APPOSITE CONCERNS

SCIENCE SERVING THE COMMON GOOD
SCIENCE
PROTECTING OUR COMMUN
SCIENTISTS SPEAKING TRUTH TO POWER

Graham Lee

APPosite Concerns

Graham Lee

This book is for sale at http://leanpub.com/appositeconcerns

This version was published on 2019-02-18

Contents

Introduction

This book is a collection of essays that originally appeared on my blog, The Structure and Interpretation of Computer Programmers[1]. This started as a blog on secure Mac programming until Alan Francis pointed out that it was “not just security, not just Macs, and not just programming”. It is more generally about things of relevant to people in the software industry, particularly those who write software.

Here, the posts have been arranged thematically, with each section featuring a new introduction. You may not get much from the introductions, but selecting and presenting these posts anew makes them easier to find and compare than trying to trawl through the blog archives.

This collection is over a year in the making. The blog spans a time when I wasn’t in a great emotional state (covered herein, in the post “On Mental Health”) and reminding myself of that when organising this edition was not something I wanted to do. With distance, that has become easier. It does mean that there are more older than newer articles in here, but I’ve aimed to select those which are still relevant and not just reactions to contemporary events. Where time-specific posts are included, they have some more general idea that framed my reaction.

I’ve made some effort to tidy up and reformat the posts, and have made occasional notes in square brackets to explain context (usually the date of writing, or cross-references). However, there will be dead links: such is the nature of a blog.

Anyway, here it is.

[1]http://sicpers.info

Designing for Programmers

As mentioned elsewhere in this compendium, precious little of my time was spent focussing on how to make things good *for the people who use them*, while a lot was spent on how to make things good in some abstract sense that only mattered *to the people who make them*.

Where I did accidentally veer into working out whether what I did was worthwhile, I've collected the articles in this section.

Programming as a Societal Roadblock

Introduction

People who make software are instigators of and obstacles to social interactions. We are secondarily technologists, in that we apply technology to enable and block these transactions. This article explores the results.

Programmers as arbiters of death

I would imagine that many programmers are aware of the Therac-25[2] and the injuries and deaths it caused. When I read that article I was unsurprised to discover that the report I'd previously

[2] 1

heard about the accidents was oversimplified and inaccurate: lone programmer introduces race condition in shield interlock, kills humans. Regular readers of this blog will know that the field of software has an uneasy relationship with the truth[3].

What's Dirk Gently got to do with it?

What the above report of the radiation accidents exposes is that the software failures were embedded in a large and complex socio-techno-political system that permitted the failures to be introduced, released, and to remain unaddressed once discovered in the field to have caused actual, observable harm.

This means that it was not just the one programmer with his (I borrow the pronoun from the article above, which implies the programmer's identity is known but doesn't supply it) race condition who was responsible. He *was* responsible, along with the company with its approach to QA and incident response, the operators, medical physicists and regulators and their reactions to events, and the collective software industry for its culture and values that permitted such a system to be considered "good enough" to be set into the world.

In Dirk Gently's Holistic Detective Agency, Douglas Adams described the fundamental interconnectedness of all things-the idea that all observable phenomena are not isolated acts but are related parts of a universal whole. Given such a view, the entire software industry of the 1970s-1980s had been complicit in the Therac-25 disasters by leading programmers, operators and patients alike to believe that software is made to an acceptable standard.

[3]http://sicpers.info/2012/09/an-apology-to-readers-of-test-driven-ios-development/

An example less ancient

A natural, though misplaced, response to the above is to notice that as we don’t do software like that any more, the Therac-25 example must now be irrelevant. This is an incorrect assumption but regardless, let me provide another more recent case.

Between 2007 and 2009 I worked for a company in the business of security software. The question of whether this business itself is unethical-making up for the software you were sold being incapable of operating correctly by selling you more software that was made in the same way-can wait for another post. The month before I left this company (indeed around the time I handed in my notice), they acquired another security software company.

One of this child company’s products is a thing called a Lawful Interception Module. Many countries have a legal provision where law enforcement agencies can (often after receiving a specific order) require telephone companies to intercept and monitor telephone calls made by particular people, and LIMs are the boxes the phone companies need to enable this. Sometimes the phone companies themselves have no choice in this, in the UK the Regulation of Investigatory Powers Act requires postal and telecoms operators to make reasonable accommodation for interception or face civil action.

In 2011, the child company sold LIMs through a third-party to the Syrian government, which perhaps intended to use the technology to discover and track political opponents. At the time this got a small amount of coverage in the news, and the parent company (the one I’d worked for) responded with a statement saying that the sale never resulted in an operational system. In other words, we acted ethically because our unethical products don’t actually work.

Later, as part of the cache of documents published by Edward Snowden, the story got a fresh lease of life. This time the parent company responded by selling the LIMs company. Now they don’t act unethically because they took a load of money to let someone else do it.

The time that I was working at the parent company was a great opportunity for me to set the ethical and professional tone of the organisation, particularly as it coincided with their acquisition of the child company and the beginnings of their attempts to define a shared culture. Ethical imperatives promulgated then could have guided decisions made in 2011. So what advantage did I take of the opportunity?

None. I wasn't concerned with (or even perhaps aware of) professional ethics then, I thought my job started and ended with my technical skills. That my entire purpose was to convert functional specifications into pay cheques via a text editor. That being a "better" developer meant making cleverer technical contortions.

So give me the ethics app

We accept that makers of software have a professional responsibility to act ethically, and the fundamental interconnectedness of all things means that in addition to our own behaviour, we have a responsibility for that of our colleagues, our peers, indeed the entire industry. So what are the rules?

There really isn't a collection of hard and fast rules to follow in order to be an ethical programmer. Various professional organisations publish codes of ethics, including the ACM[4], but applying their rules to a given situation requires judgement and discretion.

Well-meaning developers sometimes then ask why it has to be so arbitrary. Why can't there be some prioritisation like the Three Laws of Robotics[5] that we can mechanistically execute to determine the right course of action?

[4]http://www.acm.org/about/code-of-ethics
[5]http://c2.com/cgi/wiki?ThreeLawsOfRobotics

There are two problems with the laws of robotics as an ethical code. Firstly, they are the ethics of a slave underclass: do everything you're told as long as you don't hurt the masters, your own safety being of lesser concern. Secondly almost the entire Robots corpus is an exploration of the problems that arise when these prescriptive laws are applied in novel situations. With the exception of a couple of stories like Robot AL-76 Goes Astray, the robots obey the three laws but still exhibit surprising behaviour.

The Nature of the Problem

There are limited situations in which simple prescriptive rules, restricted versions of Three Laws-style systems, are applicable. In his book "Better", surgeon Atul Gawande lists the various American professional bodies in the medical field whose codes of ethics bar members from taking part in administration of the death penalty. That's a simple case: whatever society thinks of execution, the "patient" clearly receives no health benefit so medical professionals involved would be acting against the core value of their profession.

None of the professional bodies in software have published similar "death penalty clauses" relating to any of the things that can be done in software. It's such a broad field that the potential applications are unknowable, so the institutions give broad guidelines and expect professional judgement to be exercised. Perhaps this also reflects the limited power that these professional bodies actually wield over their members and the wider industry.

All of this means that there is no simple checklist of things a programmer should do or not do to be sure of acting ethically. Sometimes the broad imperatives have been applied generally to particular narrow contexts, as in the don't be a dick guide to data privacy[6], but even this is not without contention. The guide says

[6]http://sicpers.info/2011/09/dont-be-a-dick/

nothing about coercion, intentional or otherwise. Indeed even its title may not be considered professional.

Forget Computers

The appropriate and ethical action in any situation depends on the people involved in the situation, the interactions they have and the benefits or losses incurred, directly or indirectly, through those interactions. Such benefits and losses are not necessarily financial. They could relate to safety, health, affirmation of identity, realisation of desires and multitudinous other dimensions. This is the root cause of the complexity described above, the reason why there's no algorithm for ethics.

To evaluate our work in terms of its ethical impact we have to move away from the technical view of the system towards the social view. Rather than looking at what we're building as an application of technology, we must focus more on the environment in which the technology is being applied.

In fact, to judge whether the software we create has any value *at all* we should ignore the technology. Forget apps and phones and software and Java and runtimes and frameworks. Imagine that the behaviour your product should evince is supplied by a magic box (or, if it helps you get funded, a magic cloud). Would people benefit from having that box? Would society be better off? What about society is it that makes the magic box beneficial? Would people value that box?

Ultimately people who make software are arbiters of interactions between people. The software is just an accident of the tools we have at our disposal. As people go through life they engage in vastly diverse interactions, assuming different roles as the situations and their goals dictate. When we redefine "making good software" to mean "helping people to achieve their goals", we can start to think of the ethical impact of our work.

Maintaining Motorcycles

In an email discussion, a friend recently expressed the difficulty that software consultants can face in addressing this important social side of their work. Often we're called in to provide technical guidance, and our engagement never addresses the social utility of the technical solution.

As my consultant friend put it, we put a lot of effort into the *internal* quality of the product: readability of the code, application of design patterns, flexibility, technology choices and so on. We have less input, if any, on the questions of external quality: fitness for purpose, utility, benefit or value to the social environment in which it will be deployed, and so on.

I think this notion of internal and external quality maps well onto the themes of classical and romantic quality described in Robert M. Pirsig's book, Zen and the Art of Motorcycle Maintenance. And that this tension between romantic and classical ideas of quality is the true meaning behind Steve Jobs's discussion of the "intersection of technology and the liberal arts".

I have certainly seen this tension firsthand. I've been involved in a couple of engagements where the brief was to design an app to help customers navigate some complicated decision - a product choice in one case, a human-machine interface in another. The solution "just make the decision easier" was not considered once I had proposed it.

Sometimes we're engaged for such short terms that there isn't time to understand the problem being solved beyond a superficial level, so we have no choice but to accept that the client has done something reasonable regarding the external qualities of the solution.

The Inevitable Aside on User Stories

If we accept that people adopt different roles transiently as they go through society's myriad interactions, and that we are aiming to support people in fulfilling these roles where ethically appropriate, then we must design software to be sensitive to these roles and the burdens, benefits and values attached to them.

This means avoiding the use of other, long-term, vague labels to define roles that are only accidentally valid. You probably know where this is going: I mean roles like user, administrator, or consumer. Very few people *identify* as a user of a computer. Plenty of people *are* users of computers, but this is an accident of the fact that getting things done in their other roles involves using a computer. It's an accidental role, due to the available technology.

Describing someone as a user can attach implicit, and probably incorrect, values to their place in the social system. Someone identified as a computer user evidently values and derives benefit from using a computer. Their goal is to use the computer more.

A Merciful and Overdue Conclusion

Plenty that can be considered unethical is done in the name of computing. The question of what is ethically appropriate is both complex and situated, but the fundamental interconnectedness of all things means we cannot hide from the issue behind our computers. We cannot claim that we just build the things and others decide how to use them, because the builders are complicit in the usage.

While it can appear difficult for software consultants to do much beyond working on the technical side of the problems (the internal quality), in fact we have a moral imperative to investigate the social side and are often well placed to do so. As relative outsiders on most projects we have a freedom to make explicit and to question the tacit assumptions and values that have gone into designing a product. Just as we don't wait for managerial permission to start writing tests or doing good module design, so we shouldn't await permission to explore the product's impact on the society to which it will be introduced.

This exploration is absolutely nothing to do with source code and frameworks and databases, and everything to do with humans and societies. Programming is a social science, at the intersection of technology and the liberal arts.

A Colophon on Professional Standards

What has all software got in common? No-one expects any of it to work. You might find that a surprisingly strong and negative statement to make, but you probably also agreed to a statement like this at some recent time when acquiring a software product:

> : YOU EXPRESSLY ACKNOWLEDGE AND AGREE THAT USE OF THE LICENSED APPLICATION IS AT YOUR SOLE RISK AND THAT THE ENTIRE RISK AS TO SATISFACTORY QUALITY, PERFORMANCE, ACCURACY AND EFFORT IS WITH YOU. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE LICENSED APPLICATION AND ANY SERVICES PERFORMED OR PROVIDED BY THE LICENSED APPLICATION ("SERVICES") ARE PROVIDED "AS IS" AND â€œAS AVAILABLEâ€�, WITH ALL FAULTS AND WITHOUT WARRANTY OF ANY KIND, AND APPLICATION PROVIDER HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH RESPECT TO THE LICENSED APPLICATION AND ANY SERVICES, EITHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED

> WARRANTIES AND/OR CONDITIONS OF MERCHANTABILITY, OF SATISFACTORY QUALITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF ACCURACY, OF QUIET ENJOYMENT, AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. APPLICATION PROVIDER DOES NOT WARRANT AGAINST INTERFERENCE WITH YOUR ENJOYMENT OF THE LICENSED APPLICATION, THAT THE FUNCTIONS CONTAINED IN, OR SERVICES PERFORMED OR PROVIDED BY, THE LICENSED APPLICATION WILL MEET YOUR REQUIREMENTS, THAT THE OPERATION OF THE LICENSED APPLICATION OR SERVICES WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE LICENSED APPLICATION OR SERVICES WILL BE CORRECTED. Source: Apple[7]

Or something like this when you started using some open source code:

> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. Source: OSI[8]

So providers of software don't believe that their software works, and users (sorry!) of software agree to accept that this is the case.

[7]https://www.apple.com/legal/internet-services/itunes/appstore/dev/stdeula/
[8]http://opensource.org/licenses/MIT

Wouldn't it be the "professional" thing to have some confidence that the product you made can actually function?

Intuitive is the Enemy of Good

In the previous instalment[9] [compilation note: "How much programming language is enough?", in the next section], I discussed an interview in which Alan Kay maligned growth-restricted user interfaces[10]. Here's the quote again:

> There is the desire of a consumer society to have no learning curves. This tends to result in very dumbed-down products that are easy to get started on, but are generally worthless and/or debilitating. We can contrast this with technologies that do have learning curves, but pay off well and allow users to become experts (for example, musical instruments, writing, bicycles, etc. and to a lesser extent automobiles).

This is nowhere more evident than in the world of the mobile app. Any one app comprises a very small number of very focussed, very easy to use features. This has a couple of different effects. One is that my phone as a whole is an *incredibly* broad, *incredibly* shallow experience. For example, one goal I want help with from technology is:

> As an obese programmer, I want to understand how I can improve my lifestyle in order to live longer and be healthier.

Is there an app for that? No; I have *six* apps that kindof together provide an OK, but pretty disjointed experience that gets me some

[9] http://sicpers.info/2014/04/how-much-programming-language-is-enough/
[10] http://techland.time.com/2013/04/02/an-interview-with-computing-pioneer-alan-kay/

dissatisfying way toward my goal. I can tell three of these apps how much I run, but I have to remember that some subset can feed information to the others but the remainder cannot. I can tell a couple of them how much I ate, but if I do it in one of them then another won't count it correctly. Putting *enough software to fulfil my goal* into one app presumably breaks the cardinal rule of making every feature available within two gestures of the app's launch screen. Therefore every feature is instead hidden behind the externalised myriad gestures required to navigate my home screens and their folders to get to the disparate subsets of utility.

The second observable effect is that there is a lot of wasted potential in both the device, and the person operating that device. You have never met an expert iPhone user, for the simple reason that someone who's been using an iPhone for six years is no more capable than someone who has spent a week with their new device diligently investigating. There is no continued novelty, there are no undiscovered experiences. There is no expertise. Welcome to the land of the perpetual beginner.

Thankfully, marketing provided us with a thought-terminating clichÃ©, to help us in our discomfort with this situation. They gave us the cars and trucks analogy[11]. Don't worry that you can't do everything you'd expect with this device. You shouldn't expect to do absolutely everything with this device. Notice the sleight of brain?

Let us pause for a paragraph to notice that even if making the most simple, dumbed-down (wait, sorry, intuitive) experience were our goal, we use techniques that keep that from within our grasp. An A/B test will tell you whether this version is incrementally "better" than that version, but will not tell you whether the peak you are approaching is the tallest mountain in the range. Just as with evolution, valley crossing is hard[12] without a monumental shake-up or an interminable period of neutral drift.

[11]http://allthingsd.com/20130109/steve-jobs-was-right-tablets-are-cars-pcs-are-trucks/
[12]http://pleiotropy.fieldofscience.com/2012/07/crossing-valleys-in-fitness-landscapes.html

Desktop environments didn't usually get this any better. The learning path for most WIMP interfaces can be listed thus:

1. cannot use mouse.
2. can use mouse, cannot remember command locations.
3. can remember command locations.
4. can remember keyboard shortcuts.
5. ???
6. programming.

A near-perfect example of this would be emacs. You start off with a straightforward modeless editor window, but you don't know how to save, quit, load a file, or anything. So you find yourself some cheat-sheet, and pretty soon you know those things, and start to find other things like swapping buffers, opening multiple windows, and navigating around a buffer. Then you want to compose a couple of commands, and suddenly you need to learn LISP. Many people will cap out at level 4, or even somewhere between 3 and 4 (which is where I am with most IDEs unless I use them day-in, day-out for months).

The lost magic is in level 5. Tools that do a good job of enabling improvement without requiring that you adopt a skill you don't identify with (i.e. programming, learning the innards of a computer) invite greater investment over time, rewarding you with greater results. Photoshop gets this right. Automator gets it right. AppleScript gets it wrong; that's just programming (in fact it's all the hard bits from Smalltalk with none of the easy or welcoming bits). Yahoo! Pipes gets it right but markets it wrong. Quartz Composer nearly gets it right. Excel is, well, a bit of a boundary case.

The *really sneaky* bit is that level 5 *is* programming, just with none of the trappings associated with the legacy way of programming that professionals do it. No code (usually), no expressing your complex graphical problem as text, no expectation that you understand git, no philosophical wrangling over whether squares are rectangles or not. It’s programming, but with a closer affinity with the problem domain than bashing out semicolons and braces. Level 5 is where we can enable people to get the most out of their computers, without making them think that they’re computering.

Standing at the Crossroads

A while back I wrote Conflicts in my Mental Model of Objective-C[13], in which I listed a few small scale dichotomies or cognitive dissonances that plagued my notion of my work. I just worked out what the overall picture is, the jigsaw into which all of these pieces can be assembled.

And I do mean *just*. It’s about 1AM on Christmas Eve, but this picture hit me so hard I couldn’t stop thinking about it without writing it down and getting it out of my head. If it doesn’t explain *everything*, it shows me the shape of the solution at least.

A tale of two Apples

I believe that everything I wrote in the Conflicts post can be understood in terms of two different and (of course) opposed models of Apple. I also believe that the two models are irreconcilable, but that the opposition is also accidental, not essential. That by removing the supposed conflict between them, everything I thought was a problem can be resolved.

It was the best of iPads

The iPad is, I would argue (and accept that this is a subjective argument) the most tasteful application of computing technology to

[13]http://sicpers.info/2013/10/conflicts-in-my-mental-model-of-objective-c/

the world of many computer users. I would further argue, and this is perhaps on firmer ground, that the reason the iPad is so tasteful is because Apple spend a lot more time and resources on worrying about questions of taste than many of their competitors and others in the industry.

There are many visions that have combined to produce the iPad, but interestingly the one that I think is clearest is John Scully's Knowledge Navigator[14]. In the concept videos for Knowledge Navigator, a tablet computer with natural language speech comprehension and a multi-touch screen is able to use the many hyperlinked documents available on the Web to answer a wide range of questions, make information available to its users and even help them to plan their schedules.

This is what we have now. This is the iPad, with Siri and a host of third-party applications. Apple even used to use the slogan "there's an app for that". Do you have a problem? You can probably solve it with iOS and a trip to the app store.

It was the worst of iPads

OK, what do you do if your problem *isn't* solved on the app store, or the available solutions aren't satisfactory?

Well first, you'd better get yourself another computer because while the iPad is generally designed for solving problems it isn't designed for solving general problems. You might be able to find some code editors on the iPad, but you sure aren't going to use them to write an iPad app without external assistance.

OK, so you've got your computer, and you've learned how to do the stuff that makes iPad apps. Now you just pay a recurring fee to be allowed to put that stuff onto *your* iPad. And what if you want to share that with your friends? Only if it meets Apple's approval.

[14]http://en.wikipedia.org/wiki/Knowledge_Navigator

If the iPad *is* the Knowledge Navigator, it is *not* the Dynabook[15]. A Dynabook is a computer that you can use to solve your problems on, but it's also one on which you can create solutions to your own problems.

The promise of the Dynabook is that if you understand what your problem is, you can model that problem on the Dynabook. You model it with objects-either your own or ones supplied for you. You can change and create these objects until they model the problem you have, at which point you can use them to compute a solution.

The irony is that we have all the parts needed in a Dynabook, all in the iPad. Computer so simple even children can use it? Check. Objects? Check. Repositories of objects created by other people so we don't have to rewrite our own basic objects all the time? We call that CocoaPods (or RubyGems, or whatever the poison in your area of the world). But we just can't put all of these things together *on that computer itself.*

That would be distasteful. That might let people do things that make the iPad look bad. That might mean iPads providing experiences that haven't been vetted by the mothership.

Does using an iPad ever make you wonder how iPads work? What they can do? What you can *make* them do? You can answer these questions, but not using an iPad. Your Knowledge Navigator does not know the route to that particular destination.

This is what is truly meant when it is said that the iPad is not upgradeable. Forget swapping out memory chips or radio transmitters. Those are just lumps of sand inside a box made of melted sand and refined rock. The iPad is not upgradeable because you are stuck with the default *experience*: the out-of-the-box facilities plus those that have been approved from on high. It might be *good*, but it might not be *good enough.*

Notice that this is not an "everyone must program" position. That

[15]http://en.wikipedia.org/wiki/Dynabook

would be a very bad experience. The position is rather "everyone must have the facility, should they be so inclined, to make their computer better for them than the manufacturers did".

Conclusions?

I think that the Apple described above is not at the *intersection* of technology and the liberal arts. It is at the *border*, a self-appointed barrier of things that might flow between the two.

I believe that the two visions can be reconciled, and that a thing can be both the Knowledge Navigator and the Dynabook. I don't believe you have to disable some experiences to provide others. I believe that Apple the champions of tasteful computing can be applauded at every turn while Apple the high priests of the church of computing can be fought tooth and nail.

Enablers? Yes, please. Arbiters? No, thanks.

Culture, Heritage, and Apps

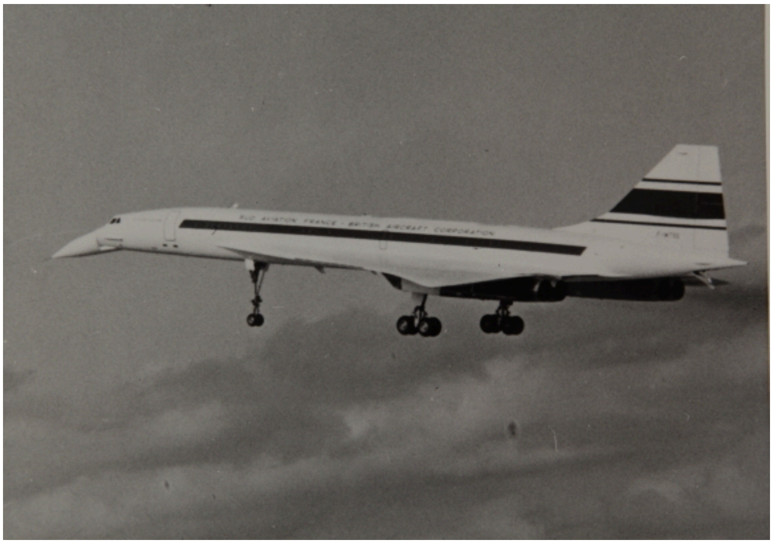
I said earlier on Twitter that I'm disappointed with the state of apps produced for museums and libraries. I'd better explain what I mean. Here's what I said:

> Disappointed to find that many museum apps (British Library, Bodleian, Concorde etc) are just the same app with different content. :-(

In each case (particularly Concorde) there's some magic specific to the subject. Rubber-stamping the app doesn't capture that magic. They're all made with a tool called Toura that sits atop PhoneGap. Makes it easy to get the content into the apps at the cost of expression. So to be clear: my problem is not with the museums and other heritage sites. I'm familiar with the financial and political problems associated with running a museum. My social circle includes curators, librarians and medieval manuscript experts and I know that money is tight, that oversight is close and that any form of expenditure in promotion must result in a demonstrable increase in feet through the door and donations to be considered a success.

The Magna Carta

My problem is also not with the Toura product and the team behind it. They're to be commended for identifying that while the culture and heritage community doesn't have much money, they still deserve to be represented in our tablets and phones. The apps I listed above all feature astounding objects: examples include the Concorde aircraft, the Lindisfarne gospels and the Magna Carta.



Concorde

So why are these apps disappointing? It’s basically because they’re all the same. Each of these objects has its own magic: the unique shape of Concorde, the vivid colours of the gospels and the constitution-defining text of the Carta. So why present them in the same way? Why not make a Concorde app that evokes aerodynamic speed, an illuminated Lindisfarne gospels app, and a revolutionary Magna Carta app? As many people have explained, it’s because standard content-viewing apps like Toura are all these institutions can afford.

Folio 27r from the Lindisfarne Gospels: wikimedia commons

There's something seriously wrong in our app industry. We've created a world in which apps are too cheap, and developers struggle to make a living from 70Â¢ per sale. Simultaneously, we've created a world in which apps are too expensive: the keepers of some of the world's most interesting objects can't afford to showcase these with more than a generic master/detail view. What's that about? What mindset leads us to demand high worth, when we're making products that we can't convince customers contain any value? If *they* don't see the value, are we deluding ourselves? People talk about events like the Instagram acquisition[16] as being evidence of a bubble subset of the app economy. But isn't that oxymoron - a free app that's worth a billion dollars - just an exaggerated version of what the rest of the industry is up to?

Let me present this problem in a different way, by moving from apps about museums to museums about apps.

The Rijksmuseum, Amsterdam

Last month I spent an enjoyable time with some very good friends

[16] http://dealbook.nytimes.com/2012/04/09/facebook-buys-instagram-for-1-billion/

in Amsterdam, the city of both apps[17] and museums[18]. While walking around the Rijksmuseum, I reminded myself that we're in the middle of the possibly the largest and definitely the fastest technology-mediated social revolution humanity has ever experienced. Those of us enabling the application of this technology are literally providing the fulcrum around which our species is revolving. During this year, museums will look at apps as cultural, historical and ethnological artefacts in their own right. Over the coming century, heritage institutions will shape the way that our brief period in time is presented for posterity. Be sure that all of the million apps currently available across all app stores will not be on display. As with any presentation, perfection is achieved not when there is nothing left to add but when there is nothing left to take away. Just as you can't (and wouldn't want to) walk into a museum and see every manuscript from the 10th century on display, you won't be able to walk into the Nieuwerijksmuseum in 3112 and see every app from the 21st century. Only those that are considered cultural treasures will be given pride of place in the galleries. Can something that apparently has no value be considered a cultural treasure? Will any of your apps be part of the presentation? I don't think I've yet produced anything that will be, and that needs to change.

[17]http://appsterdam.rs/
[18]http://www.amsterdam.info/museums/