Tinker, tailor, software engineer, surgeon: specialization in software systems creation and evolution

Stephen G. MacDonell and Diana Kirk

SERL, Computing and Mathematical Sciences
Auckland University of Technology
Private Bag 92006, Auckland 1142, New Zealand
stephen.macdonell@aut.ac.nz, diana.kirk@aut.ac.nz

Abstract

This position paper advocates a change of mindset regarding how we perceive of and support those who develop and maintain software systems. We contend that a lack of explicit specialization is impeding our ability to deal effectively with the challenges that arise in the creation and evolution of software systems. Observations from the health sector lead us to reconsider the roles of the professionals involved.

Keywords: software engineering, software evolution, role specialization

1. INTRODUCTION

The burgeoning complexity faced by those building and maintaining software systems is widely acknowledged [1]. A consequence of this complexity is that, at all stages in a software life-cycle, an increasing range of expertise is required of the humans designing and managing the software product or service. The architect for a banking system with expertise in databases may be faced with designing an upgrade from a client-server to a web-based architecture. The need for improved security to address newly identified network vulnerabilities may require novice developers to rapidly learn and use protocol analysis techniques. An organization with an innovative software product may need to implement a 'light' approach to development with personnel that lack the understanding or ability to effectively support clients.

Why not simply implement training, initiate process improvement, hire new personnel, and so on -'standard' responses to such demands? We contend that technologies are just too complex and fast-changing for training to keep up and the required expertise has become too broad for individuals to cope. If we consider the product or service alone, we must consider all of the *application area* (business, telephony, health...), the *product type* (web-based transaction, embedded real-time...) and the *operational environment* (desktop application, client server, distributed...). For a single organization, it is likely that the first (*application area*) would remain constant but equally likely that step changes in technology will force ongoing revisions of both *product type* and *operational environment*, with repercussions for both development and support personnel. If product and management processes are also considered in addition to the product or service, we must add the need to familiarize with a range of evolving techniques and tools, e.g. relating to requirements elicitation, change control, development, test and documentation. In addition to the techniques and tools that are 'standard' for the organization, project members must know how to deal with contextual issues. For example, an 'agile' group may have a process that assumes an on-site customer, and be left confused as to what to do on a project where there is no such access to customer.

There is a sense then that in software engineering (SE) there is more and more that we *need* to know. In broadening the body of knowledge, however, something has to give -we need more time to learn, to develop professionally, and/or we need to trade off breadth for depth. As a community we have not addressed this issue in an explicit and meaningful way. Basically, we have just tried to fit more in into curricula, training courses, expectations for professional development. Such an approach is not sustainable, and we believe that the time has come to encourage greater disciplinary specialization. Elsewhere [3] we have suggested that the state of a software system at any point in time may be considered in terms of its health and well-being, just as we ourselves move through life stages characterized by states of wellness or illness. Taking this metaphor further can also provide insights into how we view the roles of those involved in the development and support of systems over time.

2. SYSTEMS AND SPECIALIZATION

Leveraging a human health metaphor can enable us to 'see' a software system in multiple ways, to depict a system depending on issues in focus. For instance, we may consider a software system to be young, mature, or aging [4]. Software systems in each grouping have particular attributes to consider when assessing their state of well-being. Similarly, we may view some systems as being ill and others as being well. Complementary to these approaches is a characterization of software systems according to their type. Systems that are hardware-embedded and operating in real-time are different to web-based transaction systems, and these again are different to personalized adaptive systems to be used on mobile devices. We can also consider systems in terms of their core domain or operating context -systems for business, systems for manufacturing, systems for assisted living.

We see such perspectives having direct analogues in the health sector, with what we believe to be useful implications. A person is also a complex system of systems that can be viewed in different ways -this may be in general classifications of age or maturity, or states of wellness/illness. We have infrastructural systems such as the cardiovascular, the musculo-skeletal, the respiratory, the nervous (somatic and autonomic) and the digestive. These could be seen as mapping to our network and database systems, to our computer and software architectures, to our application, interface and domain representations. As humans we also comprise physiological and neurological systems that have physical and conceptual or logical elements -which may map to systems supporting workflow and enterprise activities.

All of these are valid perspectives for the consideration of software systems, each encourages and enables us to think about the system in a particular way, with a particular set of attributes in focus. However, in acknowledging that the various perspectives exist we are also acknowledging that they each represent part of the whole, and as such provide a potentially useful but inherently limited sense of the system. Furthermore, such systems and subsystems are not independent, they will interact in generally expected but sometimes unpredictable ways. Traditionally, software engineers have attempted to understand and cope with this breadth of perspective -an alternative is to allow for greater specialization with comparatively greater depth.

We take our model for role specialization from the health professions, an analogy also considered by Laplante [2] and others. Like software professionals, health professionals deal with a very complex entity in the human person. They are aware of interacting subsystems within that entity, as described above. In order to effectively understand, diagnose and treat that entity health professionals have adopted a specialized model. So, there are personnel with specific competencies in medicine or in surgery. Individuals may be experts in mental health, or provide specialist diagnostic support. Others take particular responsibility for assessment and rehabilitation. In the provision of services there is a clear distinction between primary and community care. In acknowledging that people of different ages may require different forms of support the health disciplines have gerontologists for those who are aging, and paediatricians for the young. In acknowledging the many subsystems that make up a person they have the nephrologist, the cardiologist, the gastroenterologist, the neurologist and the neurosurgeon. And there is finer granularity still when these perspectives intersect e.g. the paediatric nephrologist. This is not to say that generalists are not permitted. Quite the contrary, in fact, there are of course general practitioners in health. However, general practice is itself seen as a *specialty* - a specialty in breadth, requiring particular diagnostic skills, capabilities and ongoing professional development.

It may be suggested that we already have specialization in SE. This may be true to an extent, and certainly there are recognized specialist areas within the discipline, specialist degrees and courses that can be taken. In general, however, we continue to educate ‘software engineers’ - the equivalent to health’s general practitioner. Our archival publication venues address the whole of the discipline, and job advertisements still seek software engineers albeit with particular skills and capabilities. However, these skills tend to reflect competence with the tools and technologies rather than with the application of tools to a particular class of (sub)system -like advertising for a surgeon with the requirement “Must have recent experience with Scalpel 3.0”.

It may also be suggested that specialization is in fact a bad thing. For instance, specialization in SE has been criticized as causing an inability to “think holistically about the particulars of the problem that have been abstracted away, and that now may be the responsibility of no one in particular” [5]. We see this kind of specialization as a kind of ‘functional specialization’ that applies to the scoping of *activities*. For example, a developer may understand that (s)he must not modify design decisions, as this is the responsibility of the designer. We submit that the kind of specialization proposed in this paper is essentially different in that the scoping relates to the nature of the system, for example, to characteristics of the product or operating environment. Furthermore, the health model for team-based treatment helps to avoid inadequacies arising from specialized (and so limited) understanding. Multi-disciplinary teams (MDTs) work on a single entity in sequence and sometimes in parallel, often co-ordinated by a senior clinician. Members are called upon depending on the conditions encountered and the treatment plans chosen. Inter-disciplinary teams (IDTs) have an even greater degree of interaction -rather than a chain of individual specialists the group forms a network.

3. IMPLICATION

Carried through to software, the health model has particular implications for education and training. Prospective professionals would spend longer in formal education, but this would be increasingly practice-based, with internships the norm rather than the exception. A general foundation education would be followed by specialist learning and training, ongoing under compulsory development programs monitored by peer professionals. Some of this is done already, but it is not common practice. Professional development carries with it an expectation of self-reflection and learning in concert with a community-based approach to developing knowledge. For instance, the legal profession has case law, the medics have their grand rounds and evidence-based learning enabled by the Cochrane Collaboration. Of course, software is not a social service that governments support for the greater good, as they might for the health professions. But nor is law, yet we would not tolerate a law profession that did not learn as a community. Along with other changes in thinking and conduct [1, 2], disciplinary specialization may add to SE's growing professional standing.

4. REFERENCES

[1] T. DeMarco. Software engineering: an idea whose time has come and gone? *IEEE Software*, Jul/Aug, 2009.

[2] P. A. Laplante. Professional licensing and the social transformation of software engineers. *IEEE Tech. Soc.*, Summer, 2005.

[3] S. G. MacDonell, D. Kirk, and L. McLeod. Raising healthy software systems. In *4th Intl Workshop Softw. Evoln Evolvability*, L'Aquila, 2008. IEEE CS Press.

[4] D. L. Parnas. Software aging. In *16th Intl Conf. Softw. Eng. (ICSE)*, Sorrento, 1994. IEEE CS Press.

[5] R. Schaefer. A rational theory of system-making systems. *Software Engineering Notes*, 31(2), 2006.