simplify complex processes by making them analogous to simplistic process.

Analogy is a two edged sword, however, not only because of the emotions that it evokes in the recipients, but also because our ability to understand the allegory (or analogous object) tempts us to draw inappropriate conclusions as to our ability to understand the alluded object.

Just as there are many dictionaries with different definitions of the word "metaphor" (just how different can be observed from this alternative definition; "figure of speech in which word is used to denote something different from its usual meaning" [4]), there are many different metaphors being advocated for the process of software engineering. All of these metaphors are molded by the authors' own intolerances and propensities. Most of us, since we are educated in the 20th Century Sciences, rather than the earlier more holistic philosophy of Newton and Galileo, find ourselves naturally asking the question: "Which is the correct definition?" The answer to this question is, of course, "All of them." The search for and the belief in a Holy Grail of single truth is perhaps one of the most fallacious developments of the age of reason. As in the legend of the search for the Holy Grail, it is not the object itself but the process of the search which is the most important. Even when models have been replaced by better models, the former may have their uses. Newtonian dynamics is still adequate enough to land a man on the moon, even though it has generally been superseded by Einsteinian theory, which itself is perhaps in the process of being superseded. Whereas Newtonian dynamics and Ensteinian theory are substantially more than metaphors, the same holds true that within a certain level of scrutiny all metaphors and models stand in their own right. The degree of scrutiny may always be increased to the point that the metaphor/model fails. As yet no-single model has been substantiated to the extent and exclusion of all other possible models. We believe, however, that if a person has, for example, the whole of the Newtonian Dynamics at his finger tips, he can justly be said to understand more about the universe than a person with a superficial or popular knowledge of Einstein's theories. In the same way it can be said that software engineers who adopt a metaphor for software development, and apply it with success to their own work, have a better understanding of their vocation than those who do not. To quote McConnell: "Over time, the person who uses metaphors to illuminate the software development process will be perceived as someone who has a better understanding of programming and produces better code faster than people who don t use them."

So, where does that lead us? To a metaphor:

The Wild West Lifecycle Model

The Wild West lifecycle model makes the software development process analogous to the process of territorial conquest in the American west. Three different types of people are identified has having participated in different phases of this process. These are quoting directly from Spinrad [1]

Trailblazers carved out paths and brought back knowledge about the dangers that lay ahead.

Pioneers followed in their paths and made the destination safe for those who followed.

Settlers came by caravan to establish residence in the destination.

Spinrad does not identify the phases that these individuals participated in by name, but it might be helpful to describe them as the Exploration Phase, the Colonization Phase, and the Civilization Phase respectively. The original paper suggests that phases analogous to these exist in the software development lifecycle, and that the skills required by the software engineers during these phase are those skills exhibited by the trailblazers, pioneers, and settlers in the Wild West A further aspect of the analogy that may prove illuminating is the amount of external direction needed.

Trailblazers, pioneers, and settlers required a different amount of outside direction to do their jobs well. In the same manner, software trailblazers, software pioneers, and software settlers ought to have differing amounts and kinds of outside direction.

Trailblazers had only the barest of outside direction, and some worked with none at all; others set their own. An overall view of the ultimate goal was usually all they had. They had no maps, no exact arrival times, no intermediate check-in points. They vanished into the wilderness for years at a time, some never to be seen again. Software trailblazers should have the same kind of freedoms. They do not need to follow formal procedures, because these are too focused and too restrictive. They probably don't need naming rules or coding conventions, because it is not certain just exactly what form the code they write or the paths they create will take. As long as the activities are those of the trailblazer, it is too early to set up intermediate goals. Having been given the same freedoms, however, they stand the same risk of suffering an untimely and unknown end as their historical alter-egos.

The historical trailblazers were normally loners, who relied not a little on luck, but also on their own ability and individual skills to survive the environment. They needed to have consummate knowledge of the wilderness, to be able to "read" the situation to avoid dangers wherever possible, and where this was not possible, take the appropriate action. In the same way software trailblazers are usually individualistic, but in addition they need to be skilled in the theoretical aspects of software engineering, and practical enough to survive the catastrophes that occur. They need to know what the theories are and how and when to apply them to a particular situation.

Pioneers used the rough map set up by the trailblazers. Based on what the trailblazers shared, pioneers could set rough timetables. They knew where the significant mileposts were. Even though they probably did not follow the exact path of the trailblazers, they could base their progress on the directions, however incomplete and crude, of the trailblazers. Software pioneers need guidance in setting up their own rules and conventions. The path has been blazed; it may need some smoothing out The milestones are known, even though unexpected pitfalls may arise along the way. The Wild West pioneers were moving from a higher level of civilization to a lower level. As such they could take a lot of existing material with them, be that in terms of provisions or philosophies. In addition. pioneers were generally like-minded groups of individuals. These two facts – that they are a group of individuals, and they have support from and existing civilization – mean that the pioneers do not have to rely so heavily on individuals being multi-skilled. The skill set can be provided by the group, not the individuals.

It should be noted, however, that many of the historical pioneering exploits were driven by a desire to establish a different code of ethics or religion, or to escape from a regime that was deemed to be oppressive. In some instances this is also true for the software pioneers, and judgment should be made as to what the driving force behind the exodus is – a desire to escape or develop.

Settlers had a map that was drawn with all the care and expertise available to the pioneers. The pioneers attempted to catalog not only the major mileposts, but also all of the intermediate stopping places. As far as they could, the pioneers took care of the dangerous places in the passage, either by eliminating the danger, by posting sentries, or, if the danger could not be eliminated, providing careful instructions as to how to behave and thus make it through. Software settlers should have a Pert chart or timeline, with all major and most minor mileposts marked. If the pioneers have done their work well, settlers will find very few unexpected dangers along the implementation road. The software pioneers will either remove the danger (maybe by suggesting a different tack in the coding); post sentries (in-line comments, warnings about critical interactions, etc.); or provide instruction as to what precautions the settlers should take at the critical points. Thus, the settlers can set for themselves a dependable schedule, with the confidence that everything has been done to ensure that they can meet the schedule. The drive for settlers is different from that of pioneers, in as much as they desire to take advantage of tangible benefits that the new territory has to offer, and are less ideologically driven.

Spinrad identified only the three classifications of individuals discussed so far. Of course the Wild West metaphor is filled with many more possible allusions than those. No mention has been made of significant role models such as Cowboy, the Native American, the US Cavalry, the Railroad Company, etc. Indeed, what about the individual legends such as "Buffalo Bill," "General Custer," and "Billy the Kid?" Although it may well be argued that these are sub-classes of the original metaphor, some further insight could be gained from exploring these characters in the metaphor further.

Before we can pursue the metaphor much further and discuss the software equivalent to "Custer's Last Stand" or "The Battle of the Alamo" (remember that?), or draw a parallel between Bill Gates and a historical counterpart, we must sidetrack slightly here, the reason being that we must define our base of reference. When we are discussing the Wild West, are we referring to the factual account, or the Hollywood interpretation? Dependent upon your base of reference, a cowboy is a champion of the ordinary folk, with an innate understanding of morality and a high sense of justice. Highly skilled with a gun, he is usually to be seen wearing a white hat and maybe singing to his horse. Or he is an ignorant dirty itinerant, unreliable and untrustworthy[10]. In fact there were probably never more than 10,000 cowboys [5] in total throughout the whole of the period known as the Wild West. It is a sobering fact that there have probably been more actors and stunt men who have played cowboys than there ever were cowboys. Thanks to Hollywood however, the Wild-West metaphor is recognized

[10] It is interesting to note that an informal definition of Cowboy in an English (as opposed to an American English) dictionary is "Bad workman who charges too much." Yet another indication of the power of the Wild West metaphor to apply to software engineering.

and understood worldwide, but the Hollywood interpretation only[11].

This in itself has a striking parallel with software engineering, around which there is a large popular mythology as large as that surrounding the Wild West, and just as with the Wild West mythology some worthy principles can be conveyed, so also many dangerous messages can be imparted. So which to use? As with the many dictionary definitions of "metaphor," the word, many interpretation of the Wild West are also valid. We suggest here that initially the Hollywood interpretation be used, since it is easily available, universally recognized, and widely accepted. As the familiarity with the metaphor and the software lifecycle increases, much more insight may be gained by beginning to explore the more factual histories of the period.

Who are the software Native Americans? This is difficult, but let us submit that a good analogy would be those individuals that were there even before Software Engineering existed as a concept – the self-taught garage programmers. Is ISO9000 the equivalent of the US Cavalry, bringing law and order to the troubled frontier? And who are the cowboys? Considering the definition given earlier, perhaps we all are. Where was "Custer's Last Stand?" Maybe it is yet to be fought. A couple of our experiences could come close, but that is another story. Which historical figure is Bill Gates? It is left for the reader to decide.

In conclusion, consider one more facet of the history of the Wild West, and the characters described here. The history was written against the fact that the Wild West was moving from a chaotic unknown state to a civilized developed state. As Spinrad points out, it was the participation of the people that defined these phases, but in addition they drove the process. The transition boundaries were times of great social upheaval and ultimately the people that made each step possible were no longer needed or indeed desirable commodities in the world that they created. Only if the individuals were able to adapt and change, to go from trailblazer to pioneer to settler was there a continued role for them to play. Otherwise they became an anachronism.

It remains to be seen whether the Wild West metaphor holds, and if the above scenario is the case with Software Engineering in general. But even if there is an infinite territory to conquer, the process of Software Engineering maturity is on going within industry, and unless individual software engineers are lucky enough to be employed by a company large enough (or enlightened enough) to support all the phases of the model, they will more than likely encounter the above social issues. The question is now, as it was then, do you up stakes and move on, or stay and make the best of it?

[11] Spinrad himself gave a clue to the fact that he had in mind the popular image of the Wild West when he alluded to pioneers "circling their wagons," a practice that was invented by the movie industry, with no historical precedent.

References

1. Spinrad, Mark, and Curt Abraham. 1985. The Wild-West Lifecycle (WILI). *ACM SIGSOFT Software Engineering Notes* 10.no.3 (July): 47-48.
2. McConnell, Steve. 1993. *Code Complete*, Washington: Microsoft Press: 7-11
3. Collin, P. H., 1988. *Harrap's Mini English Dictionary.* London: Harrup Ltd.
4. Caswell D.M., and R. Batchelor-Smith. 1980. *Hugo English Dictionary.* Bungay, Suffolk, England: Richard Clay (The Chaucer Press) Ltd.
5. Bryson, Bill. 1995. *Made in America*, London: Reed Consumer Books Ltd. Page 155 Honeywell

Structural Defects in Object-Oriented Programming

David Rine
Computer Science Department
George Mason University
Fairfax, VA 22030-4444
drine@cne.gmu.edu

THE SHARING OF OBJECTS STRUCTURAL DEFECTS

A recent study (Offutt, 1995) indicates that over fifty percent of common structural defects found in the testing of C++ code would never have occurred had the code been written in a safer programming language such as Ada83, having a very strong typing and type checking model. The purpose of this paper is to report on structural defects in object-oriented programming due to object sharing.

There are a number of structural errors common to object oriented programming (e.g., in C++) when objects are dynamically introduced by pointers. Examples are (1) use of a pointer before it is initialized, (2) reference of an object as a parent after it has polymorphically changed to a child, and (3) various memory resource errors. A fourth (4) structural error common to many object-oriented programming languages such as C++, Smalltalk and Eiffel occurs when an object o is shared by two owners x and y. This is often introduced when, first, o is created for owner x (x points to o) and, second, x is assigned to y. Sharing can, also, be carried out through parameter passing by procedure calls. o can be created for owner x, x can be, for example, 'read', and x can be 'write' to y. If the object-oriented language does not prevent such sharing and, also, does not support effective object garbage collection, then the later deallocation of o to x may leave the allocation of o to y in place as an undesirable side- effect. Furthermore, x and y may have different security attributes which may cause one of the owners to illegally leak information. When multiple ownership is allowed its use should be carefully policed.

PREVENTING THE SHARING DEFECT IN ADA.

It is easy to prevent this structural error from occurring