Focus Group: Metaphor in Software Development

Kevlin Henney (kevlin@curbralan.com)
Andy Longshaw (andy@blueskyline.com)

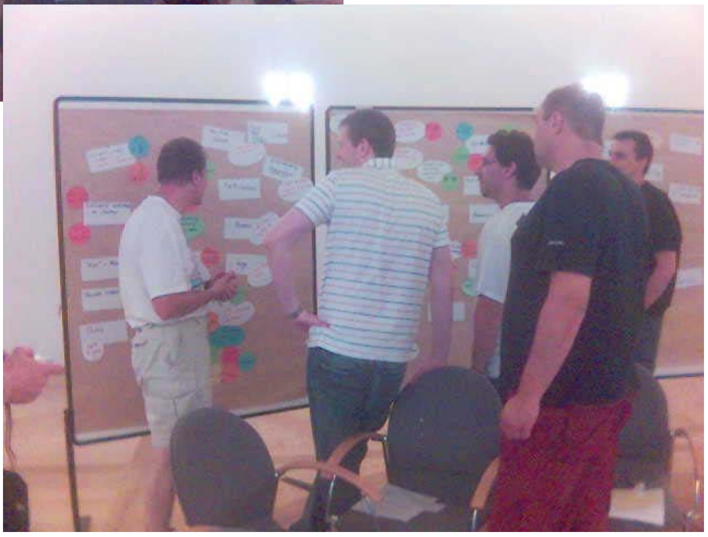
Metaphor appears in many aspects of software development [Henney2003] from architectural, such as classic XP's expression of architecture through a system metaphor [Beck2000], to metaphors for software development itself, such as building architecture or civil engineering. The primary purpose of metaphor is to assist the understanding of the audience, whether of the written word or in discussion:

> ***metaphor*** a figure of speech in which a word or phrase is applied to an object or action to which it is not literally applicable; a thing regarded as representative or symbolic of something else, especially something abstract.
>
> *The New Oxford Dictionary of English*

A powerful and well-chosen metaphor can speak a thousand words in terms of the insight it gives to the audience; a poor one can hide and confuse intent. Since software development involves communication of abstract and aphysical concepts, it is no surprise that metaphor is a frequently used tool, whether in naming code artefacts, explaining architecture, naming patterns [Buschmann+2007], characterising process or communicating user interface models.

The Focus Group

The focus group sought to explore the use of specific metaphors in software development as well as the general benefits or liabilities of using metaphor, and the properties of a successful metaphor. The focus group participants split into two groups to brainstorm different metaphors and look at their positive and negative entailments. The results were combined and the implications discussed.



Metaphors that Work

There were some metaphors that seemed self-contained and worked consistently. For example, *zombies* are dead processes that continue to consume system resources and may cause trouble. A *virus* is associated with infection and mutation, and is addressed using *anti-virus* software. Some metaphors are not employed in a manner that is completely consistent or obvious, but the inconsistencies are minor in comparison to their other benefits. For example, a *tree* has a root, branches and leaves, but in the world of software we find the root is higher than the leaves. The question of direction, however, shows up elsewhere and is not always used consistently. For example, in a *download*, which way is up and which way is down?

Metaphors may also work in spite of some potentially quite strong negative connotations. In the case of thinking about software development in *sprints* (as found in the Scrum process [Schwaber+2002] and in Open Source development) the implications are different for different audiences. In this case, sprint is a term that is likely to sell to managers more than to developers. Sprints are goal oriented and fast, but they are exhausting and are not intended to be repeated iteratively. Software development is more like distance running, with its connotations of sustainable pace and a long-term objective, than it is like a series of sprints. However, management is more likely to buy into sprinting than jogging!

Misfit Metaphors

Some metaphors simply don't appear to work. For example, in addition to automation, *software factory* [Greenfield+2004] suggests an inhuman, labour-intensive environment, a sweat shop, and a great deal of explanation of the metaphor is required to get across its intended application (as occurs in the book of the same title). This appears to be back to front: an effective metaphor should explain something; it is not the metaphor itself that should be attracting the detailed explanation!

It also emerged that some metaphors just shouldn't work because they fail to capture — or even contradict — the actual implications of the original concept. For example, a *port* is something that is associated with docking and undocking, loading and unloading, etc, and yet in software it is normally associated with connections and a different set of vocabulary (open and close, send and receive, etc). A *Demilitarised Zone* (DMZ) is a space that is owned by neither side and is used to prevent (or make more visible) attacks by either side. However, in configuring a DMZ [Schumacher+2006] for a corporate network this symmetry and disownership clearly does not apply, although the residual implication of some kind of buffer zone makes sense. In considering a *firewall* on a PC, its purpose is to keep intentional damage out rather than contain accidents. By contrast, preventing the spread of fire by keeping fire contained is the purpose of firewalls in buildings, where the metaphor springs from.

In some cases a term is adopted and may just stick. In other cases a term appears sexy and eye-catching — and that appears reason enough. There are also many cases where a basic misunderstanding leads to a misapplication of something as a metaphor, but the concept sticks. It was wryly noted that, based on a number of the examples identified, perhaps the security community is more prone to getting metaphors wrong than other groups!

Metaphors that Reflect Their Origins

Metaphors can also be very much of their time and place, capturing the style and thinking of a particular culture or era. For example, Apple Macs (1980s) have *trashcans* where Microsoft Windows (1990s) have *recycle bins*. Metaphors also come with an implied set of values. For example, *garbage collectors* collect items that are no longer in use, implying that anything that is unused is by definition waste. Sometimes a metaphor will transcend its origins or remain stable over paradigm shifts. For example, talking about programs *running* and *crashing* suggests an object in motion, typically a machine. This terminology has remained with us in spite of significant shifts in how programs are perceived by both programmers and users.

Conclusions

There was a general consensus in the focus group that metaphors could be highly expressive, but to actively seek them might be too forced or the result too contrived. It is not common that a single metaphor is comprehensive enough or maps faithfully enough to the problem or solution domain to fully capture a whole system. Although a metaphor may successfully capture a part or some aspect, coverage of most concepts is improved by explaining them through multiple metaphors [Lakoff+1980], i.e. working with mixed metaphors is the norm rather than the exception. This perhaps helps to explain the common failure of the XP system metaphor practice, which seeks a single metaphor to capture a system, and its omission from XP2 [Beck+2005], replaced by a stronger acknowledgement of the role of software architecture.

References

[Beck2000] Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.

[Beck+2005] Kent Beck with Cynthia Andres, *Extreme Programming Explained: Embrace Change*, 2nd edition, Addison-Wesley, 2005.

[Buschmann+2007] Frank Buschmann, Kevlin Henney and Douglas C Schmidt, *Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages*, Wiley, 2007.

[Greenfield+2004] Jack Greenfield and Keith Short, with Steve Cook and Stuart Kent, *Software Factories*, Wiley, 2004.

[Henney2003] Kevlin Henney, *Beyond Metaphor*, JAOO, 2003.

[Lakoff+1980] George Lakoff and Mark Johnson, *Metaphors We Live By*, University of Chicago Press, 1980.

[Schumacher+2006] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann and Peter Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*, Wiley, 2006.

[Schwaber+2002] Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum*, Prentice Hall, 2002.