The State We're In

Martin Hunter, April 2017
https://www.continuity.kiwi.nz

"The State We're In". This is a play on the book of the same name by Will Hutton, but here we're rather more interested in the state of the IT industry rather than the state (or State with a capital S) of the U.K. from the mid '90s. And "in a state" the IT industry is; a somewhat worrisome state in which major IT project failures continue to plague the tech headlines[1], a failure rate of 70% + is the norm not the exception[2], and chaos seems to rule the bulk of software projects and products rather than solid engineering practice, rigour and discipline. In recent years anecdotal evidence seems to point to a worsening of the trend, the opposite of what one would expect of a well-oiled industry that takes structured lessons-learned exercises seriously and embeds learning in practice and regulatory policy going forward. IT projects seem to run up against the same old problems time after time. Why is this? Why does IT continue not to learn from its repeated mistakes? Why does it appear to be getting worse? Is not the definition of irrationality to continue acting in ways that you know don't work? In this article I hope to expose some of the mechanisms that I think are at work behind these trends and what we can do, collectively in our industry, to resolve them. I put forward that a perfect storm has occurred with the culmination of a natural evolution division of labour, commercial strategies and a general lack of understanding of what professional experience in IT means to the lay-person, investor or managing customer.

But first, I'd like to point out some general observations from my 20-year career in IT, what I might contend as a truism or a form of evidence if you will, that I hope some people working in IT are also familiar with. These form the basis of my arguments for what we do next.

---

1 http://calleam.com/WTPF/?page_id=3

2 https://home.kpmg.com/nz/en/home/insights/2013/07/project-management-survey-2013.html

(c) Martin Hunter, ArcWorx LTD., April 2017

The Evolution of IT Roles

The IT industry is old enough now (about 50-odd years) that it has seen 3-4 generations of people through it[3]. Historically people that we now call software "Developers" did most things; stakeholder engagement, analysis, design, programming, testing, infrastructure setup, deployment, operational support, even project management and sales. At some point in the 90's they began to be referred to as "Analyst / Programmers"; later the term "Developer" became the role title "du jour", after agile methodologies movements strove to differentiate themselves from the "old ways" of waterfall development, but reading even the entry paragraphs on agile methodologies[4] reveals that really we're talking about the same – or at least very similar – people and skill sets.

With the first few generations of workers, IT was still very young. Professional IT roles defined based on experiences in other industries gave way to roles that embedded some of the learning of the IT industry itself. Gradually roles evolved, as did the people performing them. In fact, most roles in the young IT industry seem to have evolved *with* the people that performed them: As overall industry experience grew, opportunities for specialisation were identified, understood and acted upon. As Adam Smith points out in his *Inquiry in to the Nature and Causes of the Wealth of Nations*, the division of labour is a pervasive force in almost all human endeavour, because it has such obvious advantages and immediately fruitful payoffs. But, as Smith also points out, when you specialise you also lose a little something; an understanding and direct personal experience of what other folks do. If we follow the evolution of our Developer roles, what we now see in the industry is division into specialist Developers (basically these are programmers now as specialisation has eroded their scope of practice) software designer, architects, analysts (business, systems), testers, operational supporters and managers; each and every one a different person. In recent years, new graduate entrants to the IT industry have tended to immediately specialise rather that engage in roles that teach

---

[3] If you count a generation as being approximately 15 years

[4] for example, http://www.ambysoft.com/essays/agileRoles.html, http://www.selfishprogramming.com/2009/04/23/the-role-of-an-agile-developer/, https://www.agilealliance.org/glossary/team/

(c) Martin Hunter, ArcWorx LTD., April 2017

them “a bit of everything” like the workers of a few decades ago received. As a consequence, it is fair to say I think that people working in the industry’s roles have become less aware of what their colleagues actually do each day. Anecdotally at least, analysts don’t really understand programmers, programmers don’t understand testers, and no-one seems to understand what an architect does except possibly architects themselves. Furthermore I am seeing a trend in programmers specialising even further; into front end developers (script monkeys), service developers, report developers and so on.

A Software House is a Craft Centre, not a Manufacturing Plant

Manufacturing industries have thrived on the division of labour due to their need to employ labour to mass-produce products through the performance of small, repeatable steps. Extensive division of labour serves this purpose well. Software is a lot different from manufacturing; it is more like a craft. In software we *craft* our product, then we reproduce it in nauseating, almost limitless numbers through digital duplication (mass duplication is the sum total of "mass production" in the context of Software). When you examine craft industries instead of large scale manufacturing, what you witness is a division of labour, and specialisms in roles, but in these scenarios the division of labour is a lot less extreme and usually specialism of an individual is only achieved after a long, generalist education or apprenticeship in the craft domain. This gives rise to a phenomenon that is all but impossible in manufacturing – the transferability of workers between jobs that demand different, though related, skills. In a craft, whilst expert specialists exist, they can usually turn their hand to other jobs that might need to be performed if necessary as well.

Under-Appreciated Experience

The evolutionary history of IT roles in software organisations, especially that of the Analyst / Programmer, has led us I think to place more value in specialist roles. Because of the craft-like nature of IT, one traditionally had to be a person with considerable experience to attain a role such as a manager, architect, analyst or test manager. Most people in Software were reared through programming and then branched out into something else. Today these roles are generally still more well paid than programming and

(c) Martin Hunter, ArcWorx LTD., April 2017

are generally more elevated in the organisational chart. There appears to be something of a “glass ceiling” when it comes to programming; if you have a career in software, "development" roles will only take you so far. For some time now, it has been the case I think that if you want to progress beyond “Senior Developer” then you must pass into a non-programming specialism: there has been no progression of the programming path to “Principal”, “Consultant” or “Managing” Developer in this career track. Experience of Developers seem to be capped at around 10-12 years, those with 20 years or more seem to be all but non-existent.

I Know Not What You Do

The problem here I think is twofold. Firstly, unlike building tower blocks or bridges, the focus and products of software engineering are largely abstract and hidden, which makes it difficult for laypeople to understand intuitively. Software products appear on the surface to be very simple as they're just images on a screen.

Furthermore this same dynamic makes software difficult to sell, which reduces or limits the sales strategies that can be applied (unless the software exists – and even then sales can be blighted by misunderstandings and non-truths by both non-technical sellers and customers). Ultimately the sales strategies of most of the software organisations I’ve ever worked for has been reduced to one thing: bill, baby, bill. Get your people under a timesheet code and say and do anything to achieve this.

Secondly, there just aren’t that many people in the world that make good IT professionals. IT demands a blend of hard and soft skills, of both analytical and creative thinking, of ability to concentrate on one task in spades but also to keep abreast of a large number of project activities, to continually come up with new ideas but to keep this bridled for the sake of project achievability. In a nutshell, whilst the pool of new entrants is generally high, the reality is that many don’t cut the mustard.

Even if it is not the ultimate cause, the fact that software is so intangible to most has, I think, exacerbated the under-appreciation of experience in programming. The “glass ceiling” appears because Senior Developers are

(c) Martin Hunter, ArcWorx LTD., April 2017

considered “good enough” to non-technical managers, ergo the expense of even more senior programmers is unjustified. The vitality of youth is misunderstood as being productive. The problem with this is: Senior Developers alone are not “good enough”. They still need guidance, help and mentoring in their profession, they still need more senior programmers than they to learn from. Historically it has been the Solution Architects that have filled-in for this role, given that they are usually very experienced Developers themselves and are senior staff members. But Solution Architecture is not Development. A Solution Architect’s development skills will generally wain in time because they are not getting continued exposure to the Development profession, their main focus is on something else. As a result their ability to mentor and guide Developers cannot be guaranteed and is dependent much more on their own personal ability to keep pace with Development whilst doing their job as a Solution Architect.

Staff Poaching

Another industry dynamic, driven by the limited sales and commercial strategies around software, is a deeply entrenched industry trend to poach staff rather than train them. To keep margins bearable, software organisations have traditionally cut training (if they had it to begin with). They would prefer to buy in skills that have been trained by someone else rather than train themselves. This is natural, of course: there is a risk to training in that your specific staff may not actually make the cut, the outcome is uncertain. Far easier and more predictable to hire in those with the skills already.

The problem is: this is almost everyone’s strategy. As a consequence, IT staff are not being trained properly in the industry at large, and they haven’t been for some time. “Training” often comes down to the personal motivation of IT workers; their ability to keep themselves up to date. Hence, I think, why some IT workers are obsessed with always using cutting-edge technologies and frameworks. This model is not ubiquitous, however: some large, international, disciplined organisations do formally train their staff. But, there are few of them these days in comparison to 15-20 years ago, and consequently the stream of properly trained, disciplined and experienced staff is lower today than two decades ago.

(c) Martin Hunter, ArcWorx LTD., April 2017

From Where Technology Ideas Originate

The invention and propagation of new technologies and approaches in software has for a long time been driven from two main sources. The first is obvious – new ideas; i.e. truly innovative, clever ideas that add value in themselves. The second is from the industry's skills specialisation, where the consequent lack of poly-skilled people drives the invention of new technologies and approaches to get around limitations of skills in making in software itself. In other words, software bends to the skills of the people making it.

An example of the first kind would be Object Oriented programming, an idea that has transformed programming for the better, and yet took decades for programmers at large to switch to because of the barrier to change in people knowing the incumbent, functional programming. I can't tell you the countless number of times I've seen functional programming thinking executed in Object Oriented programming languages, because the programmers didn't really understand Object Orientation.

Node.js is a product that is an example of the second kind. Node is a technology that's come about because of a specialisation in (and growing numbers of) Front End development, a discipline in which programmers know little else other than JavaScript and the ways of scripting engines. Node essentially allows front-end programmers use a terribly inefficient interpreted scripting language to write server side-code. Whilst Node does have advantages in that is easy to get server-side code running quickly, this is not unique to Node. There are plenty of ways to get server-side code running easily without resorting to running script on a server (refer Microsoft's highly simplified Kestral and ASP.NET Core for example). Hence I argue that the reason a market for something like Node.js exists at all is a reflection of the changing skill base in the programming community, not because Node is somehow a better idea than the last.

(c) Martin Hunter, ArcWorx LTD., April 2017

Repeat, Ad Nauseam

The culmination of the affects of role specialisation, the limitation in the experience of Developers, and the commercial strategies has led to a perfect storm in which IT projects are now beset with poor quality products and a lack of productivity. Inexperienced teams seem to struggle with basics and overcoming problems we've seen in the industry countless times before. More worrying still, we're now witnessing an increasing trend to construct development teams with only agile coaches, and no technical expertise beyond developers who are relatively very junior still, despite holding a senior job title. This is worrying because, whilst in the past the real experience in software has moved out of the programming profession into other project roles where it has withered, now these roles are being actively excluded. Projects have begun to cut their noses off by removing the last shred of lengthy software experience from their teams, possibly an indication of just how withered real software development experience has now become. These projects will be highly likely to fail in many of the ways that projects have failed in the past: poor requirements elicitation and management, poor estimation processes, limited design, poor documentation, under-estimation of complexity, constant and uncontrolled switching of technology frameworks as the team grapples with a desire to use the latest-and-greatest technologies at an unforeseen expense of productivity. Teams will continue to drink from the fountain of promise, without realising they're just part of a huge merry-go-round: Technology is not the most significant answer to productivity, disciplined ways of working and experience generally are.

The Bean-Counter's Blindspot

As pointed out by the International Project Leadership Academy[5], the productivity and quality challenges of teams are often not picked up through accounting practice. Endless technology framework readjustments, code/fix cycles and low experience are not accounted for by traditional project accounting. This is where the agile concept of "velocity" can help teams; a metric that attempts to track overall team performance and allow justified landing-time predictions and burn-downs. Velocity requires team

---

5 http://calleam.com/WTPF/?p=4729

(c) Martin Hunter, ArcWorx LTD., April 2017

discipline to estimate, derive and administer; in short, it requires experience. Any metric is only as good as the basis upon which it is derived and interpreted, the integrity of mechanism used to establish it. If you don't have the experience to construct and manage a meaningful velocity metric, you are liable to be caught in the bean-counter's blindspot.

What Should We Do?

- *Firstly*, we need to bring back apprenticeship in software engineering. We need to recognise that software is a craft industry, and to train our people accordingly. We need the division of labour and specialisation, but only after an extensive period of education (8-10 years) whereby people experience a broad range of the roles and skills that IT has to offer and requires.

- *Secondly*, to support this, we need to bring back the Analyst / Programmer role. Unfortunately, whilst agile methods might say a Developer is very similar to an Analyst / Programmer, the fact is that developers are known for their programming skills, not their testing or management skills. There is a resistance in Developers generally to perform tasks other than programming. The industry sees Developers as programmers, and not much else. The term Analyst / Programmer at least tells lay-people that these roles are broader than only programming. I propose a T-model in which all new entrants to the industry go through an Analyst / Programmer apprenticeship track for 8-or-so years before specialising.

- *Thirdly*, to support apprenticeship, IT organisations (which is most organisations these days) need to commit themselves to building capability and experience, not prostitute themselves to billable hours. IT companies need to invest in their people; allow bench time and turn it in to training. Target your workforce to using down time wisely. Build in apprenticeship and qualifications to your graduate career paths. In fact, hire graduates in the first place and train them up! Acknowledge that building capability and experience is going to cost you, but realise that this is necessary, because gone are the times that you can just poach skilled staff from other places: the staff you desire are dwindling in supply.

(c) Martin Hunter, ArcWorx LTD., April 2017

- *Fourthly*, we need as an industry to value experience in software programming. We need to support career paths into programming as a valued specialism, as much as other specialisms like managers, architects and analysts. We need to start seeing “Principal Developer” and “Consultant Developer” job roles in our teams and organisational charts.

(c) Martin Hunter, ArcWorx LTD., April 2017