Advances in Project Management Series[1]

The return of the hacker: Rethinking projects, progress, innovation and teams[2]

By Prof Darren Dalcher
School of Management, University of Lancaster
United Kingdom

Recent articles in the series focused on entrepreneurship (alongside innovation and creativity), and on the need to reconnect with people at the core of management work. This article links the two areas while revisiting established writing in the domain of managing software projects, and people, and highlighting some of the pioneering thinking that has emerged from that domain before turning to explore new possibilities that can refresh some of the insights and renew some of the conversations.

The mythical man-month: Mixing effort and progress

Re-reading old classics can prove to be both a source of immense pleasure and intense frustration. The pleasure comes from being re-acquainted with an old friend after an extended absence. It is often enriched by the ability to make sense of and see afresh through some of the ideas and writing that appeared in the original source. The passage of time often allows for progress to be seen through an informed lens. But therein also lies the source of deep frustration, when 55 years after the fact, many of the lessons and insights remain equally relevant, yet still appear not to have been embraced or understood.

The mythical man-month was written by Fred Brooks (1975) to recount his experience of managing a very large and rather complex project for IBM ten years earlier. It contains a series of essays that enable the readers to join Brooks in making sense of his management journey. The book is the undisputed best-seller in software engineering, selling over a million copies. Yet, its appeal extends well beyond the realms of the software engineering community, generating reviews, citations and correspondence from lawyers, doctors, psychologists and sociologists (Brooks, 1995; p. 254). If we look for the underlying reason for the enduring appeal of the book, it might well be that the focus on software engineering is simply the context utilised for a more intimate reflection on people, teams, interactions, communication and achievement in projects; areas that still merit attention and that may still defy full understanding.

---

[1] *The PMWJ Advances in Project Management series includes articles by authors of program and project management books published by Gower in the UK and by Routledge publishers worldwide. Each month an introduction to the current article is provided by series editor* ***Prof Darren Dalcher****, who is also the editor of the Gower/Routledge Advances in Project Management series of books on new and emerging concepts in PM. To see project management books published by Gower and other Routledge publishers, click here. Prof Dalcher's article is an introduction to the invited paper this month in the PMWJ.*

[2] How to cite this paper: Dalcher, D. (2019). The return of the hacker: Rethinking projects, progress, innovation and teams**,** *PM World Journal*, Volume VIII, Issue VI, July.

Indeed, the challenges, constraints and tribulations recounted by Brooks reflect the experiences still encountered on many large projects. Brooks metaphorically locates large-system projects in the pre-historic tar pits where '*many great and powerful beasts have thrashed violently… Most have met goals, schedules and budgets. Large and small, massive or wiry, (yet) team after team has become entangled in the tar'*. (Brooks, 1995; p. 4)

To this day, the annals of failure are still filled with many a great behemoth mired in the metaphorical tar pits. Brooks notes that everyone seems to be surprised by the stickiness of the problem, before asserting that more software projects have gone awry for lack of calendar time than for all other reasons combined (p. 14). He duly identifies a number of contributing causes (which are paraphrased below):

- **Estimation**: Techniques for estimating are poorly developed and reflect an unvoiced assumption that all will go well
- **Techniques**: Estimation approaches tend to confuse effort with progress, fallaciously assuming that people and time are interchangeable
- **Position**: The inherent uncertainty of the estimates allows managers to collapse schedules in order to respond to wishes and expectations
- **Monitoring**: Schedule progress is poorly monitored
- **Action**: The response to identified slippage is to add resource (manpower), which Brooks equates with dousing a fire with gasoline

However, there are two additional, albeit linked, monumental issues that Brooks labels as fallacious thought modes:

Optimism: Firstly, there is a false optimism that pervades thinking in computer systems and enables management decisions to anchor on simplistic and naïve assumptions about future progress (p. 14-15). Thinking around the excessive optimism described by Brooks chimes with contemporary global discussions around strategic misrepresentation, wishful thinking, conspiracy of optimism and optimism bias (Ascher, 1993; Sharot et al., 2007; Flyvbjerg, 2008; 2009; Love et al., 2011; Sharot et al., 2011; Sharot et al., 2012; Dalcher, 2013).

Mythical effort: Secondly, the basis of estimating and scheduling in software projects is erroneously derived. The unit of effort utilised in such calculations leads to significant miscalculations. Brooks takes issue with the mythical concept of *man-month* (a hypothetical representation of the work done by one person within one month) which underpins much of project thinking. When effort is represented utilising the currency of person-months, the cost inevitably depends on the product of the number of people employed and the number of months. Progress, on the other hand, does not!

Indeed, the relationship between effort and progress is far more difficult to determine and untangle.

'*Hence the man-month as a unit for measuring the size of a job is a dangerous and deceptive myth. It implies that men and months are interchangeable*.' (Brooks, 1995; p. 16)

In reality, people and months can only be regarded as interchangeable commodities when a task can be partitioned without the need for communication or coordination amongst the team

members (i.e. when no interaction overheads need be considered). This implies a labour-intensive task that can be carved into segments. When there is a need for dependence between individuals or a requirement for continuous communication, the simple measure no longer holds. More complex and more demanding interactions therefore defy the excessive simplicity implied by the 'mythical man-month' concept. Indeed, such confounding misunderstanding, and its significant impact on how projects are understood and managed, give the book its rather unique title.

Mind what you wish for…

So, what happens when a project is behind schedule?

Conventional logic applying the *mythical man-month* formula, would typically result in adding manpower to the project to recover the schedule.

Indeed, when asked to name their single magic solution by the mythical project genie, many project managers would request either more time, or more money. Money is often used as a proxy for additional resources or manpower. Where time cannot be made, managers may resort to trying to achieve more over available time by resorting to the mythical man-month.

The logic is rather compelling. To make up for lost time we add resource, thereby enabling faster progression and execution. Brooks refers to such naïve addition of manpower as *regenerative schedule disaster* (p. 21).

Brooks offers the example of a task estimated at 12 man-months, assigned to three people, for four months. If the work meant to be completed over the first month, is only finished after two months have elapsed, what options would be available to the project manager?

The options are summarised, rebadged and repositioned as follows:

1. Early obstacle: If the task has to be completed on time, and the delay (or mis-estimate) only relates to the initial month of the work, 9 person-months of effort still remain to be completed over two calendar months. The simple solution is to add 2 people to the 3 already assigned to the project.

2. Systemic error: If the task has to be completed on time, but the delay/mis-estimation applies to the rest of the work, then 18 months of effort remain to be completed over two calendar months, requiring nine team members. Hence, 6 people must be added to the 3 already assigned to the project.

3. Reschedule: Allow enough time to complete the work needed in the new schedule by taking account of the delay or mis-estimation.

4. Trim the task: Reduce some elements of functionality, or scope, or eliminate some activities in order to allow a minimal task to complete on time.

Brooks maintains that the first two alternatives are disastrous due to the regenerative effects of such action. In the first case, the two new people, assuming they can be recruited and assigned

immediately, without any further delay, would require training in the task by one of the experienced professionals. The training will have used up three-person months, for work not appearing in the original estimate. Moreover, the addition of personnel would necessitate repartitioning of the work, further integration and testing and other overheads. By the end of the third month, substantially more than 7 person-months remain, and only 5 trained people and one month are available. The remaining work thus becomes even more urgent…

'*The temptation is very strong to repeat the cycle, adding yet more manpower. Therein lies madness.*' (p. 25)

This is of course the simplest case. If scenario 2 is more pertinent and the entire task has been miscalculated, a far more extreme injection of resources may be expected, with wider ranging impacts and an even greater need for further resource consumption…

Whilst the numbers involved in the illustrative case are small, Brooks invites readers to imagine the regenerative impact of adding hundreds of people to a project running late, and doing so, in multiple iterations as progress is consistently being closely monitored… Drawing on his experience, Brooks concludes, with what has become known as Brooks' Law, that:

'*Adding manpower to a late software project, makes it later*' (p. 25).

Reflecting on the impact of adding manpower, Brooks notes that while the programming work performed increases with direct proportion to the number of programmers (N), the resulting complexity of a project increases by the square of the number of programmers (N^2). Therefore, he asserts that it should follow that thousands of programmers working on a single project should become mired in a complex nightmare of human communication and version control.

Reimagining the computer at IBM

Brooks' Law is derived from his experience of managing a significant project for IBM. Yet, while he explains the implication of the mythical man-month mindset, he does not frame it with the actual figures from his own project. It might therefore be useful to gain an appreciation of the source of his experience by looking at other material related to the project and its context.

Brooks was a manager concerned with developing the IBM System/360 computer family. People outside the industry may not immediately recognise the product, but System/360 introduced a fundamental innovative transformation in the computer industry, akin in significance to the launch of the Ford Model T, the release of Boeing's first jetliner, the 707, or the development of the incandescent light bulb.

In 1960s IBM were the dominant computer manufacturer in the US. By 1961, they controlled two-thirds of the American market, however they offered a fragmented range of incompatible machines from across six different product lines, which could not be upgraded, adapted or upscaled.

Computer hardware was application specific and typically divided into scientific (fixed-word length, binary arithmetic) and business (variable-word length, decimal arithmetic) ranges. The two types were developing and growing independently so companies like IBM, had many

different lines of computers, which were developed separately utilising different languages and different systems software. Developers had to have a total understanding of the hardware, the operational limitations and the operational requirements as development was intimately tied to the host machine.

New computers were being introduced every year or two [Glass, 1997]. The case for each new machine was compelling, yet each new computer was different from, faster and cheaper than its predecessor. Software people were therefore forced to re-write the programs on a new computer platform whenever a new one appeared. And yet, Software was free, as hardware vendors were only too keen to give away systems software (without which the hardware would not work).

To tackle the challenge IBM established the SPREAD (Systems Programming Review Engineering and Development) task force in 1961 and based it in a secret facility at a remote location. The bold recommendation of the task force was to replace all existing IBM machines with a new unified product line that would support scientific as well as business computing. Moreover, from the smallest model, the range could share the same accessories-tapes, disks and printers-and run the same programs.

Prior to the introduction of System/360, manufacturers built each new computer model from scratch, and so the operating system would also need to be built from scratch. The emergence of the general-purpose IBM 360 series signalled the beginning of a new era. Computers were becoming faster and cheaper. The 360 family offered a range of options and platforms to suit all needs whilst allowing for growth (i.e. modularity).

System 360 would deliver a family of compatible computers that could fill every data processing or computation need offering a range of cost and performance options. Customers could start with small computers and move up or just add components as their demands increased, taking their old software along with them. No longer was there a need to ditch the computer every other year and re-write all the existing code. The 360-range comprised of anything from a small business machine to the largest scientific computer.

The proposed new concept represented a major corporate gamble that would make all existing IBM machines obsolete. The name System/360 reflects the proposed flexibility of the range and its ability to deal with any permutation of commercial and scientific information processing requirements. System 360 played a pivotal role in revolutionising IBM and ultimately, transforming the entire industry. For the first time the company manufactured its own components. Engineering and manufacturing were standardised on a world-wide basis as overseas plants and labs concentrated on producing one product line for both the domestic and foreign markets.

The development of a common architecture, as opposed to a specific implementation enabled compatibility across the entire range. This gave customers the flexibility to change machines and still retain their code. In one stroke IBM had shifted the focus from a single, one-off machine (computer), to the notion of computer systems.

To announce the launch, on April 7th, 1964, IBM held a press conference in Poughkeepsie, New York, while simultaneous events were held in 165 other US cities and 14 other countries,

engaging a total audience of 100,000 customers. IBM had even chartered a full train to deliver journalists from New York City to the main press conference. Speaking at the main event, Thomas Watson Jr, President of IBM, announced that it was a sharp departure from previous computing concepts. '*The result will be more computer productivity at lower cost than ever before. This is the beginning of a new generation - not only of computers - but of their application in business, science and government*.'

Managing the massive project

Yet, the radical notion of compatible software for the entire product range was responsible for placing tremendous demands on the programmers. The concept of the 360 family nearly failed when software problems created significant delivery delays. Previously, IBM's annual spending on software had been around $10 million (Fishman, 1981; p. 97). The anticipated expenditure for the 360 family was a far more significant $125 million, representing by far the most advanced, most expensive, and the most challenging software project IBM had ever attempted (ibid.). Lack of knowledge about testing software by the product test engineers and the unavailability of the hardware hampered the development [ibid.]. More critical was the fact that software personnel were not involved in the early planning and subsequently managed the software effort separately [ibid.; p. 101].

While Brooks (1995; p. 31) talks about the original 200-person team that started the project, Fishman (1981; p. 97) reports that at its peak the project involved 2,000 programmers that were attempting to complete the undertaking. In March 1966, Tom Watson Jr. admitted to software problems and remarked that the programming costs would run nearly as high as the hardware development costs. The actual cost according to insiders was in the region of $500 million [p. 101] – four times the original estimate. Additional enhancements to the software cost even more [ibid.].

Cortada (2019) notes that the software development staff was described as being in "disarray" as early as 1963. In response, IBM added 1,000 people to the operating system project, costing the company more for software in one year than had been planned for the entire development of System/360.

Reflecting on the cycle of the project, Brooks (1995; p. 20) notes that many of the tasks were sequential in nature, so that even when coding was apportioned to more programmers at the behest of management, there would be an essential need to test and debug the combined efforts of all individuals. Moreover, the more people that became involved, the greater the need to co-ordinate their effort (p. 18). Programmers would thus waste as much time communicating as was saved by increasing the size of the team (p. 24).

'*The promises the company had made for the software were too ambitious, because it had not measured which features were essential to the customer and which he could do without. Thus programming costs were grossly underestimated*.' (Fishman, 1981; p. 101)

Engaging in such a transformational project had not been easy, as Thomas Watson Jr later explained: '*The expense of the project was indeed staggering. We spent three quarters of a billion dollars just on engineering. Then we invested another $4.5 billion on factories,*

equipment and the rental machines themselves. It was the biggest privately financed commercial project ever undertaken.' (Sparkes, 2014)

Indeed, the actual total cost was close to $6 billion; this equates to well over $40 billion in today's terms. In terms of total project costs in the 1960s, this was second only to the Apollo project. Such a high price tag, would also easily qualify the undertaking as a very significant megaproject in today's terms.

Fortune magazine branded IBM's System/360 as the $5 billion gamble. To put that number in context, IBM's total revenue in 1962 was $2.5 billion, so the investment of what turned out to be $6 billion in a new concept, at more than twice the annual revenue, and at about 24 times IBM's annual profit, would be an enormous make or break gamble.

The reaction from customers was overwhelming. Within the first four weeks, IBM had more than a thousand orders. Within three months IBM had received $1.2 billion in orders. By the end of 1966 the System/360 had generated a billion dollars in pre-tax profits. Within five years over 33,000 units would be sold (compared to a pre-launch total of 20,000 computers of all makes across the US, Japan and Western Europe). System/360 has also left an important legacy, as every subsequent IBM machine can be said to be a descendent of System/360.

Within a short period, System/360 machines were being used by most banks, universities, airlines and government offices. Moreover, five of the machines made NASA's Apollo 11 mission possible, crunching the data relayed by the spacecraft's small onboard computer and displaying it for the support crew in Houston (Cotada, 2019).

Through the 1970s, more than 70 percent of mainframes sold were IBM's. IBM's base of installed computers rose from 11,000 in early 1964, to 35,000 in 1970. IBM itself also grew, more than doubling its workforce from 127,000 employees worldwide in 1962 to 265,000 by the end of 1971. Meanwhile, revenue rose from $2.5 billion in 1962 to $8.3 billion in 1971.

With the benefit of hindsight, the most expensive CPU project in history, that came close to bankrupting the company, had proved to be an enormous business success, that would further ensure an enduring market share in the mainframe computer market. Notwithstanding a significant underestimation of the cost required to complete the project, and despite consistent efforts from senior management to get the project back on track by doubling the size of project team and virtually destroying the entire initiative, the final product had become a success with customers, transforming the entire computer industry and society in the process by enabling many organisations to own and operate their own computers.

Learning the lessons

Brooks (1995; p. 30) concludes that the sheer number of minds that need to be coordinated affects the cost of the effort, for a major part of the larger effort '*is in communicating and correcting the ill effects of miscommunication*.' The obvious alternative would therefore be to use as few minds as possible.

'*Indeed, most experience with large programming systems shows that the brute-force approach is slow, inefficient and produces systems that are not conceptually integrated. OS/360, Exec 8, Scope 6600, Multics, TSS, SAGE etc.— the list goes on and on*.' (ibid.)

The question is, can a small team still be successful in developing large and significant undertaking?

'*The dilemma is a cruel one. For efficiency and conceptual integrity, one prefers a few good minds doing design and construction. Yet for larger systems one wants to bring considerable manpower to bear, so that the product can make a timely appearance*.' (p. 31)

Placing conceptual integrity at the core of system design (p. 42), ultimately moves Brooks to declare that the design must proceed from one mind (p. 44) or a small group of agreeing minds (p. 233). Where schedule pressures necessitate involving many hands, he recommends either a division of labour between architecture and implementation or a new way of structuring teams around the key designer. However, in order to ensure conceptual integrity someone must control the concepts. According to Brooks, '*that is an aristocracy that needs no apology*' (p. 233).

Such a conceptually integrated system is faster to build and to test. Brooks recounts a crucial decision around the writing of external specification for the operating system, where he made the wrong decision about the system (p. 47): The architecture manager had ten good men and asserted that they could write the specification and do it right, which would take ten months, three more than the schedule allowed. The Control program managers had 150 men. He asserted that they could prepare the specifications, under the coordination of the architecture team. He contended that they would get it right, do it on schedule and it would keep his team busy rather than waiting for the architecture team to finish their turn. The architecture manager retorted that if the control team were given the responsibility, the product would not be on time, but three months late, and of poorer quality.

Faced with a difficult decision, Brooks decided not to listen to his architecture manager, giving the job to the control team. In the book he admits to being swayed by the allure of having a team of 150 implementers on the case (p. 48). The result… was delivered three months late and at a much lower quality. Moreover, he reflects that '*the lack of conceptual integrity made the system far more costly to build and change, and I would estimate that it added a year to debugging time*' (ibid.).

Targeted small teams can thus be more effective than large collections of individuals, which need to be coordinated and controlled. Brooks concludes that conceptual integrity does require that a system reflect a single philosophy and that the specification as seen by the user can only flow from one, or very few, minds (p. 49). Indeed, a smaller team results in simplified communication and improved conceptual integrity that can reduce the time needed for implementation (p. 50).

Upon revisiting the work twenty years later, Brooks still emphasises the critical role of the architect as '*the most important action is the commissioning of some one mind to be the product's architect, who is responsible for the conceptual integrity of all aspects of the product perceivable by the user. The architect forms and owns the public mental model of the product*

that will be used to explain its use to the user. … The architect is also the user's agent, knowledgably representing the user's interest in the inevitable tradeoffs among function, performance, size, cost and schedule. … The architect is like the director, and the manager like the producer of a motion picture. (p. 256)

The final main point that Brooks reflects on, after 20 years, is to do with creativity and its relationship with control and structure (p. 276). Given his belief that creativity comes from individuals and not from structures and processes, the key question is how to design structure and process so to enhance, rather than inhibit creativity and initiative.

Brooks draws on Schumacher (1973), recognising that freedom and enhanced responsibility in smaller units and individuals, can result in happier and more prosperous organisations. Schumacher makes a powerful case for building economies around the specific needs of communities rather than the wishes of large corporations and conglomerates. Brooks maintains that the secret to unlocking improved productivity, is through delegated power and freedom to operate within existing structures (Brooks, 1995; p. 279).

The key notions of conceptual integrity and creative productivity continue to challenge most constructive endeavours that strive to embrace and balance innovation and creativity, with control and structure. Such concerns come to the fore particularly when the contrast between the creativity of the individual, or the small team, contrasts with large or complex organisational undertakings that seem to impose a strong preference for efficiency, structure and control. Attempting to guide and control major undertakings typically dehumanises and deemphasises the role of individuals, resulting in a direct conflict between the overarching need for structure, and the human capability for creativity and innovation.

Attempts to balance creativity and control have featured through the history of systems development, project management and organisational behaviour, primarily through a continuing struggle to accommodate talented and creative individuals as part of global initiatives. A fascinating arena that reflects the very same pervading tension can be observed through the emergence, partial disappearance and re-emergence of hackers, the innovative technology-savvy problem solvers able to utilise new approaches and perspectives in delivering creative solutions within the burgeoning software systems domain.

The emergence of the hacker: Where do hackers come from?

Hackers are not a new phenomenon. Hackers appear to have always congregated around computer systems and new technologies. In the early days of computers (c. 1950) hackers predominated. Hackers had an in-depth knowledge of programming languages and the hardware on which systems were installed coupled with a delight in solving problems and overcoming limits. Ince (1988), refers to them as the original programming magicians.

Good hackers were able to use programming 'tricks' in order to reduce memory size, get programs to run faster, link various devices to computers, bypass faulty or slow functions, create alternative options and subroutines, and save precious hours of machine time. Highly skilled individuals could thus practice their craft by attempting to maximise the machine through creative manoeuvres (Dalcher, 1999).

The typical approach to programming, known as the 'code and fix' model, used in the early days of computer programming comprised two steps (depicted in Figure 1) reflecting the essential elements of the task of creating new software:

- Coding
- Fixing errors in the code

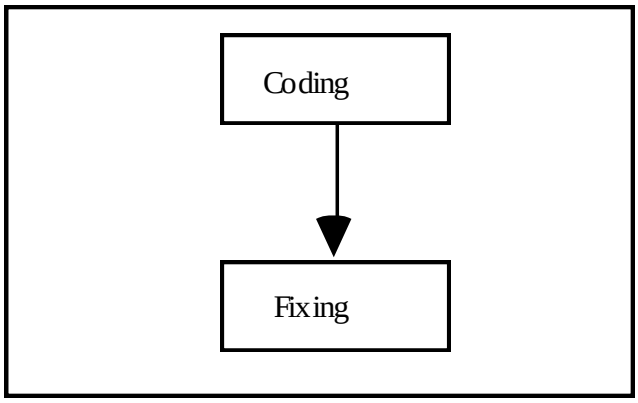

Figure 1. The Code and Fix Cycle

The cycle reflects the essence of developing new programs; software was conceptualised and developed in someone's head, coded and used. The raw code was not structured in any formal way. Hacking was thus a form of art or craft, rather than an engineering discipline. In common with Brooks notion of conceptual integrity, the initial mental concept for a new programme would be artfully 'sculpted' into a functional program, much like a sculptor transforming a raw block into a masterpiece.

Over time, frequent corrections would degrade the initial structure and the quality of the program and make subsequent fixes and corrections far more complex and labour-intensive. Without due care, the code and fix model could ultimately result in programs that grow unacceptably large and degrade to a stage where it becomes impossible to continue to use them.

Ultimately, the programs could become very convoluted. When a hacker left a company, they took with them the contextual knowledge of the program and the familiarity with the 'magic spells' driving it. As no one else was happy or able to maintain them, many programs collapsed (see for example, (Macro & Buxton, 1987; Ince, 1988; Glass, 1997)).

Over time organisations became more concerned with structure and efficiency. Procedures and approaches have been applied to increase order and add structure and control to the process of development. Indeed, it could be argued that the notion of the waterfall model partly stems from the dissatisfaction with the state of 'the art of programming' (Charette, 1986), the 'hacker running the show' (Booch, 1987; Ince, 1988), and the lack of general control over the programming process (Macro & Buxton, 1987). Structure was thus used to impose order on creative enterprises in an effort to involve larger teams and divide the work according to newly established management approaches emphasising the balanced distribution of man-month effort across wider groups.

The emergence of structured efforts to control programming and the development of concepts such as the waterfall life cycle coincided with the departure of many hackers from their existing organisations. Meanwhile, the wide spread adoption of more versatile computing systems such as System/360 and the subsequent emergence of mini and micro computers enabled hackers to reposition themselves and re-emerge in a wider variety of organisational and institutional settings.

Software houses, established in the fifties and sixties, when manufacturers were developing sophisticated product lines, were working for many of the larger organisations. Software houses were often staffed by outstanding programmers who cut their teeth on some of the early complex systems, before being released by large companies. When large corporations, military and the space agencies needed massive, complex and advanced computer software, or encountered problems on their projects, they would hire independent contractors and software houses. The experience these specialists accrued through their earlier projects and the dependence of larger manufacturers on their expertise, enabled them to hone in their skills and gain market appreciation. Others, took advantage of the easily available technology to experiment, develop, grow and position themselves as unquestioned leaders. Indeed, in 1980, when IBM decided to build personal computers, they hired Microsoft to build the operating system, thereby avoiding the need to engage with the development of a significant new operating system.

Many of the old constraints were changing and the limits of the machine were no longer a key concern. Certainly, by this point experienced hackers in a solid position to compete with the monolithic manufacturers as small start-up companies offered hardware as well as software with amazing capabilities that could reside on the top of a desk. Many hackers were already in business, offering their clients to design complex systems rapidly using innovative machines, code and tools. The result, was the encouragement of a new generation of aspiring programmers and hackers able to address a new set of challenges.

Shortages of qualified software professionals, reasonably priced powerful machines, and end user computing combined to create the ‘new age’ hacker. New age hackers had a good understanding of the software packages they were using and were capable of utilising the internal features of these packages to produce impressive results. ‘New age’ hacking was practised by individuals utilising the power on their desk within the confines of their offices, with limited quality assurance support, standards or even professional training. Steven Levy refers to the second generation of hackers, typically involved in the emergence of the PC, in contrast to the hackers of the 1950s and 1960s, which he classifies as first-generation hackers (Levy, 1985; Lin, 2007; p. 36). Being closer to the problem, the new hackers possessed an intimate knowledge of the problem and the technology which enabled them to be creative whilst maintaining the principled conceptual integration and developing a wider understanding of the overall architecture required to address the problem.

By 1985, the new generation of hackers, working alone or in small teams, was able to make good use of improved prototyping tools and approaches. Prototyping could utilise newly available higher order tools and technology to create more suitable computer programs, faster, by way of trial-and-error. Rapid prototyping reduced the cost of producing early programs through providing the technology for rapid fabrication. The notion of frozen requirements

could thus be replaced by communication and on-going dialogue, while speed of delivery could substitute for the mammoth time-spans typical of more traditional approaches involving large teams engaged in more bureaucratic cycles of conceptualisation, development, verification and delivery.

Prototyping is a creative and responsive effort that attempts to address users' needs. Once unleashed, it can prove difficult to control the rate of change, the length of the effort and indirectly, the cost implications. Many prototyping projects reported problems with controlling the number of iterations or the length of the prototyping/re-iteration phase. Costing prototyping projects is therefore, done primarily on the basis of timing the length of the prototyping effort or limiting the number of allowed iterations. The pervasive, rational and practical problem of putting a price and a boundary on creativity thus proved to be a major hurdle to utilising a truly creative act.

To address management concerns more controlled forms of prototyping and coordination were introduced. Approaches like the spiral model, Rapid Application Development (RAD), Scrum, agile development and the Dynamic Systems development Method (DSDM) place greater emphasis on the management, costing and control of the prototyping, or creative effort. The success of prototyping relies on the ability to obtain rapid fabrication of a definitive product. The application of strict management control to the process, secures the delivery of rapid functionality according to agreed guidelines. In fact, it represents, the imposition of agreed constraints on the creative process. (Interestingly, both the Spiral and RAD/DSDM/agile methods respond to the pervasive problems highlighted earlier. The Spiral controls the number of iterations through risk management, while, RAD/Scrum/agile responds to the length of the effort by imposing the concept of fixed timeboxing.)

Taking stock: Hackers gallery

The writing of Brooks conveys a strong admiration for creative programmers and heroic architects and their resulting products. Reflecting back at the end of the anniversary edition, Brooks maintains that the central argument in the book is the critical focus on conceptual integrity and the architect. Brooks maintains that elegant software products are typically designed by a single mind, or at most by a pair of minds (Brooks, 1995; p. 255), rather than by committee. He contends that the same can be said of well-loved books or musical compositions.

Levy (1985) re-positions hackers as the drivers of the computer revolution profiling the imaginative 'brainiacs' who found clever and unorthodox solutions to computer problems by taking risks, bending the rules and consequently pushing the world in a new direction. Levy's sympathetic account traces the hackers from the late 1950s labs to the emergence of desktop machines in the 80s—these would later morph into agile development, and are very much alive within the gaming community. Levy also introduces the hacker ethic, which (slightly paraphrased) advocates, that:

- Access to computers—and any other sources of important information—should be unlimited and total.
- All information should be free.
- Hackers should be judged by their hacking, and not by qualifications or position
- Art and beauty can be created on a computer

- Computers can change life for the better (and by implication therefore, hackers play a part in shaping and bringing about that shift)

Hackers have shaped many aspects of life through the development of products and artefacts that have powered further development within wider society. Table 1 identifies the names of a number of well-known, self-proclaimed hackers alongside the well-recognised products or companies that they had been associated with. Many of the products have been considered as extremely successful and innovative, and have played a part in shaping future generations of products. Intriguingly, the majority have had an active fan club at some point that recognised and appreciated the value and uniqueness of the resulting product.

Table 1: Hackers Gallery – a representative sample

Name	Association (or key product)
Bill Gates	Microsoft
John Page	HP
Charles Simoni	Microsoft (MS Word, Excel)
Jonathan Sachs	Lotus 1-2-3
Andy Hertzfeld	Macintosh OS
Wayne Ratliff	dBase III
Peter Roizen	T/Maker
Bill Gates	Microsoft
John Warnock	Adobe
Jeff Raskin	Apple Macintosh project
Gary Kildall	CP/M ooperating system
Bob Frankston	VisiCalc spreadsheet
Dan Bricklin	VisiCalc spreadsheet
Doug Englebart	Computer mouse
Mark Zuckerberg	Facebook
Richard Stallman	Gnu
Linus Torvalds	Linux Kernel
Eric Raymond	Open Source Initiative

The inventiveness of the hackers has enabled the wider computer revolution that has transformed society. Richard Stallman reflects on the values, concerns and work habits of hackers during a conference on the hacker community:

'*What they had in common was mainly love of excellence and programming. They wanted to make their programs that they used be as good as they could. They also wanted to make them do neat things. They wanted to be able to do something in a more exciting way than anyone believed possible and show "Look how wonderful this is. I bet you didn't believe this could be done*.' (Stallman, 1985).

Hacking is still alive and well While the early hackers may have enjoyed the intellectual challenge of overcoming the physical limitations of computer technology and capability, the

emerging hacker community appears to still think big and risk failure, whilst continuing to push the computer and what is viewed as possible beyond the current envelope of expectations. Hacking thus appears to revolve around combining an intimate knowledge of the problem to be solved (or the ambition) with knowledge and skill in the tools required to address the problem and the context and environment within which it is to be implemented. The key values, as always, are related to enabling a new capability, achieving a novel and unexpected outcome, outsmarting and extending the technology through unprecedented mastery, and enjoying the process of getting there. Brooks might also add a caveat around the need to maintain conceptual integrity and ability to delight the client, that chimes with the iconic status of many of the products identified in Table 1. The culture, ethics and principles associated with the act of hacking certainly offer an indication of the importance of, and belief in, purpose, and the need to focus on the wider perceived value for the user community.

Hacker communities

Brooks widely acknowledges that whilst conceptual integrity is essential to delivering significant products, the enormity and complexity of some undertakings requires engaged communities able to work with the architect to deliver useful products. In subsequent talks he alludes to the power of a small sub-community, populated with capable and talented programmers to rescue projects that become delayed and threatened. Indeed, working against impossible odds and pushing the boundaries of what can be achieved is a hall mark of the hacker approach. Isaacson (2014) further celebrates the ability of pioneers, hackers and innovators to collaborate, master the art of teamwork, foster new scales and dimensions of innovation and thereby usher in the digital revolution.

Nowadays, hacking is often associated with criminal tendencies (also known as black hat hacking) and security breaches. However, the original spirit of hacking (sometimes referred to as white hat hacking) is a pure form of innovation against all the odds, as technical experts challenge themselves to confront insurmountable problems. A further intriguing concept is the ability of hacking to create communities that are able to share, engage and work collaboratively.

Brooks refers to ancient cathedrals as examples of conceptual integrity, where each generation of architects involved in the ultimate construction, is happy to sacrifice some of their own ideas of design so that the whole might be of pure design (1995; p. 42). In this way, a multi-generational edifice is created that fuses together ideas and thinking into a satisfying whole. Raymond (1999) contrasts the cathedral view of development, where between releases or generations, each new element of work is restricted to an elite group, with the Bazaar model. In the Bazaar model, the main work is developed over the internet in full view of the public, allowing multiple participants and co-creators. This satisfies Raymond's maxim that '*given enough eyeballs, all bugs are shallow*'. In other words, Raymond maintains that the larger the number of testers and participants, especially through public testing, scrutiny and experimentation, the more rapidly that all latent bugs and errors will be uncovered. Therein lies a further potential value that can be derived from the development of communities of capable and interested hackers. However, the assertion that all bugs will be uncovered, and more rapidly addressed compared to planned discovery, is yet to be rigorously tested and verified.

Moreover, the bazaar approach allows for the mixing of different agendas and styles as individuals decide to observe, participate and mingle in different aspects. Rapid and frequent releases allow the content to develop and mature over time as the wider community gets to engage with it over multiple iterations. The development of Wikipedia has been described as the application of the Bazaar model through the engagement of an active community (Poe, 2006). Open source communities—sometimes also known as hackerspaces or makerspaces—have grown in many parts of software development and therefore carry the potential for speeding up and upscaling innovation (see for example, von Hippel & von Krogh, 2003; Kostakis et al., 2014; Söderberg & Delfanti 2015).

It is particularly noteworthy, that the example of Wikipedia offers a direct challenge to Brooks' assertion that the addition of person-power can actually delay or complicate the task, as Wikipedia's creation clearly attests to the value of an engaged community of participants who are keen to participate, develop, improve and co-create something better. Indeed, it would appear that adding more human power to some important initiatives can lead to faster, better and richer products. Does this imply that in truly collaborative, and voluntary endeavours, adding human-power can actually enhance and improve the results? Moreover, does the spirt of true co-creation enable us to overcome the constant struggle between creativity and structure? Does it offer new insights into how to manage or support creative development? And for that matter, given the success of Wikipedia, might the development of the intellectual equivalent of the next System/360 be done on a more public basis and deliver more successful results, at a faster pace?

What are the wider implications for innovation and for projects?

Having taken a rather wide sweeping historical perspective of the hacking approach and explored its roots in software development projects, it is time to return the focus to the future of projects. A culture that grew at the margin of the computer industry, now seems ready to offer the potential to reposition, recalibrate and realign innovation. As agile thinking, active experimentation ideas and open source concepts are entering mainstream business, there is an opportunity to utilise some of these new ways of thinking in order to foster better projects, improved outcomes and more satisfied users.

Could the new hacker make a difference in the wider business context? The challenge is to explore the potential and implications of hacking. Our relationship with technology has changed immensely over the past 50 years as computers have transformed most facets of our society. Digital natives, people brought up during the age of digital technology, are entering the workforce and the ideas of hacking may simply seem more natural to those who have grown up with computer games, continuous on-line presence and a wide variety of apps (Prensky, 2001; Bennett et al., 2008; Prensky, 2009; Palfrey & Gasser, 2011; Albright, 2019). This month's contribution by Tim Rayner is developed from his book, *Hacker culture and the new rules of innovation*, published by Routledge and aims to explore the rich potential of hacking as a timely, creative approach.

Rayner recognises that the organisational landscape has changed. Innovation implies creativity, entrepreneurship, self-organising teams, cheap experimentation and fast learning. It also requires an imaginative new mindset and Rayner is keen to explore the potential for importing some of the ideas that have transformed our world through new technologies.

Against the backdrop of smart cities, self-driving cars and wearable technologies, this is a good time to reflect on the pace of innovation. How did technology transform our lives so rapidly and so drastically in a span of a mere 50 years? Rayner attributes the fast progress to two main factors: open standards and innovation culture (Rayner, 2018; p. 2). Sharing and open standards enable entrepreneurs to build on each other's work, to grow innovation combinatorically, to continuously expand, improve and enhance. However, the second factor, the culture of technological innovation plays a critical role.

It would be simple to focus on the allure of promising technology. However, by resisting the call of the technology sirens to take the easy route, Rayner is therefore able to make a far important contribution; instead he draws attention to the requisite condition that has enabled the computer industry, and society at large, to benefit from such a tremendous rate of innovation: His focus is on the culture of innovation, the way people share, coordinate, work and collaborate to produce innovation.

In many ways Rayner's perspective provides a much-needed refresh and update to the view of a large firm engaged in a major innovation project that is imposed on the workforce. Here instead we have a democratised approach to the co-creation and co-development by consensus of cutting-edge innovation that is not limited by pre-ordained ambition or by accidental constraints. By looking at hacking as the vehicle for improvement and development, Rayner is able to offer a significant alternative to conventional technological innovation models.

Whilst the factor that appears to make the difference is hacking, it is hacking-in-the-large, in a wider social setting, rather than the initial hacking by very capable individuals that comes to the fore. The power of hacking seems to be multiplied when it is used to engage and connect complete communities, enabling people to stand on the shoulders of others, and thereby continually grow, stretch and expand what is possible and attainable. In a true community spirit, where there is interest and commitment, hacking can act as a powerful multiplier that enables entrepreneurship to flourish. The *hacker culture* that Rayner talks about seems to be particularly important in establishing the dynamics needed to engage the multiplier and benefit from a more involved and passionate form of social and communal problem solving.

Cultures that are shaped by hierarchy, tradition, obedience, egos, targets and KPIs will continue to struggle to deliver in an increasingly dynamic and evolving marketplace. Start-ups have a lot to offer and a great potential to benefit from such fresh thinking: Start-ups that embrace the technological revolution have been able to transform their offerings by taking a customer-centric approach and engaging with their emerging markets. Rayner is therefore documenting a new stage in the development of hacker culture as a wider and more encompassing pre-condition for achievement of a hitherto unexpected and unimaginable new level of innovation.

Rayner's re-focus in on the all-important social space. The hacker entrepreneur is far better connected and therefore more powerful than his poor cousin, the technical magician, from fifty years ago. Leaders interested in successful transformation in the digital age would do well to begin by looking at the principles identified by Rayner in the context of starting to embed innovation within more agile and flexible organisations with better designed innovation space. The principles he sets forth are essential for setting the scene for introducing meaningful innovation.

Experimentation, trial-and-error and re-adjustment are part of the standard repertoire of successful hackers. As we move to the realm of more social and strategically positioned hacking, perhaps the next iteration of the hacker gallery (represented in Table 1) would have specific communities and wider sections of society as the key members as organisational leaders become more comfortable with their role in facilitating large scale social experiments and transformations. It would be interesting to see how larger systems developed through hacking solutions would evolve over time. At the very least, it gives us another experiment and another approach to test as we continually seek to improve, evolve and grow.

Furthermore, it enriches the conversations and provides much needed inspiration and the beginning of an important new dialogue in this arena. The tension between creativity and structure through various management regimes has shaped development and progress over generations. Ultimately, the thinking put forward by Rayner offers the potential to reignite innovation, create a connected social focus within the enterprise, and prepare the ground for delivering a new form of entrepreneurial management—a much needed strategic capability in uncertain and unpredictable times.

References

Albright, J. M. (2019). Left to their own devices: How digital natives are reshaping the American dream. New York: Prometheus Books.

Ascher, W. (1993). The ambiguous nature of forecasts in project evaluation: Diagnosing the over-optimism of rate-of-return analysis. *International Journal of Forecasting, 9*(1), 109-115.

Bennett, S., Maton, K., & Kervin, L. (2008). The 'digital natives' debate: A critical review of the evidence. *British journal of educational technology, 39*(5), 775-786.

Booch, G. (1987). *Software engineering with Ada*. San Diego: Benjamin-Cummings Publishing Co.

Brooks Jr, F. P. (1975). *The Mythical Man-Month: Essays on Software Engineering*. Reading, MA: Addison Wesley.

Brooks Jr, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E*. Reading, MA: Addison Wesley.

Charette, R. N. (1986). *Software engineering environments: concepts and technology*. New York: McGraw-Hill.

Cortada, J. W. (2019). *IBM: The Rise and Fall and Reinvention of a Global Icon*. MIT Press.

Dalcher, D. (1999). Simplicity and power: when more means less. *IEEE Computer*, (5), 117.

Dalcher, D. (2013). Project economics: Wishful thinking, conspiracy of optimism or a self-fulfilling prophecy? *PM World Journal*, 2(12), December 2013, pp. 1-4. https://pmworldlibrary.net/wp-content/uploads/2013/12/pmwj17-dec2013-dalcher-project-economics-SeriesArticle2.pdf

Fishman, K. D. (1981). *The computer establishment*. New York: McGraw-Hill.

Flyvbjerg, B. (2008). Curbing optimism bias and strategic misrepresentation in planning: Reference class forecasting in practice. *European planning studies*, *16*(1), 3-21.

Flyvbjerg, B. (2009). Optimism and misrepresentation in early project development, In *Williams, T., Samset, K., & Sunnevåg, K. (Eds.). Making essential choices with scant information: front-end decision making in major projects*. London: Palgrave Macmillan, London, pp. 147-168.

Glass, R. L. (1997). *In the beginning: Recollections of software pioneers.* Los Alamitos, CA: IEEE Computer Society Press.

Hippel, E. V., & Krogh, G. V. (2003). Open source software and the "private-collective" innovation model: Issues for organization science. *Organization science*, *14*(2), 209-223.

Ince, D. (1988). *Software development: Fashioning the baroque*. Oxford: Oxford University Press.

Isaacson, W. (2014). *The innovators: How a group of inventors, hackers, geniuses and geeks created the digital revolution*. Simon and Schuster.

Jordan, T., & Taylor, P. (1998). A sociology of hackers. *The Sociological Review*, *46*(4), 757-780.

Kostakis, V., Niaros, V., & Giotitsas, C. (2015). Production and governance in hackerspaces: A manifestation of Commons-based peer production in the physical realm? *International Journal of Cultural Studies*, *18*(5), 555-573.

Levy, S. (1984). *Hackers: Heroes of the computer revolution. New York: Bantam Doubleday*.

Lin, Y. W. (2007). Hacker culture and the FLOSS innovation. In *Handbook of research on open source software: technological, economic, and social perspectives* (pp. 34-46). IGI Global.

Love, P. E., Edwards, D. J., & Irani, Z. (2011). Moving beyond optimism bias and strategic misrepresentation: An explanation for social infrastructure project cost overruns. *IEEE Transactions on Engineering Management*, 59(4), 560-571.

Macro, A., & Buxton, J. N. (1987). *The Craft of Software Engineering*. Wokingham: Addison Wesley.

Palfrey, J. G., & Gasser, U. (2011). *Born digital: Understanding the first generation of digital natives*. New York: Basic Books.

Poe, M. (2006). THE HIVE-Can thousands of Wikipedians be wrong? How an attempt to build an online encyclopedia touched off history's biggest experiment in collaborative knowledge. *Atlantic Monthly*, *298*(2), 86.

Prensky, M. (2001). Digital natives, digital immigrants part 1. *On the horizon*, *9*(5), 1-6.

Prensky, M. (2009). H. sapiens digital: From digital immigrants and digital natives to digital wisdom. *Innovate: journal of online education*, *5*(3).

Raymond, E. S. (1997). *The cathedral and the bazaar*. Sebastopol, CA.: O'Reilly Media.

Rayner, T. (2018). *Hacker culture and the new rules of innovation*. Abingdon: Routledge.

Schumacher, E. F. (1973). *Small is beautiful: A study of economics as if people mattered*. New York: Random House.

Sharot, T., Riccardi, A. M., Raio, C. M., & Phelps, E. A. (2007). Neural mechanisms mediating optimism bias. *Nature*, *450*(7166), 102.

Sharot, T., Korn, C. W., & Dolan, R. J. (2011). How unrealistic optimism is maintained in the face of reality. *Nature neuroscience*, *14*(11), 1475.

Sharot, T., Guitart-Masip, M., Korn, C. W., Chowdhury, R., & Dolan, R. J. (2012). How dopamine enhances an optimism bias in humans. *Current Biology*, *22*(16), 1477-1481.

Söderberg, J., & Delfanti, A. (2015). Hacking hacked! The life cycles of digital innovation. *Science, Technology, & Human Values*, *40*(5), 793-798.

Sparkes, M. (2016). IBM's $5bn gamble: Revolutionary computer turns 50. *Telegraph*, 7th April 2016. https://www.telegraph.co.uk/technology/news/10719418/IBMs-5bn-gamble-revolutionary-computer-turns-50.html
Accessed 23 June, 2019.

Stallman, R. (1985). Interview, in, *Hackers-Wizards of the Electronic Age*, TV documentary, KQED.

About the Author

Darren Dalcher, PhD

Author, Professor, Series Editor
Director, National Centre for Project Management
Lancaster University Management School, UK

Darren Dalcher, Ph.D. HonFAPM, FRSA, FBCS, CITP, FCMI SMIEEE SFHEA is Professor in Strategic Project Management at Lancaster University, and founder and Director of the National Centre for Project Management (NCPM) in the UK. He has been named by the Association for Project Management (APM) as one of the top 10 "movers and shapers" in project management and was voted Project Magazine's "Academic of the Year" for his contribution in "integrating and weaving academic work with practice". Following industrial and consultancy experience in managing IT projects, Professor Dalcher gained his PhD in Software Engineering from King's College, University of London.

Professor Dalcher has written over 200 papers and book chapters on project management and software engineering. He is Editor-in-Chief of *Journal of Software: Evolution and Process,* a leading international software engineering journal. He is the editor of the book series, *Advances in Project Management*, published by Routledge and of the companion series *Fundamentals of Project Management*. Heavily involved in a variety of research projects and subjects, Professor Dalcher has built a reputation as leader and innovator in the areas of practice-based education and reflection in project management. He works with many major industrial and commercial organisations and government bodies.

Darren is an Honorary Fellow of the APM, a Chartered Fellow of the British Computer Society, a Fellow of the Chartered Management Institute, and the Royal Society of Arts, A Senior Member of the Institute of Electrical and Electronic Engineers, a Senior Fellow of the Higher Education Academy and a Member of the Project Management Institute (PMI) and the British Academy of Management. He is a Chartered IT Practitioner. He sits on numerous senior research and professional boards, including The PMI Academic Member Advisory Group, the APM Research Advisory Group, the CMI Academic Council and the APM Group Ethics and Standards Governance Board. Prof Dalcher is an academic advisor for the *PM World Journal*. He is the academic advisor and consulting editor for the next edition of the APM Body of Knowledge. He can be contacted at d.dalcher@lancaster.ac.uk.