Raising Healthy Software Systems

Stephen G. MacDonell, Diana Kirk and Laurie McLeod

SERL, Auckland University of Technology
Private Bag 92006, Auckland 1142, New Zealand
{smacdone, dkirk, laumcl88}@aut.ac.nz

Abstract

We elaborate on the analogy between humans and bespoke software systems and we use this analogy to inform an alternative perspective on the development and management of such systems.

1. INTRODUCTION AND MOTIVATION

The treatment of software development as an engineering endeavour has encouraged us to view and treat software as something we can manufacture. This has led naturally to the notion of software factories and ongoing efforts in component-based software engineering (SE). There is no doubt that such thinking has been useful and has enabled significant progress; it is our contention, however, that maintaining such a view exclusive of others will limit further progress. Current and future contexts for software development exhibit such scale and complexity that complementary views are needed. This is due to a variety of factors: the growing diversity of deployment environments; the increasingly always-on nature of software; the expected interoperability of systems. And the fact is that bespoke software does not generally exist in isolation – it almost always exists as part of a system and as part of a process or product solution.

In short, there is growing acknowledgement that:

- ‘simple’ commodity systems become packages; so those we build will be increasingly complex [1]
- complexity is already well beyond any individual’s ability to understand [2]
- complexity comes from ‘the edges’ – interfaces, interactions, interoperability [3,4]
- some of today’s software engineering methods are increasingly inadequate in this context [5]
- current approaches for managing complex projects are misdirected and outdated [6,7].

In light of this, the limited scope of SE to the software rather than the solution/service means that the future usefulness of this dominant paradigm alone could be questioned. Similarly, the effectiveness of contemporary project management methods to guide the development and ongoing management of complex software systems is no longer assured. So: are there metaphors from or analogies to other domains that could provide better insights and improved methods? In computing generally a growing body of research is being directed towards the metaphor *du jour* – nature- or biologically-inspired computing (e.g. [8]). While this certainly deserves attention, restricting ourselves to just this metaphor could also be severely limiting [9,10]. In our research then, we consider an alternative analogy – between *software systems and people*.

2. SOFTWARE SYSTEMS AND PEOPLE

We propose that each bespoke software system should be viewed as an evolving individual in a population, and that useful parallels can be drawn between our own development and management and that of software systems. Both individuals and systems evolve over time through life stages. As with people, the timing for these stages is uncertain and varies from system to system. Broadly speaking, however, the stages are:

Creation. An individual has general characteristics that adhere to evolutionary norms, for example, has two eyes (structural), arms attached at shoulders (organizational), empathy for others (societal). An individual's specific form and function (blue eyes) are heavily influenced by their parents via inheritance.

Bespoke software systems have general characteristics, for example, modules with interfaces, with specific form and function defined by the development team responsible for software creation.

Incubation. Ante-natal care in the form of advice to parents and preventative support are generally given to expectant parents. Advice relates to behaviors that will help ensure a good outcome (healthy baby), for example, 'no smoking', 'take folic acid'. Support involves monitoring the values of some key system indicators (for example, heartbeat) with a view to highlighting potential problems.

For software systems 'in embryo' i.e. under development, the development team is advised to behave according to industry best practice to ensure a good outcome. Key system indicators, for example, numbers of defects and effort, are monitored with a view to highlighting potential problems.

Childhood. During childhood, system behavior is characterized by adherence to rules, compliance, and generally predictable behavior, once taught. Children begin to interact with the rest of the world under the watchful eye of parents. Parents facilitate this interaction by providing information about, for example, likes and dislikes or sleeping habits. If the child is ill, a doctor is called to isolate the source of the problem and advise parents on treatment.

After development, a software system is used by others and instructions for use are generally provided. It also interacts with other software systems. If stakeholders have problems, the maintenance team is alerted and it is expected that they will isolate the source of the problem and provide a resolution.

Adolescence. During this phase, external influences can lead to significant change, erratic behavior and lack of predictability. The strength of symptoms depends on many factors and includes both factors inherent in the adolescent and those in the interaction between adolescent and environment.

Software systems often experience a period of rapid change as features are added or the environment in which the system exists changes in unexpected ways.

Adulthood. Both individuals and software systems may reach a level of expected maturity, a stage characterized by behavioral stability and predictability.

Mid-life crisis/second childhood. For humans, this stage often occurs as a result of a changing environment. As children leave home or financial pressures change, 'business as usual' for the adult is no longer appropriate and this forces a struggle to 'fit in' to new situations and changed expectations.

For software systems, the stage may be initiated by a step-change in technology creating new expectations and a situation where a system no longer behaves as required. Some efforts are made to adapt the system to new demands, but this is generally problematic.

The third age. This stage is characterized by efforts to keep systems, and ourselves, employed, sometimes as backup for newer 'replacements'. However, the cost of care tends to increase as problems become more severe and the need for support more frequent.

The analogy allows us to make the following observations. We discuss implications in Section 3.

Outcomes are determined by both nature – expertise and capabilities of developers and the environment during development – and nurture – how the system behaves and is managed during the early stages of life. System performance, considered broadly, is influenced by both sets of factors. However, just as people continue to grow and change during a life-time, software systems are never fully formed or entirely stable. This maps to the idea of emergent organisations [11] that face an ongoing need to change and adapt to their environments – which could otherwise be called evolving. Furthermore, the lifetime of a given software system may be short or long – in some cases, a product family might evolve and 'live' for generations.

If environmental factors do change then the health and well-being of the system can also change. A person with low body fat and high metabolism may function well in a hot climate, but less well in a cold climate. In a similar way, a software system that functions well in a particular business environment may function less well if the context changes. Change can of course be designed and deliberate or can be unanticipated. A person may choose to learn a new language or embark on a physical fitness program. Or a family member may become sick, forcing an unplanned change in duties and behaviors. Systems development may also be intentional, for example, a project is set up to add new features to a product, or initiated by environmental change, for example, the stakeholder market adopts a new technology.

What is important with respect to health status and well-being depends on life stage and on the specific characteristics of the system. For example, babies undergo a '10 point test' at birth - the higher the score, the healthier the baby. Also there could be specialized attributes or attribute thresholds that are sought in certain types of individual, for example, high performance athletes. In a similar way, software systems are monitored against key indicators, for example, defect levels, and 'healthy' values depend upon life stage – some defects may be acceptable during early childhood, but not during adulthood.

Processes that move you away from core values and objectives cause malaise. For many humans, a poor diet and lack of exercise may cause high blood pressure and cholesterol levels. For bespoke software systems, poor change control may lead to cost over-runs. In both cases, a strategy of changing root cause is more effective than addressing symptoms only. That said, where cause and effect is not known, treatment of symptoms may provide short-term relief.

Once a set of behaviors is established, change is difficult. This characteristic is almost non-existent in children but becomes more pronounced with age. It can be difficult to change from processes that reduce health status to alternatives that prevent or reduce the likelihood of sickness. Software systems show a similar characteristic. A product could be difficult to change, with unanticipated consequences, and system stakeholders may also be resistant to change.

As noted, people sometimes become ill and require treatment from an external professional. But a person may also proactively self-manage health, for example, by monitoring body weight and cholesterol levels and changing eating habits if these fall from desired levels. If we consider a software system as comprising a product ('body') and stakeholders ('consciousness'), we may observe that a system 'gets sick' and requires treatment (maintenance); or it may proactively choose to monitor performance and implement process change if this falls below desired levels. Taking this notion a step further leads easily to the principles of software system autonomy, reflecting systems' abilities to self-protect, self-diagnose, self-heal and so on.

In Table 1, we illustrate some of the ideas presented in this section. In the left column, we show four scenarios of system health: sickness, prevention, growth and environmental change. For each, we state some characteristics. In columns two and three, we provide some examples of these characteristics for human and software systems.

3. IMPLICATIONS AND INSIGHTS

Our discussion of life stages raises many points for discussion, only some of which we consider here. Research indicates that the reality for many organizations is constant change [11]. In such a context, software systems are never fully formed, and must also continually adapt. We may view these systems as being in a prolonged state of adolescence or mid-life crisis. It is no longer appropriate to undertake development with the assumption that 'adulthood' is the only end-goal.

Table 1: Examples of system characteristics

Scenario	People	Software
Sickness - Symptoms - Indicators unhealthy - Find cause and treat	 Headache Temperature Take medicine	 Stakeholders unhappy Defect numbers Fix defects
Prevention - Monitor indicators - Preventative action	 Cholesterol Lifestyle change	 Product quality Process/product change
Growth - Identify objectives - Confounding factors - Make changes	 Run marathon Motivation Training, diet	 Globalise Short term focus Gap analysis & change
Environment change - New environment - Indicator gaps - Confounding factors - Close gaps	 Redundancy Computer skills Confidence Take course	 Business environment All web-based No web developers Hire web developers

The idea that systems are characterised by different values of key indicators at different times and that systems with different objectives require different indicators leads us to question the applicability of the current focus on process definition. Processes cause change to indicator values. Before knowing which processes to apply, we must first understand what we are aiming to change. A different approach to defining 'best practice' may therefore be needed. This also relates to software process improvement. This tends to involve a gap analysis against 'best practice processes' and implementing missing processes. However, from Table 1 it is clear that appropriate process change is dependent upon first understanding what is the scenario for change and then identifying what indicators are relevant. An individual may not want to have the heart rate of a top athlete if it involves spending time they do not have and no plans to run a marathon. We must first understand whether the system is sick, needs a preventative approach, must exist in changed circumstances or is reacting to unplanned change in the environment. We may then investigate indicators and appropriate processes that will close gaps. When viewed from this perspective, we may also consider that improvement systems, for example, CMMI and SPICE, are based on the assumptions that health is the same for all systems and that 'sickness' is the only reason for change.

Human parents are given guidelines for a healthy outcome, but are not all expected to, for example, eat the same foods, take the same amounts of rest or exercise in the same way. If we apply the idea of guidelines rather than prescription to software projects along with the understanding that there are multiple 'parents' contributing to decisions and outcomes, this suggests the need for a project management paradigm that acknowledges context, perspectives and other 'soft' factors as key. This viewpoint is consistent with ideas emerging from the project management literature that advocate a move away from historical prescriptive project management practices towards an approach that focuses more on social process and context-dependent judgment [6]. In our analogy there is scope for both approaches – 'sickness' and 'prevention' may be viewed as ongoing aspects of life, whereas 'growth' and 'environment change' are generally out of the ordinary and require the setting up of projects i.e. defining of goals and desired outcomes.

One final observation relates to the 'doctoring' aspect of software systems. For humans, the roles of parent, person and doctor are clearly separated. People have General Practitioners (GPs) who monitor aspects of health and make initial diagnoses across a broad range of ailments. Where problems are more complex specialist professionals may need to be consulted. Specific treatments can be administered by yet other specialists. Well-person clinicians can recommend proactive steps in order to prolong well-being. Parents, although responsible for the characteristics of a small number of related offspring, are not expected to be experts in, for example, the digestive systems of their offspring, or how the various human systems interact.

With software systems, however, we suggest that there is a less clear separation of concerns. Developers are generally expected to understand many aspects of the products they create. Such aspects relate not only to 'simple' structural concerns, for example, module structure, but to complex interactions between system components under many different environmental situations [4]. Developers may be supported by Quality Assurance personnel, but although the QA role is to monitor and suggest process change, the advice may be general and unlikely to provide support specific to the nature of the problem. It is rather like always telling an expectant mother to 'eat well' when there is a problem of gestational diabetes and some stronger intervention is indicated. After 'birth', maintenance personnel deal with problems, but again, there is no expectation of specialist expertise - indeed, novices may be assigned maintenance roles, and developers are generally seen as the 'experts' when a tricky problem is encountered.

So perhaps a rethink of the roles is indicated. We certainly need to continue to educate and train parents and GPs, but we may also need to direct more effort to training specialist practitioners in diagnosis, treatment and intervention, mapping to surgical, medical and other disciplines. Functional area specialists may hold domain expertise: pediatrics ≈ business systems. Technical area specialists may be infrastructural experts: vascular surgeon ≈ network specialist. Under such an approach there may be greater acceptance that there are levels of judgement involved in the work of these individuals. However, we would also expect from them informed, evidence-based practice, and that their professional competence would be monitored in an ongoing manner through peer assessment.

4. FINAL REMARKS

Metaphors "are inherently partial" and serve to "illuminate certain features of a phenomenon whilst obscuring others" [7]. There are certainly aspects of the analogy that fail to map effectively – a simple example is the lack of a fixed incubation length for software systems. There are also sensitive aspects of both the analogy and its link to medicine that need scrutiny. However we believe that at present the insights gained outweigh such limitations. Given space constraints we have also not addressed related work here, although it most certainly exists. In particular, software system health has been considered in [12], and in [13] under an aspect oriented approach, and the notion of software aging has been considered by many authors, including [14] and [15]. A very relevant staged model of software was described in [16] and life-stage models exist for other technological innovations [17]. We believe that our work is complementary to these studies and extends the analogy to enable further insights to be gained.

5. REFERENCES

[1] Broy, M. (2006) "The 'Grand Challenge' in informatics: engineering software-intensive systems", *Computer* October: 72-80.

[2] Lawson, H.W. (2002) "Rebirth of the computer industry", *Commun. of the ACM* 45(6): 25-29.

[3] Brooks, F.P. (1987) "No silver bullet – essence and accident in software engineering", *Computer* (April): 10-19.

[4] Fiadeiro, J.L. (2007) "Designing for software's social complexity", *Computer* January: 34-39.

[5] Müller-Schloer, C. (2004) "Organic computing – on the feasibility of controlled emergence", in Proc. *CODES+ISSS'04*, ACM.

[6] Cicmil, S., T. Williams, J. Thomas, and D. Hodgson (2006) "Rethinking Project Management:

Researching the actuality of projects”, *Intl Jnl of Proj Mgmnt* 24: 675-686.

[7] Drummond, H., and J. Hodgson (2003) “The chimpanzees’ tea party: a new metaphor for project managers”, *Jnl of Information Technology* 18: 151-158.

[8] Liu, J., and K.C. Tsui (2006) “Toward nature-inspired computing”, *Commun. of the ACM* 49(10): 59-64.

[9] Teuscher, C. (2006) “Biologically uninspired computer science”, *Commun. of the ACM* 49(11): 27-29.

[10] Wang, W.-L. (2002) “Beware the engineering metaphor”, *Commun. of the ACM* 45(5): 27-29.

[11] Truex, D.P, R. Baskerville and H. Klein (1999) “Growing systems in emergent organizations”, *Commun. of the ACM* 42(8): 117-123.

[12] Iverson, D.L. (2004) “Inductive system health monitoring”, in Proc. *IC-AI*, CSREA Press.

[13] Lau, A., and R.E. Seviora (2005) “Design patterns for software health monitoring”, Proc. *ICECCS*, IEEE.

[14] Madhavji, N. (2002) “Beyond the next release”, in Proc. *CASCON*, IBM.

[15] Miller, R.L., and J. Morley (1996) “Geriatric systems: the need for reverse engineering”, in Proc. *NAECON*, IEEE.

[16] Rajlich, V.T., and K.H. Bennett (2000) “A staged model for the software life cycle”, *Computer* (July): 66-71.

[17] Nolan, R.L. (1979) “Managing the crisis in data processing”, *Harvard Business Review* 57 (2): 115-126.