Chapter 69
Assessing the Potential Improvement an Open Systems Development Perspective Could Offer to the Software Evolution Paradigm

James Austin Cowling
Cardiff University, Cardiff, UK

Wendy K. Ivins
Cardiff University, Cardiff, UK

ABSTRACT

Emerging stakeholder needs and a changing environment drive increasing demands for the constant adaption of software through maintenance and new capability development. A more evolutionary software engineering approach is sought to improve engineering responsiveness; Open System Development appears to offer a partial contribution but presents many challenges. This exploratory research proposes a new definition of the evolution of complex engineered systems, building on the essential features of 'openness' described by Cowling et al (2014) and Lehman's (1980) ideas of evolutionary software. Using Checkland's (1999) Soft System Methodology enabled a structured literature review and analysis of the relative contributions of three divergent methodologies to the success of systems outcomes. These methodologies are: Plan-Driven, Agile, and Open Source Software Development. The analysis reveals several opportunities and highlights the critical issue of determining return on investment which needs to be overcome if an open approach is to contribute to evolutionary software engineering.

DOI: 10.4018/978-1-7998-3016-0.ch069

INTRODUCTION

When engineering bespoke information management systems at scale, for a diverse range of specialist users with rapidly changing needs, established software practices may be sub-optimal. The software engineering discipline may respond to such complexity and volatility in a variety of ways but this paper considers countering such challenges with a more evolutionary approach; exploring the characteristics of open system development that may improve the usefulness of the evolutionary software paradigm. Large scale open community engagement together with iterative development centred on conjoined software maintenance and improvement are of significant interest in building a model for complex evolutionary software engineering.

Engineering disciplines have made great strides in developing repeatable, measurable, predictable, and controllable processes to aid the management of increasingly complex system development projects. Engineering management tradecraft has tackled complexity in a range of ways including: breaking down the end deliverable into smaller elements and systems, each of lower complexity for subsequent integration or; prioritising requirements and developing a solution in a number of successive stages.

What one might consider "traditional" practice encourages the process of elaborating requirements, decomposing the solution, and, through hierarchical governance structures, controlling the planned delivery of the resultant work packages into an integrated final deliverable. This plan-driven approach offers advantages in cases where the requirements are understood and complexity of the solution stems from its *intricacy*.

In more recent times the "agile" movement has focussed on software engineering practices that encourage iterative development methods, centring on close interactions between the developers and customer. Early advocates of these practices encapsulated their ideals in the "Agile Manifesto" triggering the emergence of a tranche of engineering methods that address complexity rooted in the *uncertainty* of requirements; allowing for the exploration of the problem over a large number of short development iterations. quote from The Agile Manifesto, August 2001:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

The challenge of complexity is heightened when both solution intricacy and requirements uncertainty are high; and further magnified when these conditions are driven by volatility in the environment within which the software system operates or is intended to operate. Lehman (1980) defined such software as E-programs: "software systems that are inherently highly prone to change and, given they mechanise a human or societal activity have a strong interaction with their environment and must adapt to changes in that environment". Lehman went on to propose a number of laws of software evolution including that of "Continuing Change" which states that "A program that is used and that as an implementation of its specification reflects some other reality, undergoes continual change or becomes progressively