School of Computer Science University of Birmingham UK

The iSoft affair:
Open Letter to my MP about government IT procurements (Originally sent August 2006)

Aaron Sloman
School of Computer Science, University of Birmingham
http://www.cs.bham.ac.uk/~axs

NOTE ON FORMATTING:
Adjust length of lines by adjusting the width of your browser window.

A PDF version of this document (generated by html2ps and ps2pdf) is available here, though the html version is the 'master' version.

CONTENTS

SEPARATE DOCUMENTS REFERENCED (including comments on this letter)

Anyone wishing his/her message or name associated with a message to be deleted should let me know immediately.
My original message soliciting comments stated that comments would be posted here.

- Pointers to the NHS project and news about it,
  including a disturbing report from Computer Weekly suggesting that the National Audit Office has been involved in a 'whitewash', and a letter to the Guardian suggesting that alongside the NHS disaster is another disaster in government schemes for IT in schools.
- Comments from members of South Birmingham Linux Users Group
  (several in local companies)
- Comments from academics on a list for professors and heads of CS departments
  including some signatories to the open letter criticising management of the NHS IT project in April 2006, and suggesting

  > that the Health Select Committee help resolve uncertainty about NPfIT by asking the government to commission an independent technical assessment with all possible speed.

- Comments from others.

ABSTRACT

The nature of large, multi-million pound long term (multi-year) IT contracts (and many non-IT contracts) with fixed requirements and budgets, including for example the UK Government project to provide a single IT system for the National Health Service in England, is that they are almost all **doomed** to fail, no matter how they are managed, monitored, audited, etc. I explain why, in terms of

a) the intrinsic complexity of the tasks, which makes accurate advance budgeting impossible, and specification of satisfactory design solutions in the original contracts impossible,
b) the complexity and unpredictability of the environment with which such systems must interact, which makes it impossible to find out all the important requirements in advance, especially in long term projects where external developments -- including changes in technology and culture -- will change the constraints and the opportunities and where the end users include many different kinds of individuals in many different organisations.

On the basis of this analysis, an incremental, experimental strategy is recommended for large scale IT development projects, in which the nation can learn what is worth doing and what is possible, by building, using and comparing partial systems developed in parallel. A collection of contractual requirements can then ensure that partial results achieved through public funding are available for future publicly-funded developments whether or not the original contractors remain involved.

A crucial condition for obtaining requirement specifications and design recommendations over such a long time time scale is involvement of end-users of all subsystems, preferably including some with a deep understanding of programming and other relevant technology. Asking senior managers in large end-user organisations (e.g. senior NHS managers) to specify or approve requirements at the beginning of a five or ten year project is a recipe for disaster, since they are in no position to understand or predict detailed requirements of a large and complex organisation with many sub-organisatins over a long period of time, and for various reasons often fail to understand all the detailed requirements already known to staff in their organisations. (Comparisons can be made with management decisions in the Challenger Shuttle disaster, where middle management staff told senior staff what they wanted to hear rather than the truth).

The only long term (multi-year) IT projects that can meet their specifications are those whose tasks are essentially routine because they involve little innovation, and even those will fail because at the end the requirements specified will be out of date.

There may also be issues of bad management, dishonesty, ignorance or wishful thinking that contribute to failures of large projects. But removing all those sources of poor performance will not overcome the intrinsic problems: non-routine, innovative, long term monolithic projects are doomed by their very nature to fail and need to be replaced by a totally different development process which is far more experimental, incremental, and flexible, allowing for large and unknown gaps in our knowledge at any time, which can only be identified and filled through further experiment, allowing for new relevant information and design ideas to come from totally unexpected sources.

An open society needs to engage in open problem-solving.
(Responding to an earlier draft Colin Tully commented that this proposal could be summarised in words echoing the writings of Karl Popper.)

Updates

Updated 14 Mar 2007
Added table of contents. Minor additional reformatting.
Updated 10 Oct 2006
A significant new development. The signatories to open letters calling for an independent inquiry into the NPfIT Programme have set up a web site here.
Updated 24 Sep 2006
Added a note on different uses of the word 'requirements'.
Updated 23 Sep 2006
In response to comments from Dennis de Champeaux, attempted to clarify the scope of the claims being made, e.g. more clearly excluding certain sorts of commercial IT projects, where existing systems and tools are applicable to a well-understood problem. He bears no responsibility for any remaining errors or omissions and has not checked the revisions.
Also added new comments from John Knapman here.
Updated 19 Sep 2006:
Expanded the References document to refer to the 'Down at the EPR (Electronic Patient Records) Arms' web site, which includes pointers to the 2003 invitations to tender, and a useful presentation by Sean Brennan in December 2005. Sean Brennan commented in this web site here.
Updated: 5 Sep 2006:
Added a further comment on differences between software and physical designs.
Updated: 31 Aug 2006: added abstract above.
Update: 29 Aug 2006 (Fixed some typos and minor errors.)

Open Letter to my MP Lynne Jones
(Liable to be updated. Comments welcome: they will be added to this web site.)

Dear Lynne

This letter is prompted by yet another botched high cost government procurement exercise: the iSoft affair, though the points made here are more general.

According to the BBC:

> The firm has been blamed for delays to the 10 year multibillion upgrade of the NHS computer system.
>
> It is supplying software to health trusts in the Midlands and north of England which will operate a new national electronic patients register.
>
> The National Audit Office warned in June that the huge project faced "significant challenges" if it was to be delivered on time and budget.

Additional press comment can be found here.

In view of your apparent interest in IT issues (I've read your sensible comments on the identity card proposals and seen this approving press comment on your criticisms), I thought I'd pass on some reflections on government IT procurement procedures.

Why large IT development projects are problematic

Although triggered by the iSoft affair, this document is actually a comment on all large IT development projects that involve designing large and complex novel systems over many years with contracts fixed in advance -- especially government-funded contracts like the UK NPfIT project.

This is unlike many commercial projects where the customer is a single organisation with strong management, making it possible for requirements and constraints for a solution to be developed using existing technology, in a fairly short time. My comments do not apply to such projects, which tailor and deploy existing systems in a new environment, using tools designed for that task. Multi-million pound long term government projects involving multiple types of users in multiple organisations are different in ways I shall try to explain, using the development of the internet as an example of how such projects might work.

Despite being a philosopher in a computer science department I am an experienced programmer, and I have experience of managing a project development team, including a team at Sussex university which produced Poplog, a system that won a 'Smart' award for exceeding $5M sales around 1991. So I have some experience that is relevant. In addition I offer an analysis of the unavoidable causes of the problems of predicting delivery times and budgets, and make some recommendations. The analysis is also relevant to some high technology physical construction projects, though similar solutions may not always be possible.

The root of the problem is whether software development can follow a predictable path. While managing software developments, I noticed that there was an interesting relationship between the size of a task and the reliability of the best estimates programmers could make as to how long the task would take. They nearly always underestimated, but if it was a task that they estimated would take up to a few days they were usually fairly accurate.

If they estimated that it would take longer, their estimates turned out to be incorrect by a factor that grew rapidly with the estimated time. So if they thought a task would take several weeks, I learnt to expect results in a few months, and if they thought it could take several months I learnt not to expect it to be finished in under a year, and sometimes it could take much longer. Why was this?

The mathematics of searching for a design

There is a fairly simple, partly mathematical, explanation for what seems to be something like exponential growth of the *prediction error* as time estimates increased.

The explanation has three main components.

1. The more complex the task, the less likely it is to resemble in detail previous tasks. So there will be many new design decisions to be taken, as opposed to simply applying previous designs with minor modifications (like writing control software for a slightly modified version of an old hardware device, or specifying parameters to tailor a large existing tool for a class of applications to a new application in that class).

2. The more complex the design task, the harder it is to tell at the time when early design decisions are taken whether they will prove successful, or whether they will lead to dead ends (e.g. because of unforseen interactions between different decisions) requiring back-tracking and new options to be explored.

3. Finally a point that probably requires study by social psychology. Programmers who are asked to estimate how long some novel development task will take are under pressure from sales, marketing or management staff to produce an answer when in fact they ought to say that they cannot answer. So they try to guess, or worse, give way to pressure from their superiors (like some of the people involved in the Challenger shuttle disaster, as reported by Feynman[*]). If they are forced to guess, they inevitably use relatively superficial features of the problem specification to guess on the basis of other tasks they have performed. The more novelty there is in the solution to the new problem, the larger the number of new design decisions that will have to be taken and that, as we'll see, can dramatically increase the error in the time estimate. Moreover, the more novelty is required in the solution to the problem, the less likely it is that programmers can detect that novelty in advance.

---

[*] Richard Feynman wrote:
Finally, if we are to replace standard numerical probability usage with engineering judgement, why do we find such an enormous disparity between the management estimate and the judgement of the engineers? It would appear that, for whatever purpose, be it for internal or external consumption, the management of NASA exaggerates the reliability of its product, to the point of fantasy.
http://www.fotuva.org/feynman/challenger-appendix.html

[I suspect that this is an example of a general phenomenon. A.S.]

How search spaces grow

The first point, about the number of design decisions, has mathematical implications. The more complex and novel the design task, the more it will require *searching* in a complex space of possibilities for a solution, instead of directly applying known techniques and solution patterns. It is a simple mathematical fact that if you have to take N decisions and at each stage you have at least D options, then you have a space of size at least D^N (i.e. D raised to the power N) options. So if you have only 2 options at each point and there are 3 decisions to be taken then there are $2^3 = 8$ options. If there are 20 decisions to be taken then there are $2^{20} = 1,048,576$, i.e. over a million options. But the numbers grow ever faster as N grows.

If there are 50 binary decisions to be taken the space grows to $2^{50} = 1125899906842624$ options.

If there are 10 alternatives at each point and 50 decisions to be taken it explodes to

```
100000000000000000000000000000000000000000000000000
```

options. If each decision takes only a second and everyone works 40 hours a week, 52 weeks a year, the number of person years required for exhaustive search is $(10^{50})/(52x48x60x60)$ i.e. approximately (after rounding):

```
13354700854700854700854700854700854700854701
```

So a team of 10,000 people working on the problem and doing an exhaustive search could take $(10^{50})/(52x48x60x60x10000)$ i.e. about

```
1112891737891737891737891737891737891738 years
```

Now a large and complex IT project lasting several years can require hundreds, or even thousands of decisions to be taken, adding hundreds or thousands more digits to the number of person years for an exhaustive search for an optimal solution. In fact if there are several hundred independent design decisions, the number of possible combinations can exceed the estimated number of electrons in the universe.

Of course, I am exaggerating the point for effect, since exhaustive searches are not required: there are many ways of reducing search, some based on the structure of the problem (e.g. symmetries), some based on decomposing the problem into many independently solvable sub-problems, some based on previous experience (or luck) that can lead engineers quickly to good solutions to some of the sub-problems, and some based on the fact that dead ends are often detectable at an early stage. Moreover abandoning the requirement for *optimal* solutions and accepting *workable* solutions ('satisficing' instead of 'optimising') can enormously reduce the task, by removing the need for exhaustive search.

But the remaining search space may still be very large and very deceptive -- a subject of much theoretical research in Artificial Intelligence in the last few decades. The biggest difference I found between really experienced programmers and less experienced programmers who were equally intelligent, was that the experienced ones were able to produce good solutions with much less searching, though even they produced imperfect first drafts, because of the enormous difficulty of anticipating all the complexities of a non-trivial (or more precisely, non-routine) novel design. This suggests that even automated design mechanisms using advanced AI techniques will not entirely remove the problems, except in cases where the machine is solving problems of a type with which it (or its design team) is already familiar from past experience. (As seems to have happened with great success in the design of computer components containing very large numbers of elements.)

Consequences for tendering

I have used 50 decisions as an example. However, a programming task requiring only 50 decisions is very small compared with the multi-million pound projects governments (and some big companies) like to commit our money to, which I expect (though I am not in a position to check this) include thousands of design decisions at various levels of abstraction and various degrees of novelty. So governments unwittingly give companies a completely impossible tendering task. Rather than lose business, companies do their best, and then governments enter into horrendously wasteful contracts. Is this avoidable?

Partly, but there's another fact about the problem that needs to be noted.

Getting requirements right from the start is impossible

The situation is made worse by the fact that often it is impossible even to get the *requirements* for new complex system right, let alone the design, because people cannot tell what they want until they have had time to try out various alternatives, in order to educate themselves about what the possibilities are, and what the tradeoffs are in choosing some possibilities rather than others. Also, if the project is going to take a long time, changes in other things may lead to changes in the requirements, including for example new technologies that make unforeseen kinds of interaction possible, or changes in security threats that make unforeseen kinds of protection desirable.

No amount of prior consultation among experts, potential users, focus groups etc. can overcome the problem of specifying requirements, if nobody has had prior experience of working with a system of the sort being planned, which by definition is *always* true of very large software projects of the sort discussed here.

After writing the above I found the following news report on the iSoft affair which seems to illustrate the problem of identifying requirements in advance:

> The company agreed to satisfy early stages of the NHS contracts with existing software. It would simultaneously develop its Lorenzo software to satisfy their later stages.
>
> However, it found that the needs of GPs, hospitals and health professionals differed enormously. It was unable to agree with Accenture and CSC, the two IT groups managing the deployment, on whether the software it was providing was meeting the needs of those working in the NHS.
> Timesonline, 26 Aug 2006

Certainly the computer experts cannot predict in complete detail what their systems will do, because usually the systems interact with humans, with the physical environment, with complex machinery and with social and economic processes in the environment (e.g. changes in power or salary costs, or in safety or privacy legislation), and computer experts are not usually experts on all of those parts of the total system. Nor is anyone else.

My own experience suggests that the best systems depend on a small subset of very bright *users* who also learn to develop software. They are then in a much better position to explore solutions that are both implementable and relevant to users, though of course there may never be unique solutions that work for all the users of a certain category if there are many of them. An illustration of what can be achieved by the combination of software engineering expertise and first hand experience of being a user is the outstanding OCAD map-making system, used world-wide, originally designed by a software engineer, Hans Steinegger, who was also a map-maker and an orienteer who used the maps. Of course this is not a networked, multi-site application.

> Others working on the project have also been users. This product has been developed since about 1992. But it was not a product for which anyone tried to provide requirements specifications for the long term in 1992. It evolved in response to the developing needs of users and a changing environment, including changing rules for orienteering maps produced by the International Orienteering Federation. I expect there are many other examples of successful products designed and developed incrementally over a long period of time in close interaction with users, but *without* a fixed budget, fixed time scale, or fixed requirements specification. Even some operating systems are like that. An outstanding, very much larger, example is described below.

A biologist will appreciate that the chances of coming up with a good solution in a reasonable time to a very large new software problem are analogous to the chances of a sequence of genetic mutations, even deliberately chosen mutations, leading rapidly from a microbe to a monkey, or even from a cat to a dog.

Are problems unique to IT projects?

Software designers construct new *virtual* machines that run on computers. This is in many ways different from building *physical* machines, because the space of virtual machines is not subject to as many constraints as physical machines that cannot escape the laws of geometry, physics, chemistry, etc. Adding a new connection between two arbitrary components in a virtual machine is normally always possible, whereas in a physical machine there may be no possible route available for such a connection because of the layout of other physical components.

So there is not so much scope for novelty in physical design. So building a large new bridge, or even a large new aeroplane, does not normally involve such a large search space because the fact that there are necessarily so many physical similarities between different bridges with the same sort of function and between different aeroplanes with the same sort of function will normally strongly constrain the search space for a new design.

Physical constraints

Moreover, in many cases, as a design for a physical system grows, it will, because of the geometry and physics, and the total space limits for the entity being designed, constrain further additions more and more strongly, whereas in general as a software design grows, the possibility of making further additions increases rapidly, although it may be temporarily limited by amounts of physical memory in available machines, limits of cpu speed, etc. However those computational constraints tend to recede rapidly over time, unlike physical limits constraining new bridges, buildings, vehicles, machines, etc.

Even so, jumping to an entirely new physical design, or a physical system requiring a very different kind of software control system (as happened years ago with with the Comet airliner, then with Concorde, and later with Airbus) can lead to huge search spaces, and very little chance of finding optimal solutions without a lot of search (and usually accidents, possibly including deaths). The failure to take account of effects of human bodies and brains in specifying requirements for the Millenium bridge is another example. Probably many delays in development of new physical weapon systems are another example (e.g. give "laser weapon" to google). I don't know whether the Wembley stadium delay is due to simple management incompetence or is another case of not understanding the implications of venturing into unfamiliar design territory.

It might be thought that the fact that features of physical systems can vary continuously will always make the search space of possible designs infinitely large. However it is often useful to chunk a continuous range of values into a relatively small discrete set of (possibly overlapping) intervals, and searching only in the set of intervals. Moreover, continuous variation sometimes allows 'hill-climbing' mechanisms to speed up search (repeatedly perturb everything a little and follow the direction of maximum improvement). However this can lead to a low grade local optimum requiring further search.

Anyhow it should not be thought that any software design task will require a larger search space than every physical design task, as there are special cases of both that are exceptions.

Implications for Government policy

There is a simple conclusion that follows from all this. Governments should *never* invest in huge long-term IT projects by entering into monolithic long term contracts with suppliers to provide the required systems, (a) because it is *impossible* for any software company to make reasonable estimates of costs or delivery times, and (b) because nobody can work out in advance what the detailed requirements are -- even if many people deceive themselves into thinking they can, e.g. because politicians who do not understand the intricacies of software engineering think that design requirements follow from high level political goals. (How many members of the UK government cabinet, or members of the civil service involved in detailed contractual negotiations, have had careers in engineering design, especially software design, especially design of powerful user interfaces required to meet multiple, evolving sets of requirements? I call that 'very soft ware'.)

> A corollary is that any political party that puts into its manifesto promises to deliver highly desirable results by introducing complex new mechanisms, policies, procedures, laws, etc. shows itself to be incompetent at understanding complex systems and how they change.

What can be done?

If there is a national problem that seems to require the development of a large and complex system, then governments must find a way to *grow our understanding of the problem* as we grow the solution, and thereby grow our understanding of all the detailed sub-problems, in small steps with a lot of parallel exploration of options.

There should never be a presumption that the companies that start on the projects will be the ones who finish them: contracts with such presumptions make it easy for contractors to go on milking tax-payers for a long time.

This means that all large IT contracts must allow for partially completed tasks to be taken over by new contractors with new ideas, possibly in parallel with and in competition with others. It should also be possible for adventurous companies to learn about the partial solutions and remaining problems and develop new solutions independently of any government funding, though they may then bid for funding for further development work.

All contracts must allow for regular 'check-point' (e.g. at least annual) reviews on the basis of which the project may be aborted or revised in various ways, including the possibility of assigning further contracts to new competing developers. Moreover, in order to make this feasible, it should be a contractual requirement that all the partial results must be made fully public wherever possible, so as to allow transfer of partially completed projects, and so as to allow new companies to gain expertise and produce potential extensions initially at their own expense, with the possibility of bidding for contracts for future extensions on the basis of demonstrable achievements. Legal and financial expertise may come to be far less important in contract negotiations and progress monitoring than technical and scientific expertise.

Some suggested prerequisites: requirements for openness

To make all that possible, there probably has to be a requirement for all solutions developed by the review check-points to be fully open:

1. so that alternative solutions can be compared properly,

2. so that others can both interface to the solutions and also, if they wish to try modified versions using new ideas, can implement variations of current versions, instead of having to waste time reinventing from scratch details already produced using tax-payers' money.

I.e. the outcome of such public ventures must *never* simply be proprietary code. (Even though that means that other rival economies can learn from our mistakes and our solutions. If they do, they may produce better final solutions than we can, and we may benefit by buying their solutions. It's a two way process).

If some contractors insist that optimal solutions require proprietary code that they have *previously* developed at their own expense (so-called 'prior art') then there are at least the following options:

- They can offer their proprietary code for sale to the government, including full documentation of file formats, protocols and programming interfaces, so that it can thereafter be made open source and freely available for other developers working on the system, who then have the option of using it, or improving it, or studying it as a basis for producing better systems.

- They can undertake to make the prior proprietary code available in closed source form to all others working on the project at a fixed price per licence, agreed in advance.

There may be other solutions to the problem of allowing partial solutions to be further developed by new contractors. In all cases, any contractor paid from public funds to produce a working system that includes any proprietary code should, in the public interest, be required to make all the file and communication formats, used by their proprietary code, and all the programming interfaces, fully public, so that rival developers can attempt to produce alternative components with the same functionality as the proprietary ones, and so that if the original contractor goes out of business or withdraws its services the nation is not left with unmaintainable systems, and also so that a publicly funded project never permanently requires the use of a particular piece of proprietary software till the end of its life, which could stifle some developments, and waste huge amounts of money over the years.

Companies that refuse to agree to these conditions are too risky to do business with in very large and expensive projects.

Projects managed in this way may look messy and complex, and the companies involved may find that they have to get used to smaller, shorter, contracts than they have been accustomed to. However, the good ones will benefit from the openness, causing technology to advance faster because of the sharing of ideas and results, and newcomers with powerful ideas will have opportunities to show how they can do better than the established giants: something we have seen happen often in the world of hardware and software development (especially software). Such a project can grow organically over years, with requirements and designs being reshaped as more is learnt about what is useful, what works, and what the tradeoffs are and the knowledge gained at public expense is shared among many with the potential to make good use of it, instead of being hoarded for use by companies who may lack some of the people who are best able to follow up the developments.

Is this beginning to sound familiar? You already know of a spectacular example, whose development had many of these features.

A precedent for this proposal: The internet

Suppose some government or international authority had decided 35 years ago that we need an international communication system allowing anyone on earth to find available information about anything, and allowing anyone with a product or service to sell it to anyone willing to buy. And suppose they then invited tenders from large IT companies (which existed in 1971) to produce such a system. In that case we might still be waiting for it; or there might be a partial implementation with many flaws being tested somewhere after hugely overrunning its timetable and its budget. The project might still be in the grip of one of the companies that existed 35 years ago, but has since failed. Tax-payers' funding might have prolonged its life, but without much public benefit.

That's not how the internet actually started. There were some US government-funded contracts to produce experimental computer networks, with limited functionality, notably US funding for what was then called ARPANET, later supplemented by funding in other countries to do collaborative experiments with the US, but not funding to do the whole of what we now know as the Internet. Nobody at that time could have produced a requirements specification listing all the current uses of the internet. (See also the wikipedia summary.)

That's partly because I doubt that anyone could have anticipated all the technological changes that now support that functionality. I remember sitting at a teletype at Sussex university in 1973 or 1974, connected by a telephone link to a computer in London, and from there to a transmitter in Norway, signalling via satellite to a receiver in California, connected to Xerox research lab in Palo Alto, where they had a big experimental computer (MAXC) on which I was running test programs, and typing email messages to people at Xerox. However, in those days each time I pressed a key it could take a few seconds for the echo to come back. Transmitting images or sounds in real time was just out of the question.

In fact the main original objective was to allow expensive computing resources to be shared for running programs. So it was assumed that data would be transferred for a computing task, and the results would be transferred back, so that most of the computing power was to be used for executing programs on one machine. What was not predicted and was not built into the initial specification was the requirement that the network should meet another major need: human to human communication. That soon proved far more important than the original goal of distributing computing power, a requirement whose importance diminished as both the power and the power to cost ratio of computers grew rapidly. (However the original goal was recently revived as one part of the e-Science initiative.) The importance of networks for human interaction was not understood by many who would be using that functionality a few years later. I recall a letter from someone in industry to *Computer Weekly* arguing that 300 bits per second would suffice for terminal connections because nobody can read more than 10 characters per second.

How the internet grew

Instead of being a monolithic long-delayed project wastefully consuming many billions of tax dollars, the network grew through successive parallel and often unplanned developments of a collection of key ideas and many partial experiments using techniques that became publicly available for others to try out and extend. There were thousands of small, medium and large experiments of different sorts, often done in parallel and often done independently except for the requirement to conform to publicly available standards that had been agreed at the time (many of which later changed in the light of experience), where many of the experimental ideas worked for a while and then were replaced by better ones, a process that is still going on.

The internet did not all start with Berners-Lee: key steps were made much earlier, though his contribution was catalytic for many subsequent developments. Others had grasped the enormity of what was going on a decade earlier, as shown by the brilliant slogan of Sun Microsystems: 'The network is the computer' around 1984. (Some features of that vision are only now being realised as more and more of what individuals do happens on remote machines, not the machine they are currently using, e.g. reading and writing email messages stored on a remote host, filling in tax forms, submitting research council grant proposals, doing banking transactions, running educational software, running spreadsheets on google, exploring digitised maps, making travel reservations, etc.: I would be surprised if the government's IT plans are based on understanding of that trend, making PCs redundant for many users in the near future, because 'thin clients' will suffice.)

For about 20 years, there were many different experiments and a whole sequence of different resulting inventions that produced what was essentially the internet without the world wide web. Some of us were by then regularly using email, posting to electronic bulletin boards, transferring files using ftp, and using remote login to run programs, even interactive programs, on big remote computers. Some of those developments might have happened faster if the near monolithic power of IBM in the commercial world had not prevented deployment of much better alternatives to

IBM PCs (e.g. products from Sun, Apple, Apollo and other smaller companies using non-Intel based cpus, and non-Microsoft software). It's not easy to tell whether there would have been the same drop in costs because of growth of numbers and the existence of clone makers. (Sun Microsystems from the start allowed others to build hardware to run their operating systems, unlike Apple, for instance.)

Then Berners-Lee produced a collection of platform-independent ideas that made it much easier for most people to use those facilities to share information and that was a catalyst for a host of extremely rapid new developments that nobody could have planned and whose solutions no software company on earth could have delivered.

Some of the history of the ten years from 1990 can be found here. It mentions several standards that were developed in that period but there were lots more formal and informal standards connected with the growth of the internet. Of course, there are many more histories of computing and the internet available online, emphasising different things.

> I have given the development of the internet as an example because everyone reading this has already encountered it, and knows something about its profound importance. As indicated in one of the comments on this document there are other examples.

Implications for government policy (continued)

If all this is correct, Governments have to stop thinking they can make promises about what will happen or that by taking decisions they can make good things happen on a large scale. Instead they have to understand that at best they are like experimental breeders, trying many things out, learning, and using things that work well as a basis for further developments.

I think the arguments about designing complex software systems apply to designing almost anything complex at government level, e.g. a transport system, a fiscal system, a primary school educational policy, etc., for the same reasons: the future is always too complex and too unknown for good decisions to be taken except on a small scale and as a result of parallel experiments where outcomes of alternatives are compared over time. Wishful thinking about desirable objectives is no substitute for that cumulative growth of understanding of problems and opportunities.

Are some projects exceptions?

Some people may feel that military projects, a major source of income to high technology companies and their shareholders, must be an exception because the processes cannot be made public for fear of actual or potential enemies getting the benefit of the results.

A cynical response is if that happens then it may be faster to learn from what those enemies achieve than to continue as originally planned. An alternative answer is to adopt the above recommendations but instead of having complete openness as specified above, have openness within a collection of companies and research and development organisations that have passed some test for trustability. After all, if *any* commercial organisation is trusted with military technology secrets there have to be tests for their suitability, and in that case the same tests can be used for a pool of potential co-developers and competitors.

A snag is that having a closed collection of allowed developers increases the risk of collusion between the companies in order to cheat governments, but that risk has to be dealt with through monitoring processes. Also if the list is not closed permanently and other companies can be

added, the resulting competition could defeat the colluders.

A bigger risk is that the best solutions are sometimes available only in companies or organisations that for whatever reason are not in the trusted set (e.g. because the requirements for being trusted with military developments impose too many restrictions on activities that are crucial for other kinds of innovation and deployment -- e.g. university teaching and research).

Concluding Comment

I am not recommending a free market and non-interventionist government, as some may think.

On the contrary, all this is consistent with government-inspired and government-funded parallel experiments with a government-enforced policy of openness regarding outcomes produced using funding from tax-payers, so that learning is maximised and benefits are shared. This may produce experiments that will not happen through market forces in a capitalist system.

If all this is correct, then probably the huge monolithic project based on a single contract should never have happened, and the government should re-structure the project instead of wasting more money. I don't know enough about that project to know exactly what should happen, but I suspect the best way forward is to fund several small scale experiments in parallel, constantly comparing results and after each major review building on the best results so far, with all partial results being in the public domain, for the reasons given above.

It's a bit like managing a complex ecosystem with a provisional set of high level goals for change, where the goals themselves may have to change over time. Sometimes the ecosystem manages its managers.

From a report in the Guardian in March 2007, it appears that moves in this direction are now starting. Look at this

---

NOTE: Related comment

For a related document, see my response to a UK government consultation on personal internet security.

Maintained by Aaron Sloman
School of Computer Science
The University of Birmingham