DIGITAL ALL THE WAY DOWN: INNOVATION IN THE SOFTWARE INDUSTRY

Matthew Jones, Judge Business School, University of Cambridge, UK, mrj10@cam.ac.uk

Jeremy Rose, University of Skövde, Sweden; jeremy.rose@his.se

1. INTRODUCTION

The call for papers for this workshop defines digital innovation as the application of digital tools, technology and digital infrastructure in new products, services, and business models that offer customers enhanced or unique value. While such a broad definition may be appealing in embracing a broad range of innovation practices, it may be argued to conflate two distinct types of innovation: those that bridge digital and physical domains and those that operate solely in the digital domain. An example of the former might be innovations associated with the digitalisation of automobile control systems in which digital technologies enhance the capabilities of a physical product. An example of the latter might be innovation in the software industry, in which the product itself is digital. In this paper we will review the literature on the characteristics of digital artefacts to try to identify the ways in which innovation practices may differ between these two types. Drawing on evidence from 18 small and medium-sized software enterprises in a specific geographic locale, we will explore the extent to which innovation in these firms may be seen as reflecting the particular characteristics of digital products. It will be argued that innovation of digital products, such as software, may be considered as a special case of digital innovation.

2. DIGITAL INNOVATION

The characteristics of digital products

There have been several attempts to define the distinctive characteristics of digital products in general. Thus Quah (2003), from an economics perspective, identifies digital goods as exhibiting non-rivalry (their use by one person does not degrade their use by others – multiple individuals can watch the same YouTube video simultaneously and the constraints on this arise not from the product itself, but from the hardware used to access it), infinite expansibility (the quantity of digital goods can be made arbitrarily large, arbitrarily quickly at no cost – any costs are again attributable to the medium on which the digital product is borne (Falkner & Runde, 2013) rather than properties of the good itself), they are (initially) discrete (they are indivisible – half a software program is no software program at all); aspatiality (they are everywhere and nowhere at the same time) and recombination (new digital goods that are created through the combination of antecedent digital goods may have features not present in the original goods).

Yoo et al (2010a:8) identify "seven material properties that differentiate digital artefacts from their non-digital counterparts". They list these as reprogrammability (they can accept new instructions to alter their behaviour); addressability (individual digital artefacts can be uniquely identified within a particular setting); sensability (they can sense and respond to their surroundings); communicability (they can send and receive messages with other digital artefacts); memorizability (they can record and report their previous states); traceability (they can identify and store events and entities over time) and associability (they can be related to and identified with other entities enabling inferences to be drawn about future states). Digital materiality, in turn, is identified as one of six "dimensions of digital innovation" (Yoo et al, 2010a:11), with the others being convergence, generativity, heterogeneity, locus of innovation, and pace, although the relationship between material properties and the digital innovation dimensions is not elaborated. An alternative list of characteristics of digital technology is provided in Yoo et al (2010b) that emphasises reprogrammability, homogenisation of data (any digital content can be stored, transmitted, processed and displayed using the same device and heterogenous content can be seamlessly combined to deliver new services) and self-referentiality (digital innovation requires digital technology). This paper also emphasises that these digital characteristics "pave the way" for a layered, typically modular, architecture of digital products characterised by standardised interfaces between components (Yoo et al, 2010b, 726-727)

In their own review of literature addressing the properties of digital artefacts, Kallinikos et al (2013) characterise them, (based on Kallinikos et al (2010)) as exhibiting: editability (pliability and openness to modification), openness (they are modifiable by programs other than those that govern their own behaviour), transfigurability (they do not present themselves as clear cut, distinct and enduring objects) distributedness (akin to Quah's aspatiality – they may exist in more than one place at once) and interactivity (they are open to different actions depending on the user's choice)

Two papers that specifically consider the properties of software, as a particular type of digital artefact, are those of von Englehardt (2008) and Boudreau (2012). Focusing on how these properties affect the economics of software, von Englehardt (2008) draws on Quah (2003), but proposes that software possesses a somewhat different set of properties from digital goods in general, namely that in addition to discreteness, recombination and aspatiality (renamed as intangibility), software exhibits network effects, that is the utility of software products increases with the number of users. Software, von Englehardt argues, is also an information good in that its value cannot be determined prior to a transaction unless the vendor discloses the information e.g. by providing a trial version.

Perhaps most relevant to this paper, Boudreau specifically addresses innovation among software app developers and seeks to relate these to "several important properties of application [software] development that should at least shape the broad outlines of the innovation process in this context" (Boudreau, 2012: 1411). A number of these, such as network effects and recombination, have already been referred to, but Boudreau also identifies "virtually infinite product space", "uncertainty and skewed outcomes" and "low cost facilitated development". The first of these refers to the breadth of development possibilities, which Boudreau connects with Zittrain's concept of generativity (Zittrain, 2007) – a capability for expansive novelty. Thus a particular software program may be developed in almost unlimited ways. Boudreau also suggests, citing Stoneman (2010), that this positions software among a class of goods that supports a type of "soft" innovation that is distinct from traditional industrial innovation.

Uncertainty and skewed outcomes follow from this infinite product space and network effects. Where product possibilities are unbounded and there are increasing returns to network size, many new products may be developed, but only a few will succeed. These conditions further encourage the adoption of platform strategies in which potentially significant investment in an initial infrastructure serves to support the low-cost proliferation of a large variety of products that can deliver both flexibility and economies of scale.

A variety of, partially overlapping, characteristics of digital products in general and software in particular, may therefore be identified that may be significant in shaping innovation practices in the software industry, as summarised in Table 1.

	Quah (2003)	**Yoo et al (2010b)**	**Kallinikos et al (2013)**	**VonEnglehardt (2008)**	**Boudreau (2012)**
Focus	*Digital goods*	*Digital technology*	*Digital artefacts*	*Software*	*Software apps*
	Non-rivalry	Reprogrammability	Editability	Network effects	Network effects
	Infinite expansibility	Homogenization	Openness	Information good	Virtually infinite product space
	Discreteness	Self-referentiality	Transfigurability	Discreteness	Uncertainty and skewed outcomes

	Aspatiality		Distributedness	Intangibility	Low cost facilitated development
	Recombination		Interactivity	Recombination	Recombination and reuse

Table 1: Characteristics of digital products

3. SOFTWARE INNOVATION PRACTICES

As part of a broader study of software innovation, interviews were conducted with experienced software developers working for small and medium-sized companies in and around Cambridge, England. This area, colloquially known as the "Silicon Fen", is recognised as a "cluster of creativity" (Koepp, 2003). The Cambridge cluster map[1] currently lists 332 information technology (IT) and telecommunications companies. Of these, however, only six report more than 250 employees or revenues of more than €50M.

Over 100 of these companies were contacted at random and invited to participate in the study. Of these, 18 companies responded positively and a total of 25 in-depth semi-structured interviews were held with company CEOs and software developers across these companies. The companies ranged in size from 3 to 120 people, and produced many different kinds of software, from companies working primarily with a single packaged product to companies developing many different products for clients, as shown in Table 2.

Code	**Area**	**Interview**
CompanyA	SME Enterprise software	1
CompanyB	Online visualisation of clothes on 3D models	2
CompanyC	Examination and survey software	3
CompanyD	Design services	4
CompanyE	Home digital control systems	6
CompanyF	Engineering and scientific software consultancy	7

1 http://www.camclustermap.com

CompanyG	3D Rendering software	8
CompanyH	Smart cards and security products	9
CompanyI	Transport logistics	10
CompanyJ	Seismic analysis software	11, 12, 13
CompanyK	IP Television	14, 15
CompanyL	Text-mining software	16
CompanyM	Recruitment Agency management software	17
CompanyN	Natural language search	18
CompanyO	Embedded control and monitoring solutions	19, 20, 21
CompanyP	Genome analysis	22, 23
CompanyQ	Custom sensors	24
CompanyR	Cognitive Impairment evaluation software	25

Table 2: Types of software produced by companies studied

All interviewees were specifically invited to choose a recent example of an innovation in their company and to describe what this involved and how it came about. Interviewees were prompted to consider innovation as consisting of two types: innovation in a product or service and process innovation and to select whichever seemed the most significant in their company. Following transcription, the interviews were independently coded by the two authors to identify the characteristics of software that contributed to the innovation process.

4. SOFTWARE CHARACTERISTICS AND SOFTWARE INNOVATION

Many of the innovations reported by interviewees could be seen as deriving from the layered modular architecture of software. This meant that innovation frequently occurred "behind the scenes" in architectural layers that were not necessarily visible to the end user. Indeed their effect was often to make these layers transparent to the customer such that they no longer constituted a barrier to the higher-order task they wished to carry out. At the lowest modular level this could involve the development of hardware abstractions that enabled software to be ported across multiple platforms.

> "Yeah, so, we have developed a software stack over the last 10 years, which we have basically developed a hardware abstraction layer that is reasonably thick and that abstracts our main core software stack and applications away from the hardware platform. In doing that we have been able to port fairly smoothly and quickly to IBM Power PC, Texas Instruments, Sigma, SD Micro, Intel, etc. etc. [K15]

This offers the potential for not just product innovation (supporting new hardware), but also process innovation

> "So we can develop the software on the PC, debug it, the tools are so much better and it's quicker, and it's the base platform for everything. We then … when the customer wants the applications modifying, one chap can be modifying the applications through the PC while the other chap is taking some horrible piece of hardware emulation kit and building the hardware abstraction layer on it with no concern about the applications yet, and then the two pieces can come together and it can all fly again. We'll hand over to the customer the software for the target platform." [H9]

At higher modular layers, innovation in some companies was focused on support for multiple execution platforms. Sometimes this reflected the complexity of their product, as it had evolved over time drawing on different domains of programming expertise:

> "most of this computer vision stuff is written in C++ against Windows APIs. Most of the middle tier code was written in Java and Spring and simply through a process of needing to hit deadlines continuously, this is the way things stayed. So we've got a technology base where we're supporting three different execution platforms in that we have tens and tens of codeblocks of JavaScript code running in the browser, that's the rich client that's our front-end application, we have JVM code that's running on middle tier and database ops and Dot Net code and Windows code that's running the visualizations" [B2].

Other firms supported multiple platforms as a way to hedge their bets on emerging industry standards:

> "But we recognized that the industry was beginning to get polarized into other unit needs, Windows' needs. We didn't really know where that was going to go, where it was going to end up. So we adopted the QT to give us that cross-platform development framework, so that anything we developed for one Linux type environment could very easily be made to Windows" [J11].

Or used modularity to enable the same software to be deployed on multiple infrastructures:

> The workflow system has been designed to be very flexibly deployable. Actually we can deploy the identical workflow on Amazon web services. We can do it on a various flavours of high performance compute, so this is Condor or SG or what have you. Whatever is scheduled, it works, it's very flexible" [P22].

Modular architecture also supported innovation in a number of firms who provided a front-end to third-party products:

> "we knew for example from the beginning that supporting Accelrys Pipeline Pilot was vital because we had already had to provide an integration for customers without having support that really would enable us to do it well. … Because our customers are pharma companies doing science, they're often using these science

> workflow technologies because they're often taking science data from essays and literature searches, doing a lot of filtering on them and then charting and looking at visualizations of them. It provides a very concrete socket that you have to plug into" [P23]

or who facilitated the integration of products between two third parties:

> "We work with the developing department in Oslo and we kind of say, "This would be nice for our customers to be able to do." Then when we come to the integration bit, that is really a process we control. We don't write the code, thus we don't know anything about the coding, but it is the platform. It's basically something that's called Connect My Apps, and we then basically take the technical lead there that this is what we want. Quite often we do the research, so we kind of say, "We want to integrate through this application and here is the API for that"" [A1]

While modularity may be a distinctive feature of digital innovation, not all firms had initially designed their products to take advantage of it and a number had found themselves needing to undertake a significant revision of their product architecture to simplify maintenance and development of a proliferating portfolio of products.

> "Historically CompanyC has developed their own core products and sold those, but customers have come to us and said can you add a report to do XYZ and can you do this, can do you do that, which is more of the bespoke software built on our core technology, and we have done that. I think we've ended up in a bit of a mess due to poor software development because we've now got way too many products and we've struggled to support them all....[Consequently] We're trying to develop this new product, and we're designing that in a way, so if a customer does come to us and they want something bespoke it will be an add on module to the core products. They will be part of the core products; they won't be a separate product. We update the core product, the bespoke product gets updated. It makes it an awfully lot easier for us to maintain and extend product". [C3]

Or to open up their product as customers' requirements evolved

> "In our case we wanted to move the product from being one that was purely used interactively to one that we provided an API to that exposed product to be driven by other pieces of software embedded in workflows. That was a big step for our software because it was kind of designed purely; well primarily as an interactive one. That was its unique selling point. We had to begin the process of chopping it up and rearranging it.... We had to separate the presentation aspect from the underlying logic … once the logic was separate from the GUI, we could move the logic from the client program where it had been stuck and move it in to our server program where it could be exposed to other clients". [L16]

Innovation was not solely associated with layered modular architecture, however, but drew on more basic characteristics of software, such as recombination. One senior developer identified writing code in ways that would enable its future reuse as a key aspect of software innovation.

> "our software still has lines of code in it I guess that were written seven or eight years ago but they may be in complete different modules than what they were in before. They may have completely different interfaces

> than what they were in before. The information needs to be flexible enough in order to be maintainable down the line. Otherwise it crystallizes into one thing and you can do nothing with it". [G8]

While recombination could improve efficiency of software development, however, it could also be a source of problems, meaning that further process innovation was necessary to manage interdependencies.

> "it's testing that we're not breaking things as we're doing it, right? For example, we might make some change here for trial, like you do to solve the problem, not thinking that it could actually have an impact somewhere else, not just in the same product, but because the products are sharing technology across. You know, we could do something on [ProductA] which could affect [ProductB], not really realizing that's what we've done, you know. So having sort of tests that could check that things are … that we're not upsetting anything could be something we need to do better". [J11]

Although recombination could be applied to a company's own code, software innovation in some companies was focused on the development of products using Open Source code.

> "The software that we depend upon, the building blocks of what we do are algorithms that have been developed mainly in academic settings and are mainly released under open source licenses. What we do as a consultancy business is take these components and put those together, in particular ways to build workflows that fulfil a particular data processing task.... That was a very much how we've been operating and what you're really doing is you are taking open source software, you are productizing, you are commercializing it, you are supporting it". [P22]

Conversely for other companies, rather than recombination, it was the decomposability of software afforded by its modularity, that was a source of process innovation:

> "Yes, individual developers can go off and do what would have been called a development branch. Say we're looking at an individual feature, maybe speculative, maybe experimental or maybe just destabilizing and we don't want it published back to the central place too soon. These new distributed source code control systems, one their spinoff benefits is that effectively you're always working in a branch. That's what it's like and so a developer can break up the work into ten chunks and they're all unpublished or private until they say "I'm done". All those ten commits are all published at the same time. Then you're not forcing the developer to publish intermediate pieces of work" [L16]

The decomposability of software combined with traceability could also enable real-time experimentation, making changes to the way the product is configured for particular consumers and measuring the effect on performance:

> "During that time we've been measuring the product's performance against conversions in terms of selling more clothes and returns. Making small changes and seeing whether they have any impact on consumers and keeping that trick that we've defined". [B2]

Alternatively traceability could be used for post hoc analysis of customer behaviour to identify potential product improvements:

> "We've got huge analytics in the back that identify when the tool is less than successful. It's up to the human being to look through that, to say, "How can we improve that knowledge base." So, back in the office we have a knowledge engineer whose job it is to look over customer data and see how can you improve that specific customer's knowledge". [N18]

Sometimes, however, software characteristics were seen as making it harder for customers to recognise software innovation, leading to more attention being paid to the look and feel of a product than to how well it actually performs the task for which it was designed.

> "Absolutely, I'm very conscious that a judgment is made of a product by what it looks like from the outside, you can't get away from that. Just because software is not something that you can touch, then you could argue then that it's even more important that the way it looks, because that is the only judgment you've made. At least with a bit of hardware you can look at the outside and it might look great and lovely and blue led's and all this sort of thing …. Then you lift the lid and you look inside and it's a complete mess. You look at that, you look at how it was wired and stuff and you make a judgment, well this is rubbish I wouldn't like that. You can't do that with software, you can't open the lid, certainly at client level they would probably never look to open the lid. Even another software engineer might well open the lid on perhaps a syntactical language and it would be very difficult, certainly initially, to engage". [F7]

This could make it important to find ways to give customers experience of the product.

> "I believe if you put a working system in front of an end user you'll get far more feedback than presenting them on the other side of that coin with a hundred page document with no diagrams. It's better to have something visual and that might be a document with diagrams, but it's certainly better just for them to actually visualize a working product even though it's a prototype, it's not a real product. Users will buy into something which visually works and they'll give you feedback in a way you would never get it from a document." [M17]

Similarly software components that might not be a viable product on their own could be made available to customers to promote awareness of the company.

> "We're not looking to replace something that they already do; we're just looking to fill some middle ground that lies between the two products. It might well be that the product itself is not necessarily, certainly in its smallest form, is not actually a saleable product. It's a giveaway, it's something that we effectively sponsor, but whereby our name becomes known by people." [F7]

The intangibility of software, however, was also seen as making it difficult for customers to appreciate the value of documentation and quality of code:

> "The underlying reason is commercial in that a lot of the time, especially projects that we get brought on, external contractors get hired and we are charging a day rate, which is expensive compared to a full-time employee. The customers generally want results as fast as possible and they don't care about any other parts. Essentially what happens is, the code gets written and then the phase where you're supposed to document everything that's happened, they don't care about that afterwards because they can see it working so why would they pay you for a few extra weeks for you to write all the documentation and make it nice. This happens ten years ago and then we inherit the projects and then we have to reverse engineer everything which of course costs them more then but then they don't see that". [O20]

In some situations, however, the intangibility of software could itself be the source of innovation, for example in providing software as a service.

> "We have a flagship product which is the text mining engine and customers can either buy and install that themselves or they can rent the use of it from us over the internet. We host it and they use it as a software as a service. In that case we also built some indexes for them; the world's patent data and Medline which is a biomedical corpus and also clinical trials.gov. We kind of become a content provider as well as a software provider." [L16]

Or in enabling process innovation, such as distributed development:

> "Actually, I'm thinking specifically about a specific class of tools, which are the software development process development tools, so these are very specific tools. The first one is we're in the UK, we're working with a US SME closely to develop this thing. We need to create a specification for this thing; we need to be able to share documents for example, in a fairly secure way. We use Google Docs platform to share these documents and provide the specifications, so that's one example of software. It's much easier to do that than it is to do attachments by email, it's much easier just to work from these documents." [P22]

It could also enable interaction with customers to understand their requirements.

> We have a tool that has helped us a little bit with that recently and we should make more of this. It's interactive webinars. It's Net webinars. Particularly, software we use as going to a meeting. It's a Citrix product. Whereby, we don't have to install software on their network for them to see it. We don't have to go to their offices. They don't have to come to our office. We just meet in the cloud. I run the software on my computer, on my screen. They click on a link that allows them to view my screen and I can show them the software working. We're talking on headsets with microphones. If we want, we can even give them control of the software on our computer so they can then play with it, experiment and give us some feedback. That is a tool that has helped us quite a bit with talking to our busy clients" [J12]

Finally, in some of the companies involved in the creation of software products a particularly tight form of self-referentiality was significant in software process innovation in that the companies developed their own software tools to support their product development. One company, for example, had developed their own auto-tester to find bugs in their product (before they affected their clients).

> "When I was first starting working on the engine, we had a lot problems with the stability of it. Part of the reason for that was that a lot of functionality had been added without setting up sufficient tests or indeed many tests at all to prove that either each part works or that when you try to do everything all at once everything still works together well. There was an idea around and I took it on to write a testing application that would start to find things with the engine. We called that the auto-tester. The idea it was that it would just send inputs to the engine continually until the engine had a problem. That became an automated way of finding bugs in the engine". [I10]

Others carried out similar functions with third party software:

> "We've now moved to a tool called Build Bot which is an open source tool for centrally managing your builds and it means that the developer can actually; this is another sense in which it fosters experimental

work. As well Build Bot managing published work and building daily or hourly, whatever you like, it will also build stuff you've just sent just to make sure it runs on all platforms. You can also ask it to do a trial build on something you haven't even published." [L16]

In some companies this had been extended to building the software itself out of the tools they had developed.

In fact the latest version we've built the tools and we've built the product out of the tools meaning that when we have to maintain it it's the same tools that are tried and tested and that gives us unique flexibility compared to our competitors who every time a customer needs a change the developers have to open up their compiler and they have to do things and it costs them a lot of money." [M17]

The influence of software characteristics are therefore felt in both product and process innovation as summarised in Table 3.

Characteristic	**Product innovation**	**Process Innovation**
Modularity	Hardware abstraction Platform independence Add-on to 3rd party product Integration of 3rd party products	Hardware abstraction Source code control
Modularisation	Open product to 3rd party products	Facilitate product maintenance
Recombination	Productization of Open Source code	Code reuse
Traceability	Real-time experimentation Post-hoc analytics	
Intangibility	Invisibility of innovation Importance of experience Creating awareness Software as a service	Distributed development Remote requirements elicitation
Self-referentiality		Auto-testing Building software out of tools

4. SOFTWARE INNOVATION AS A SPECIAL CASE?

Before considering whether the characteristics of software, as an exclusively digital technology, are associated with forms of innovation that are different from those associated with digitalised artefacts (where an existing or new physical artefact includes some digital component, such as a digital instrument panel in a car), it would seem important to clarify that the software component of a digitalised artefact is itself exclusively digital (and so may reflect the characteristics of software innovation discussed here) and, conversely, that software products, of the sorts produced by the companies interviewed in this study, may also be components of digitalised artefacts. The argument therefore applies to innovation of software in general, whether as a standalone product or as a component of a digitalised artefact. It also applies specifically to the non-physical bitstrings (sequences of 0s and 1s) that represent both the logic of a software program and the data on which it acts. Following Faulkner and Runde (2013) we therefore distinguish between these bitstrings and any physical artefact on which these bitstrings may be stored, such as a CD or a server drive. This is not to suggest that these "bearers" may not be consequential for software innovation practices, rather we seek to identify those aspects of software innovation that relate to the specifically digital character of software, independent of any characteristics associated with its particular bearer in any given circumstance. So, for example, while it might be the case that innovation of a particular piece of software may be affected by the speed with which it is able to be run on the particular hardware configuration available in a certain setting, this would not be considered, for the purpose of this analysis, to be an effect attributable to its specifically digital character (although the creation of a new algorithm that would enable it to run faster on any hardware configuration would be attributed to its specifically digital character).

One immediate implication of considering software as non-physical bitstrings relates to its intangibility. As interviewees suggested this can make software innovation hard to recognise. Consequently this places a particular emphasis on creating opportunities for customers to experience software products, for example through providing prototypes or trial versions, or even giving away components for free. While this may also be the case with physical products, such as offering a test-drive of a car, the lack of alternative evidence of software performance means that awareness of software innovation is hard to achieve other than through demonstration.

At the same time, however, the aspatial character of software means that this experience can be made available to customers wherever they are located. Indeed, with Software as a Service, the same may apply to the software product itself. Aspatiality may also contribute to process innovation, for example through distributed development and remote requirements elicitation. The immediacy and low cost of customer feedback this offers can facilitate more efficient and potentially more rapid innovation. Combined with traceability and memorizability, moreover, this can provide firms with a hitherto unobtainable level of detail on how customers actually use their products in practice and the causes and conditions of any faults. While this may not, in itself, necessarily drive new innovation practices it constitutes a resource for the creation of rich insights on product performance that may inform innovation decisions.

The modular character of software and the layered modular architecture that builds on it creates a perhaps unprecedented plasticity of software design. A product can be delivered in very different ways and constructed from very different components. This applies both within the software product, for example integrating components with very different characteristics, and in the ability of that product to be delivered across many different software and hardware environments. The integration of diverse components is, of course, not confined to software products, but the ability to combine different execution platforms, more or less seamlessly, without this being visible to the customer, is enabled by software's intangibility. The customer may only ever encounter the supported services through the user interface, and the software elements that deliver these services may be unobservable, and, even if examined, inscrutable. This means that innovation may occur without customers' awareness (especially if it can be carried out remotely). Although seemingly disadvantageous to the software supplier, in that innovations may not be recognised by customers, it also offers suppliers considerable control over what they reveal to the customer (or what the customer can detect of the product's quality) and over the value they may accrue, for example from data they acquire from customers' use of their product.

The intangibility of software also means that for the customer, the product they acquire is the specific presentation of services that are delivered by the particular software and hardware environment employed in their setting. How it looks, the speed with which it runs, the linkages with other services may all vary from installation to installation. While these differences may be relatively minor, and perhaps not significant to the activities that the software supports, it does mean that software has a greater plasticity than physical products, with its particular form only being realised in the conditions in which it is run. This is the case too with physical products – a car's speed may be constrained by road conditions or the sound of an audio recording by the equipment it is played on, but with software there is no external referent. It is what it is, as it is instantiated in a particular environment. This relational character of software makes innovation more difficult as its manifestation may be context dependent, but at the same time frees it from some of the material constraints that may apply to physical products. Radically new ways of doing things may be possible with different configurations of the same basic resource of bitstrings.

That all software is built from the same "stuff" also makes innovation self-referential – software is produced by, and out of, software. This means that companies can build the tools with which they build and test their own products and creates niches for companies with the requisite expertise to make new products from the integration of 3rd party products (whether closed or open source). Furthermore the data on which software acts are bitstrings too, so can be integrated, manipulated and controlled with the same tools. The scope of software innovation is thus potentially considerably more extensive than would be the case with physical products. Software developers can work not just on the products they deliver, but on the tools that they build them with and the material on which they act.

Software innovation may therefore be considered as potentially a special case of digital innovation in which the distinctive characteristics of software as a specifically digital product may be seen as enabling particular types of innovation practices that are arguably not possible with digitalised products. Although, to some extent, this is an artificial distinction, since the characteristics apply too to the software component of digitalised products, it would seem helpful to consider the specific innovation characteristics of software both as a product in its own right, but also as a contribution to digital innovation more generally.

5. CONCLUSIONS

The intangibility and malleability of software makes it difficult to make definitive claims about software innovation practices in all settings and for all types of software. In keeping with the argument above, the character of software innovation reflects the realisation of a space of possibilities, created, but not determined, by abstract characteristics of digital products, rather than specific outcomes. Whether or not the enactment of these possibilities in particular settings constitutes (or is perceived as constituting) significant innovation may depend on the characteristics of the context as much as those of the software itself and on customers' awareness of how outcomes are achieved, as so much of software may be both invisible and incomprehensible to customers. What the evidence of innovation in these small and medium software companies suggests, however, is that the space of possibilities provides a variety of opportunities that they are able to exploit commercially.

In terms of the specific characteristics of digital products explored in the introduction to this paper, the analysis presented here suggests that these currently do not provide a sufficiently complete and consistent lens with which to study software innovation and that closer attention to the particular characteristics of innovation of purely digital products may be worthwhile. This lack of consistency on the characteristics of digital products may reflect differences in their conceptualisation of digital products as economic goods (Quah, 2003) or as a particular class of technology (Yoo et al, 2010b) or artefacts (Kallinikos et al, 2013) that may, or may not, apply to software specifically. Even where discussion of characteristics is addressed solely to software, the focus of Von Engelhardt (2008) and Boudreau (2012) is as much on the characteristics of the market as a whole, such as network effects, ease of entry and skewed outcomes as on those of software specifically. From the point of view of individual companies engaged in software development, moreover, these market characteristics may be considered secondary to their own innovation practices. They may affect the eventual commercial success of their products, but they do not shape the nature of these products directly as they are developed. We therefore still lack a satisfactory account of the characteristics of software as a class of digital products.

Given the small number of organisations involved in this study and their specific geographic location (although the aspatiality of software might be argued to mean this should have less influence) it would not seem appropriate to suggest that the findings provide a complete picture of software innovation, even among small and medium-sized enterprises. Nevertheless it would seem possible to make some preliminary observations about the ways in which innovation in software may be influenced by its specifically digital character. Thus modularity, aspatiality/intangibility, recombinability and self-referentiality would seem to be identified as making the greatest contribution to software innovation. Other characteristics, such as traceability, seemed to play a more minor role for specific products.

Although the absence of reference by interviewees to other characteristics mentioned in the prior literature, such as openness or discreteness may reflect the study's sampling bias, it could also reflect the different focus of these earlier papers whether in terms of their disciplinary base, such as economics (Quah, 2003; Von Engelhardt, 2008), their conceptualisation of the phenomenon of interest, i.e. digital artefacts (Kallinikos et al, 2013) or digital technology (Yoo et al, 2010b) rather than software, or their level of analysis (market, rather than individual firm (Boudreau, 2012)). At the very least, therefore, there would seem a need for further work to clarify the differences and commonalities between these various attempts to characterise digital products in general and software in particular and to understand how these may be related to digital innovation.

ACKNOWLEDGEMENTS

The Danish Research Council supported this research work: grant no. 12-133180. We offer many thanks to the interviewees and the participating companies.

REFERENCES

BOUDREAU, K.J., (2012). Let a thousand flowers bloom? An early look at large numbers of software app developers and patterns of innovation. *Organization Science*, 23(5), pp.1409–1427.

ENGELHARDT, S.V., (2008). The economic properties of software. *Jena Economic Research Papers*, 45. Available at: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1430885

FAULKNER, P. & RUNDE, J. (2013) Technological objects, social positions, and the transformational model of social activity. *Mis Quarterly,* 37**,** 803-818.

KALLINIKOS, J., AALTONEN, A. & MARTON, A. (2013) The ambivalent ontology of digital artifacts. *Mis Quarterly,* 37**,** 357-370.

KALLINIKOS, J., AALTONEN, A. & MARTON, A. (2010). A theory of digital objects. *First Monday;* 15, (6 - 7)

KOEPP, R. (2003). *Clusters of creativity: enduring lessons on innovation and entrepreneurship from Silicon Valley and Europe's Silicon Fen*. Chichester: John Wiley & Sons.

QUAH, D. (2003) Digital goods and the new economy. Available at: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=410604

STONEMAN, P., (2010.) *Soft innovation: economics, product aesthetics, and the creative industries*, Oxford University Press

YOO, Y. ET AL., (2010a). The Next Wave of Digital Innovation: Opportunities and Challenges: A Report on the Research Workshop'Digital Challenges in Innovation Research'. Available at: http://papers.ssrn.com/sol3/Papers.cfm?abstract_id=1622170.

YOO, Y., HENFRIDSSON, O. & LYYTINEN, K. (2010b) Research commentary-The new organizing logic of digital innovation: An agenda for information systems research. *Information Systems Research,* 21**,** 724-735.

ZITTRAIN, J., (2007). Saving the Internet. *Harvard Business Review*, 85(6), pp.49–59.