Software Engineering and Policy

Anthony Finkelstein
The Alan Turing Institute*
British Library
London, UK
afinkelstein@turing.ac.uk

ABSTRACT

In this short position paper I consider the contributions that software engineering as a discipline can make to the development and implementation of government policy. It is intended to support the growing body of knowledge on scientific advice in government and to encourage software engineers to engage with policy and the policy community.

Categories and Subject Descriptors

Social and Professional Topics [**Computing/Technology Policy**]; Software and its Engineering [**Software Creation and Management**]

1. CONTEXT

In developed societies the key functions of the State are, broadly construed, promoting economic prosperity, ensuring defence and national and individual security, and health and social welfare including education. These functions are undertaken to different extents and in different manners dependent upon the political constitution of the State and the context in which it operates. This paper draws on UK experience but seeks insofar as possible to have broader transnational application.

The means by which the State exercises its functions are complex but, simplifying, political intent formed by a government and legislature (precise arrangements differ) are realised through the development of policy, an abstract operational formulation of that intent. The executive branches of government use this policy to determine the actions they must take and the necessary distribution of resources.

Software technology, as a key constituent element of information technology, bears profoundly on the exercise of each of the key functions of the State. The economy is intimately linked to the use of software technology in support of communications and the conduct of business. The development of software systems is also in itself a significant productive business sector. Defence and security are enabled though software technology and increasingly the domain of defence and security extends to the information and communication systems forming part of the critical national infrastructure. Healthcare is dependent upon software systems for delivery and a growing class of devices and complex interventions. Not only is software technology tightly bound to the direct exercise of the functions of the State but the monitoring and control of these functions is achieved though information systems and hence by way of software technology. Clearly then, much policy must be informed by an understanding of software engineering. I am not however, intending to look at the software engineering relevant content of policy, not least because it would require a large thesis rather than this brief paper. Instead, I am intending to examine how software engineering might deliver insights into, and perhaps valuable tools for, the policy process.

2. CONTRIBUTIONS

There are three linked aspects of the contribution of software engineering to policy each separately discussed below. First, and perhaps most obviously, software engineering provides powerful conceptual tools for understanding and analysing complex systems - and many of the subjects of policy are archetypal complex systems. Second, there are deep and important similarities between policy processes and software development processes that may yield, not just intriguing analogies, but may suggest additionally suggest improved policy processes and tools. Third, computational models, of greater or lesser sophistication (ranging from large climate models to ephemeral spreadsheets), are increasingly used to inform policy, thereby tying policy to the correctness, completeness and consistency of the models (this draws on my blog http://prof.so which contains an extended discussion).

2.1 Representation

As software engineers we are inclined to discount the value and power of the conceptual toolkit that has been developed in our discipline. We can, of course, draw on a very large set of modelling languages. Simple schemes such as data flow diagrams, entity-relationship models, petri-nets, collaboration diagrams, and the like, whilst we recognise their limitations for software construction, are extremely powerful as

*The Alan Turing Institute is the UK National Institute for the Data Sciences jointly established by EPSRC and the Universities of Cambridge, Edinburgh, Oxford, UCL and Warwick. The author is at UCL and HM Government Office for Science (GoScience).

a means for unpicking complexity in policy domains. Yet more powerful is the software engineering stance in which complex domains are approached as language design challenges. Typically a software engineer when faced with a complex situation will carefully design a language (usually on the armature of a meta-modelling scheme or language) structured to express the relevant properties and characteristic organisational features of the domain. The design of the language often yields insights and the resultant models provide immense leverage. The practice of software engineers of systematic language design and use can usefully be contrasted with the general proclivity in the policy domain to use representations - even block diagrams and control flow charts - loosely and without regard to their syntax and semantics.

2.2 Processes

The relationship between policy processes and software development starts with the similarity between policy and specification. Policy provides an abstract operational formulation of political intent. It is refined through the policy process into requirements that are placed upon the executive organisations through which government acts. This is supported by legislation that provides a framework of rules that mandates the executive organisations to respond to policy and may additionally provide additional policy-derived requirements. In response to the requirements the executive organisations must provide capabilities and on those capabilities must deliver services or operate processes that satisfy the requirements. In other words they implement the specification. The service delivery is effectively the 'run-time'. There are feedback processes in operation between policy formulation and implementation, often concerning the feasibility of the policy and the resource implications of different possible implementations, and run-time monitoring feedback, largely concerning the quality and performance of the services.

The analogy is fertile, and particularly in suggesting how policy requirements might be documented and related to implementation. Thus, it brings to the fore matters such as: the precision of the expression and the consistency of the policy requirements; the specification of the test cases by which successful implementation can be assessed and more generally the extent to which the implementation securely realises the policy; the non-functional properties (performance, use ability, quality) that the services must meet; the traceability between the implementation and the policy; the environmental assumptions underpinning the policy; and so on. Software engineers have also the ability to make useful contributions to the structure and organisation of processes. Thus, spiral, incremental and agile models can potentially be applied within the policy process yielding benefits similar to those we are familiar with in software engineering.

2.3 Models

Many of the issues around which policy rotates are too complex to be determined by straightforward 'qualitative' reasoning. They involve multiple interlocking constraints that relate resources, effects, finance and anticipated behaviours by a range of independent actors. Often these give rise to feedbacks yielding dynamic behaviour that is far from obvious. Further, there are timing factors that mean that certain actions are only possible at certain times or only if specified environmental conditions hold. This creates further networks of temporal dependencies. It is clear in these circumstances that modelling is required to aid understanding and analysis.

It is unclear whether those who understand the policy dimensions of the problem are equipped to do the necessary modelling or perhaps really understand what such models can, and cannot, yield. Even if they are equipped, their first instinct is to construct a large and elaborate spreadsheet. Spreadsheets can be an extremely powerful tool and have the benefit of being relatively simple to use for straightforward planning tasks. They are however, also very difficult to debug, to test, to document systematically and to understand except in a piecemeal fashion. The sophisticated user, or worse the clever but unsophisticated user, can create serious problems. The full functionality of Excel with workbooks, scripting and pivot tables can approximate to that of a powerful software development environment. I am not sure that there has ever been a full accounting for the societal consequences of faulty spreadsheets. The fact however, remains that serious and important policy alternatives, the case for major infrastructure investments for instance, depend on spreadsheet models.

When policy analysis becomes really thorny it is necessary to construct a fully fledged computational model. Climate change mitigation and energy policy being cases in point but health resourcing and transport planning are also candidate examples. The models are usually built by experienced modellers who have access to sophisticated tools and have a good appreciation of issues such as sensitivity and model validation. Unfortunately modelling experience is not necessarily associated with a software engineering understanding of model construction. Some of the basics - specification, documentation, clear interfaces, modularisation schemes, built in error checking, systematic testing - are commonly neglected. Errors can be introduced in the data, in the modelling assumptions, in the model, in the model coding and in the language and data storage components on which the structure rests. They can also arise in the interrogation of the model and the ways in which results are presented or visualised. Chasing these errors through the multiple layers represents a major challenge that lies at, or perhaps even somewhat beyond, the state of the art in software engineering.

3. CONCLUSION

It is obviously to be hoped that the benefits will cut both ways and in areas such as stakeholder management the practice of policy development and implementation might contribute to software engineering. All analogies, however appealing have risks and it is important to understand where software engineering and policy depart from each other. We have often, as a profession, been reluctant to make strong claims about the broader value of our tools and conceptual frameworks, perhaps conditioned by a close familiarity with their limitations. There are clearly important roles for software engineers to play in the policy domain. The social and economic impact that we can make by suitable engagement is substantial. This position paper has pointed to some directions we could take and concludes with this, a call to action.