Foreword – Economics-Driven Software Architecting

John Grundy

Swinburne University of Technology

Software architecting has become a critical part of most software systems development projects. However, as popularised in the seminal papers by Shaw and Garlan (1996) and Kruchten (1995), most software architecture research and practice has historically been focused on the technical aspects of the task, not value-driven or value-creation aspects (Boehm & Sullivan, 2000; Bahsoon, & Emmerich, 2008). Given the huge impact software architecture choices have on system performance, scalability and maintainability, not to mention the actual adoption and deployment of systems, it has become accepted that software architecting can no longer ignore economics-driven imperatives (Bass et al, 2003; Kruchten et al, 2006).

So saying, there is a need to both incorporate an economics-driven perspective into the software architecting of systems and to balance this with continuing to solve increasingly challenging technical problems. I see several major areas for incorporating a software economics perspective in the architecting process:

- *Requirements Constraints* (particularly Quality of Service on software architecture) on architecture – requirements are increasingly challenging to meet with increased demand on scaling, reliability, robustness, and security. Most if not all of these constrain the architectural solution space, sometimes in conflicting ways. Meeting some implies more costly, less flexible and less extensible architectural solutions. Expected or unanticipated changing QoS can have dramatic impact on architecture, including rendering a particular architecture choice no longer tenable.
- *Technical Constraints* on architecture – technical choices may be constrained by a range of issues, not limited to cost of solution platform (software, hardware, network, 3rd party applications); availability; open vs proprietary; and likely future technological change e.g. the emergence of the cloud computing paradigm. Once commitment has been made to one set of technical solutions, change is almost invariably expensive.
- *Deployment environment* constraints – architectural solutions make heavy use of other applications and components in their deployment environment, as well as platform hardware and networks. With the advent of cloud computing, a variety of platform-as-a-service and infrastructure-as-a-service, with elastic availability, has complicated this aspect further. Different platforms offer different technical and QoS constraints, but also come with differing cost models.
- *Process constraints* greatly impact software architecting – the advent of agile methods with the concept of the "emergent" architecture has some strong cost incentives – implement only as much as you have to and no more. Unfortunately, as applications and user requirements evolve,

sometimes dramatic re-architecting and re-engineering becomes necessary, with attendant cost implications.

- *Team constraints* are an interestingly interesting area. Sometimes many of the above decisions are actually made due to team – or organisational – preferences, biases, and prior experiences. Sometimes dominant individuals or politically or economically powerful lobbies hold sway over key architectural solution decisions, disregarding some of the factors above and including cost and longer term software economic implications. Sometimes architecture choices are driven by the concept of "value" to the team e.g. a reason for exploring interesting new technologies and solving interesting technical problems, instead of value to business, stakeholders or even the maximising of the actual value of the software itself.
- Finally, *business constraints* must be taken into account. These include limitations on purchase or usage of hardware and software, desire to leverage value in the future if not the present, balancing cost vs opportunity, and potentially increasing attractiveness of product due to architectural decisions that appeal to likely or potential stakeholders. Simply choosing the "cheapest" options often turns out to be a false economy. Similarly, massively over-engineering applications and consequence of cost and loss of time-to-market edge can have severe economic and value implications.

The concept of economics needs to be incorporated into the software architecting process itself. Traditionally, most if not all of the practice of software architecture has been technical in focus. However, many constraints impacting on architectural decision making have major implications on platform and software costs, team costs, enhancement of software value, enhancement of company value and capability, and positioning of product and organisation for (un)anticipated future demand. Below I suggest some ways in each area above that this might be advanced.

Requirements Impact on Architecture Economics

Architectures must realise functional and non-functional requirements set for them. As several recent works have identified (Avgeriou et al, 2011), many now think software architecting and requirements engineering must proceed hand-in-hand rather than the former rigidly follow the later. In fact, architectural choices often constrain the requirements as often as not (Tang et al, 2011, Woods & Rozanski, 2011). Quality of Service (QoS) constraints have a major impact on the economics of a software architecture, both in terms of the expense of its solution but also the value inherent to the solution.

Typically, the higher the QoS expectations e.g. greater the throughput, reliability, security, and scalability needed by stakeholders for their application, greater the complexity, implementation and cost required. However, a simple, cheap, quick-to-build web application that is intended to manage highly mission-critical or sensitive data is going to have much less – if any – value, than one demonstrably safe, secure, robust and clearly meeting the QoS requirements set for it. Similarly, an over-engineered solution that will never be required to scale, be as secure,

manage highly private data, or face a range of difficult deployment environment conditions, will waste design, development, testing and deployment resources to little or no added value.

Balancing the current QoS needs against architectural decisions that meet these, but do not greatly exceed them, is challenging. In particular, as many QoS constraints and suitable architectural solutions conflict, trade-off analysis can be very challenging. Factoring in cost in terms of development time, testing, required platform support and business value needs to be a part of decision making, not just technical solutions (Bahsoon and Emmerick, 2011).

A major difficulty is changing requirements. Adding or modifying functionality is hard, but incorporating potentially massive QoS constraint changes is usually a much greater challenge. With the complexity and connectedness of today's applications this is probably going to get even more difficult. Architectural solutions allowing for more flexible QoS changes are likely to be ever more necessary (Allen et al, 1998).

Technology Impact on Architecture Economics

Different technical solutions for software architectures come with different inherent costs: some are free e.g. open source vs bought solutions. Some are easily modified and configurable whereas others are very constrained, saving expense if built-in functionality is required but incurring cost if extension is needed. Some require expensive hardware or third party software investment, whereas others allow for elastic, demand-driven pay-as-you-go infrastructure. Some are widely used and thus embody well-known and supported approaches. Others are bespoke solutions, possibly optimised but potentially with poorly known or undiscovered weaknesses.

Technology choice will hence impact dramatically on architectural economics in terms of both cost (both to build and to maintain), and value (e.g. is the solution able to be deployed with a wide range of off-the-shelf components or require purchase of expensive, third party components, putting off potential buyers). Some technology choices are more costly upfront e.g. cost to purchase, more difficult to use, more time-consuming for team, require more costly deployment environment support. However, they enable an architecture to be more "robust" under requirements change e.g. provide dynamic load balancing, run-time component switching, enable elastic resource management etc. As with requirements change, anticipating future technology demands is difficult in many situations but may enable both more cost-effective (in the long run) architecture choices to be made and may maximise software value.

Environment Impact on Architecture Economics

Software doesn't run in isolation – it is deployed on hardware, networks and with other 3rd party applications and services. These may well impact architecture choices and themselves impact cost-effectiveness of solutions and value of the system as a whole. A major traditional cost has been over-engineering solutions, not just the software architecture itself but its deployment environment e.g. massively over-allocation of compute nodes, data storage,

network capacity. Sometimes this "over-allocation" is actually capacity that is needed very rarely e.g. once a day/month/year peak processing.

The advent of the cloud computing model has gone some way to addressing this by enabling elastic provisioning and multi-tenant solutions, incorporated into the engineering of software systems (Grundy et al, 2011). However, variable cost models, immaturity of the platforms and development methods, and other issues such as legislative constraints on data location can impact the usefulness of this solution in terms of software economics. Understanding of different platforms implications on running costs, ability to scale up/down resource demand, robustness on failure, loss of control over security and load balancing, and other concerns makes impact on software architecting still very unclear.

Process Impact on Architecture Economics

Traditionally architecting has largely been performed before large scale implementation and testing, and re-architecting has only been attempted when major problems (scale and performance being the most common). However the advent of agile development processes, and the concept of "agile architecting" has gained prominence. Originally, architectural spikes and an "emergent" architecture was the result. However, emergent architecture is a bit like emergent behaviour or constraints for a software system: unpredictable, hard to control, hard to plan and manage, and hard to cost.

Any development process for a software system needs to factor in architecture design, analysis and – often – evolution. As many technologies and deployment environments themselves greatly constrain architectural solutions, so they will constrain architecting in the development process. Software cost estimation and management have become well-recognised areas of software engineering [Refs]. A development process that makes architecture cost estimation difficult – if not impossible – we be problematic.

Team Impact on Architecture Economics

Team dynamics, organisational context and constraints, and socio-technical aspects of software development may all play an overly-large role in architecture choices and hence cost and value implications. Developers may prefer architectural solutions and technologies they are familiar with. Conversely, they may prefer "new" or "interesting" options whose value to them is in their technical attractiveness but economically may have sub-optimal value to the software project, application, stakeholders and organisation. Architectural choices may be constrained by organisational and/or stakeholder preferences, that have little basis in sound technical – or economic – rationale. Attempting a project with a new technology the team, organisation and stakeholder has little or no experience with is a classic high risk strategy.

By far the major cost for most software projects is personnel – developer time and related overheads. Hence software architecture decisions that have positive, or negative, impact on developer time have great potential to impact a project economically. Factoring in impact on team into architectural decision making would seem an important area for research and practice.

Business Impact on Architecture Economics

Finally, software is normally developed for a purpose: to assist people (and increasingly, machines) in carrying out their activities more effectively, efficiently and in a satisfying way. Business constraints relating to software economics are of course myriad: constraints on investment in developing the software itself; constraints on platform capabilities to host the software; short-term development vs long-term maintenance costs; time-to-market challenges; changing business models; and changing legislative context. Value offered by software is also multi-faceted: improving efficient use of resources; reducing costs; enabling quick response to market demand; opening up previously unavailable business opportunities; and even enabling future software and business development by improved team skill set, system integration, and solidity of underlying software architecture. Software architecting incorporating business context and need is thus likely to better meet these needs and live within necessary constraints. Equally, software economics carefully considering architecture as a fundamental software value enabler is necessary.

References

Allen, R., Douence, R., & Garlan, D. (1998). Specifying and analyzing dynamic software architectures. Fundamental Approaches to Software Engineering, 21-37.

Avgeriou, P., Grundy, J., Hall, J.G., Lago, P., Mistrík, I. (2011), Relating Software Requirements and Architectures, Springer.

Bahsoon, R., & Emmerich, W. (2008). An economics-driven approach for valuing scalability in distributed architectures. In Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference on (pp. 9-18). IEEE.

Bahsoon, R., & Emmerich, W. (2011). Economics-Driven Architecting for Non Functional Requirements in the Presence of Middleware, Chapter 20 in Relating Software Requirements and Architectures, Springer.

Bass, L., Clements, P., Kazman, R. (2003). Software architecture in Practice, Addison-Wesley.

Boehm, B. W., & Sullivan, K. J. (2000). Software economics: a roadmap. In Proceedings of the conference on The future of Software engineering (pp. 319-343). ACM.

Grundy, J., Kaefer, G., Keong, J., & Liu, A. (2012). Software Engineering for the Cloud. Software, IEEE, 29(2), 26-29.

Kruchten, P. B. (1995). The 4+ 1 view model of architecture. *Software, IEEE*, *12*(6), 42-50.

Kruchten, P., Obbink, H., & Stafford, J. (2006). The past, present, and future for software architecture. Software, IEEE, 23(2), 22-30.

Shaw, M., & Garlan, D. (1996). Software architecture: perspectives on an emerging discipline.

Tang, A., Liang, P., Clerc, V., and van Vliet, H. (2011). Traceability in the Co-evolution of Architectural Requirements and Design, Chapter 4 in Relating Software Requirements and Architectures, Springer.

Woods, E., and Rozanski, N. (2011). How Software Architecture can Frame, Constrain and Inspire System Requirements, Chapter 19 in Relating Software Requirements and Architectures, Springer.