The relevance of Open Source to hydroinformatics

Hamish Harvey and Dawei Han

ABSTRACT

Open Source, in which the source code to software is freely shared and improved upon, has recently risen to prominence as an alternative to the more usual closed approach to software development. A number of high profile projects, such as the Linux operating system kernel and the Apache web server, have demonstrated that Open Source can be technically effective, and companies such as Cygnus Solutions (now owned by Red Hat) and Zope Corporation have demonstrated that it is possible to build successful companies around open source software. Open Source could have significant benefits for hydroinformatics, encouraging widespread interoperability and rapid development. In this paper we present a brief history of Open Source, a summary of some reasons for its effectiveness, and we explore how and why Open Source is of particular interest in the field of hydroinformatics. We argue that for technical, scientific and business reasons, Open Source has a lot to offer.

Key words | free software, open source, software framework, software development, scientific method, hydroinformatics

Hamish Harvey (corresponding author)
Dawei Han
Water and Environmental Management Research Centre,
Department of Civil Engineering,
University of Bristol,
Bristol BS8 1TR, UK
Tel: +44 117 9289768;
Fax: +44 117 9289770;
E-mail: *david.harvey@bristol.ac.uk; d.han@bristol.ac.uk*

INTRODUCTION

Computer software is the lifeblood of hydroinformatics. Until now a great deal of it has been obtained by transfusion from the fields of computational hydraulics and hydrology, which existed long before the term hydroinformatics was coined. It is clear from a review of the hydroinformatics literature (in this journal and in the conference series of the same name) that this—the 'hydro' part of the discipline—is, while naturally still vital, increasingly playing second fiddle to the more 'informatics' area of, for example, large scale distributed decision support systems. This is natural, as one of the main aims of hydroinformatics is the enfranchisement of stakeholders, allowing those who have until now been merely *objects* in the decision making process to take an active part in it (Abbott 1998). Existing modelling software, being aimed at specialists in hydrology and hydraulics, falls far short of this goal and a much broader and more integrated use of computer technology will be fundamental to the development of this field.

While many of the component parts of these integrated systems already exist, their use in this context is experimental, and thus we can think of the software systems we create (for example Yan *et al*. 1999) as analogous in many ways to physical experimental apparatus. Such apparatus must be dismantled and rebuilt several times, with the design being refined progressively. Abbott & Jonoski (1998) allude to this with a reference to *evolutionary prototyping*. While the architecture and the philosophy behind hydroinformatics systems are topics of active research (the former in particular by the present authors), little consideration has been given to how best to undertake the associated software development.

The prevalent approach to software development might be known as *closed source*, *non-free* or *proprietary*, where software is written for sale or for in-house use. The operating systems and applications that most people use every day fall into this category, as does the huge volume of software that is used only within the organisation that developed it. Eric S. Raymond, a self-styled analyst of the

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

Open Source movement, estimates that as much as 95% of code is written in-house (Raymond 1999). Whether this value is reliable or not, what the average personal computer user generally thinks of as 'computer software' is just the tip of the iceberg. Whether for in-house use or sale, however, software is almost always treated by the author or the author's employer as being a primary asset.

In recent years an alternative treatment of software development has risen in profile, first as a result of the steady progress of the Free Software Foundation (FSF) towards a complete, free Unix compatible operating system called GNU (Stallman 1998), then through the dramatic rise to prominence of the Linux kernel which, when combined with the GNU software, finally represented a viable alternative to Windows and the myriad proprietary flavours of Unix for an increasingly broad range of applications. It should be noted that 'free' is used here in the sense of 'freedom', referring to the freedom to use and modify the software as desired without restriction. Note also that *commercial* is not an opposite of Open Source—the majority of Open Source advocates are in no way anti-commercial, and the ongoing effort to find business models which can effectively harness the technical power of Open Source development is discussed in this paper.

It is increasingly the case that no one organisation can provide all of the specialist knowledge required for hydroinformatics software development projects. This is demonstrated by the increase in the number of collaborative projects being undertaken in the field, and of the formation of strategic links between otherwise competing institutions. The resulting complex network of non-disclosure agreements (explicit or implicit), required on the current business model in order to protect the commercial interests of the institutions involved, is a severe impediment to development and, as a result, the overall expansion both of knowledge and of the entire market for hydroinformatics products suffers. By contrast, the freedom gained by utilising the Open Source model, provided these institutions can modify their revenue generation methods appropriately, could accelerate development, supporting a much more rapid expansion of the total market. It is recognised that it is hard to encourage this sort of cooperative approach in a competitive marketplace, but the hydroinformatics world is perhaps small enough and strongly enough knowledge-based that this can be achieved.

It is the authors' belief that this approach—the Open Source development methodology—provides a potentially very useful alternative to the closed source methods which currently predominate. This paper explores the closely related ideas of Open Source and Free Software. We first provide an outline of their history and a summary of the main concepts, and then discuss the technical, science and business arguments for the Open Source development model, and the benefits to the end user, with a particular focus on hydroinformatics.

CLOSED AND OPEN SOURCE SOFTWARE

Closed source

It is a common belief that the *closed source* or *non-free* model of software development is 'traditional', and while this is not strictly true, this model has gained such a hold that it makes little difference. In the interest of accuracy, however, it should be pointed out that, in its early days, computing was an academic pursuit, and software was freely shared among the developer and user communities, which were largely coincident (Stallman 1998). That aside, it has long been the case that software is written within a company, and where that software is provided for use outside of the company, the providers go to great technical and legal lengths to protect their profits from the sale of the software. The use of copy protection schemes, network license managers and dongles are among the technical methods, while copyright is the primary legal device. In the United States it is also possible to patent software, and while this is a matter of great controversy and related to the topic under discussion here, it is beyond the scope of this paper (see Babovic (2000) for some web links relating to software patents).

Free software

Open Source is an outgrowth of the Free Software movement, started by Richard M. Stallman (usually referred to

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

as RMS). RMS was a system hacker at the MIT AI Lab in the 1970s ('hacker' being correctly used to mean essentially 'one who loves programming', as distinct from 'cracker', which has the meaning that the press have incorrectly assigned to hacker) and grew disillusioned with the growing trend for software suppliers to refuse to allow access to the source code for software. Since, then as now, this frequently left the users of software waiting for a long time (often indefinitely) for a needed fix or modification to a program, RMS developed a set of values that essentially states that non-free software infringes on the users' rights, and set about writing a complete operating system that provided users with those rights (Stallman 1998). He formed the Free Software Foundation (FSF) in 1985 to administer funds and resources for this development.

Note that *free* is used in this context in the sense of *free speech* and the French 'libre', and has nothing to do with cost. We will discuss the economic implications later. The Free Software Definition (FSF 2001a) states that, for software to be free, a user must have four freedoms (the numbering, betraying RMS' programming background, starting at zero). For the second and fourth criteria to be fulfilled the source code of the software must be available.

1. The freedom to run the program, for any purpose.
2. The freedom to study how the program works, and adapt it to your needs.
3. The freedom to redistribute copies so you can help your neighbour.
4. The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.

Open Source

RMS' admirable dedication to the cause of Free Software generated a large quantity of excellent software, most of it modelled on existing tools from the various proprietary Unix-based operating systems. However, the development of a free operating system kernel took much longer than expected (Stallman 1998). The users of the GNU tools, as they are called, still had to run them on a proprietary core. Linus Torvalds, a Finnish computer science student, wrote a Unix compatible kernel for his Intel 80386 PC, and released the source code for Linux, as he called it, under the FSF General Public License, or GPL (FSF 1991). He chose to do this, not for strong ethical reasons, but practical—he did not intend to develop it into a commercial product, and thought other computer scientists and programmers might be interested in experimenting with it. He was right, and quickly started receiving 'patches' (specially coded modifications) to the source code. These he applied to the 'official' source code repository, and released new versions of the kernel code very rapidly, thus maintaining the interest of what could now be regarded as co-developers by disseminating improvements rapidly. Soon the Linux kernel was sufficiently versatile and stable that, combined with the GNU tools, it represented a complete, free operating system, GNU/Linux. This is the point at which the IT industry started to pay attention to the Free Software movement.

Torvalds had unwittingly stumbled on what proved to be an astonishingly successful development paradigm, enabled by the principles of Free Software, but which had not been utilised by the Free Software Foundation itself. By encouraging users to modify the software and submit patches, then evaluating and possibly applying those patches, and finally releasing the patched whole back to the community, Torvalds harnessed a massive talent pool prepared to work for mutual benefit rather than immediate financial gain. In fact, he turned Pareto's Law (Dafermos 2001) to his advantage—he provided the initial 10% of effort in creating 90% of the required functionality, then effectively 'outsourced' the remaining 90% of effort to complete and debug the kernel.

The community which grew up around GNU/Linux, and the rapidity of its growth, took everyone, including Torvalds, by surprise. The demonstrable success of the approach started to attract the interest of the establishment, and the dogmatic attitudes of the Free Software Foundation were felt by some to be an impediment to adoption of Linux and related technologies by businesses. Painfully aware of a rapid loss of web browser market share to Microsoft's Internet Explorer, and inspired by the first considered study of the Linux approach to software development (Raymond 1999), Netscape Communications decided to release the Netscape browser source code in

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

the hope of attracting external programming talent, and in the discussion surrounding that decision, the more business friendly label 'Open Source' was coined. It is often suggested that it was this period—from the publication of Raymond's paper to the open-sourcing of the Netscape browser—that transformed the Open Source movement from an oft-derided subculture into a force to be reckoned with.

The Free Software and Open Source movements have very different, though not mutually exclusive, guiding principles: Free Software came from a strong sense of moral duty, whereas the Open Source view grew from observations of highly successful software development projects (in particular Linux). The rest of this paper discusses the practical, scientific and economic benefits of the Open Source model of development to the hydroinformatics community, and thus at this point we leave the discussion of Free Software principles. Yee (1999) provides an interesting survey of the implications of Free and non-Free software in the context of ethical trade and development.

Examples of Open Source projects

There are numerous examples of successful Open Source projects that could be presented. Linux has already been discussed. A look at *http://www.freshmeat.net* (an online catalogue of Open Source software) and *http://www.sourceforge.org* (a free hosting service for Open Source projects) can provide some impression of the quantity of software produced or under development, if not its quality. While it is difficult to find information on unsuccessful Open Source projects, this should not be understood to mean that Open Source is a 'magic bullet'. It is, however, important to remember that one factor in the success of the Open Source method as a whole is a continuous Darwinian process of natural selection in the Open Source ecosystem, ensuring rapid development and innovation. Notable Open Source successes include the Linux operating system kernel, BSD Unix and the Apache web server, which powers 50–60% of the world's web sites (Netcraft 2001). Open Source science and engineering software is less widely known but does exist, including CFD and finite element software, simulation software and a complete Geographical Information System. Links to information about some specific projects are provided in 'Web Clips' in this issue.

THE TECHNICAL CASE FOR OPEN SOURCE

With Open Source placed in its historical context, we can now examine why it represents a technically successful software development method. In this section we look at the main features of the Open Source development model, which result in it producing—in at least some cases—high quality software at a rapid pace. We also give some consideration to the types of software that seem to be particularly well suited to Open Source development.

Why Open Source works

That a large, geographically distributed group of developers and users could combine their efforts and produce a high quality operating system kernel such as Linux took most observers of the computing industry by surprise. In fact, as explained in the preceding sections it should not have done, since a substantial portion of the Internet infrastructure, and even the TCP/IP networking code in Microsoft Windows, was developed on this model. It is not new, but it is effective, and one result of the recent attention it has gained is that the reasons for this effectiveness have been examined.

The fundamental principles of Open Source, as set out in the Open Source definition (OSI 2001), dictate that, when a piece of software is released as Open Source, it includes full source code and permission for any user to modify, and to distribute modified or unmodified versions with or without charge. These are considerable freedoms when compared with the typical End User License Agreements accompanying proprietary software, but they come with a responsibility, often also encapsulated in the license; any changes made to the software must be given back to the community. In this way, the work of many developers can be utilised in the creation of the software.

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

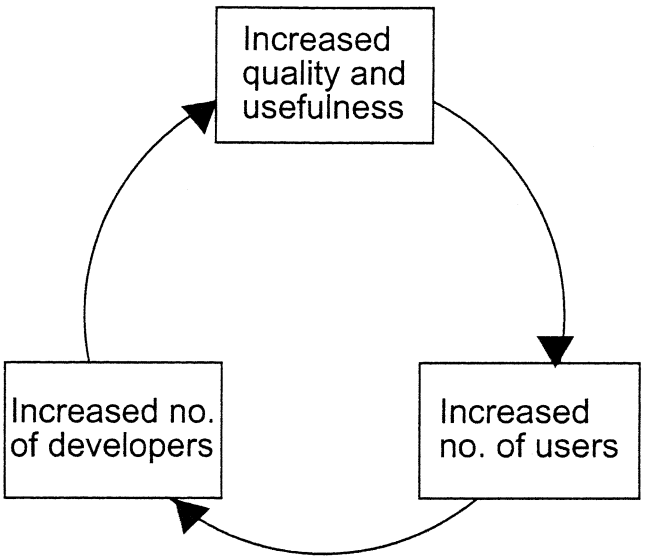

Figure 1 | A positive feedback cycle of growing use and increased rate of development.

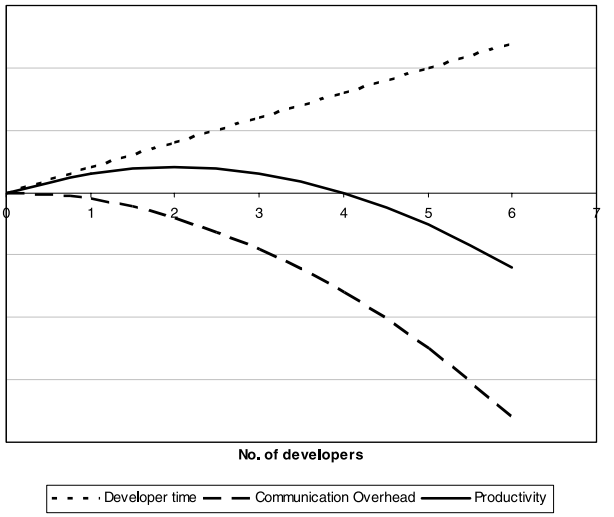

Figure 2 | Brooks' Law, generally applicable to development of complex software using teams of people, results in the growth of communication overhead rapidly overwhelming any benefit from increasing team size.

This can result in a positive feedback cycle in which increasing quality results in a growing user base and therefore a growing developer pool (Figure 1).

The principal reason for surprise that this approach succeeds in producing any software at all is that the software industry has for a long time taken Brooks' Law (Brooks 1995) as a truism. Brooks' Law states that the advantage of splitting work among N programmers is proportional to N, while the communication overhead is proportional to N^2 (Figure 2). Another reason for surprise is the common, but false, assumption that Open Source projects are anarchic in nature. In fact, along with the legally protected rights and responsibilities set out in the license are a set of widely accepted values and management structures that can be observed in many successful Open Source projects (Himanen 2001; Dafermos 2001). Successful Open Source projects generally have a strong leadership structure, often with a single individual in control, but sometimes, as in the Apache case, a team. As a project grows a team leader will accumulate a team of trusted assistants, which allows the leader to focus on overall management issues and not spend too much time assessing new modifications. Not just anyone is allowed to make direct modifications to the code. In theory, anyone could take the code, modify it and place it on his or her own web site, thus creating a competing version. In fact, this rarely happens, partly because of social taboos and a strong Open Source ethic (Raymond 1999) but also because to do so is unlikely to benefit anyone. That these structures and their working methods develop organically with the project does not appear to make them less effective.

Using a strongly modular structure can reduce the effects of Brooks' Law in development. The Linux kernel is an example of this—the interfaces between the kernel and device drivers to control hardware are well defined and documented, so a new developer can write a device driver without needing to alter code in the rest of the kernel. In this way many developers can actually work on the same project simultaneously without incurring excessive communication overheads. Many Open Source projects use the Concurrent Versioning System (CVS) which streamlines the process of applying changes from multiple developers to the master source code, and allows separate development and stable branches to be maintained simultaneously.

One area in which Brooks' Law really does not apply is that of debugging—the release of source code with software and the resulting empowerment of the users results in the remarkable debugging efficiency which is a

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

hallmark of Open Source development. This is also encouraged by early and frequent releases of the source code, usually with a clear indication that this is not the recommended stable release. In this way, adventurous and technically adept users can use the bleeding edge of development and, since they have access to the source, can characterise bugs for developers to fix, or can even supply a fix themselves.

Another benefit of Open Source in terms of the quality and innovation is the process of natural selection by which changes are adopted (Asklund & Bendix 2001). In traditional software development, a change request will be made: this request will then be assessed and either accepted for implementation or rejected. If accepted, the change will be implemented. In the Open Source model, a developer who wishes to see a feature will implement it, and submit the implementation for approval. If the project leaders are unhappy with either the feature itself or its implementation, it will be rejected. If a user requests a feature and the leaders agree it will be useful, it will be added to a wish list. More than one developer may implement the feature, and the best implementation will be chosen. This encourages an evolutionary improvement of software and increases the likelihood of an optimal implementation, at the expense of a certain amount of wasted developer time. Dyson (1998) provides a good summary:

> '. . . open source is not just about bug-fixing, which can be done in a massively parallel fashion. It's about competing innovations, which live or die in a market of other people deciding to adopt/enhance them, or ignore them. Interestingly, this is an information market . . . The Microsoft approach keeps asking: Where is the leader? And the OS approach keeps answering: The leader is whoever is ahead, by acclamation.'

Developer pool size

An issue which needs addressing in the context of the Open Source development of hydroinformatics software is that of developer pool size. The Linux operating system kernel is of use to such a large number of people, many of them excellent software developers, that finding sufficient talent prepared to work for no income is possible. In hydroinformatics this may not be the case and it might be feared that Open Source would therefore not work.

Not all of the benefits of Open Source are dependent on spreading the development. In the long run the developers of a widely adopted Open Source product might need to find ways of funding the development, but the end user benefits (discussed below) of protection from vendor lock-in and access to source code, should the developers be unable or unwilling to service the users' needs, still remain. Once a piece of software has become established, it should become clear to technically knowledgeable users that it is in their interests to contribute some time to development. Indeed, this process (release of an adequate project, followed by recruitment of project participants) is the start of the cycle described in Figure 1, and reinforces one of the central principles of successful Open Source projects: 'Show me the code'. In other words, starting an Open Source project with an idea rather than working (albeit imperfect) code is likely to fail.

An implicit assumption is that all Open Source developers are not paid for their work, which is not completely true. Some Linux kernel developers are paid by GNU/Linux distribution companies such as Red Hat. Apache web server developers are paid by major users of the program and by companies, such as IBM, who build enterprise web server software suites round the Open Source core. If hydroinformatics software companies can be persuaded of the benefits of Open Source collaboration on certain types of software (in particular frameworks, as discussed below) then funded developer time could be applied to such work.

Academic researchers should find the opportunity to have their work adopted more widely and more rapidly enticing, with widespread use enhancing their status in the academic world. The authors also feel that research funding agencies should consider whether they could obtain better value for money by encouraging researchers to open-source software produced in the course of funded work.

Software frameworks

A fourth-generation hydrological modelling tool is one that is usable by a domain expert with little or no knowledge of how the software is constructed or functions

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

internally (Abbott 1991). We believe that the fifth generation, whatever its other attributes, will only arrive with the ubiquitous acceptance of the component model of software construction, and hence flexible integrated modelling and decision making tools. These are the *multi-method, multi-model* systems foreseen by Cunge & Erlich (1999). It should be noted that the model integration projects typical of attempts to create more integrated modelling environments (for example ten Cate *et al.* (1998) and Tomicic & Yde (1998)), while important in the short term, do not fit in this category, since the integration is specific to a given set of models. Khatibi *et al.* (2001) envisage modular, 'plug and play' systems, based around an open architecture, for use in flood forecasting and warning. Schellnhuber & Hulme (2001) talk extensively of model integration on a grand scale for the assessment of climate change impact. For such systems to become a reality, we need a universal modelling *infrastructure*, which requires some level of agreement on that infrastructure across the industry.

To design and implement a framework of sufficient technical flexibility to allow us to move towards the fifth generation of hydroinformatics tools is a formidable task in itself for a single organisation. To achieve the adoption of that framework as a *de facto* standard is a near impossibility, particularly since all players other than the originator of the framework will view the motives of the originator with a healthy scepticism. While Microsoft can overcome this by the abuse of its market share (Jackson 1999), no hydroinformatics software supplier is in such an enviable position. One possible solution to this problem is in the *de jure* approach to standardisation, in which all of the (major) players conduct a design by committee exercise to create a standard to which they are then expected to adhere. In many of these cases, the result is a standard with which all players are equally dissatisfied, and as a result equally unlikely to make much use of. If they do, then there will tend to be multiple competing implementations, which interpret the standard slightly differently (Spangler 2001).

In this area, we would be wise to look to the example of the Internet Engineering Task Force (IETF), the standardisation body of the Internet. This organisation conducts its work through mailing lists and meets only three times a year. It is open to anyone to join the lists or attend the meetings, and the only barrier to participation in the full process is that you must demonstrate that you understand what you are talking about, and must not have a hidden agenda (O'Reilly 1998). Yet this seemingly haphazard, almost organic entity, succeeds in maintaining and advancing the interoperability standards (TCP/IP, HTTP, the email transmission format RFC 822, and many, many more) without which the Internet would never have happened or would have splintered into commercially controlled fragments years ago. The principal rule for proposing a new standard to the IETF is that you must have running code—a standard is useless without a reference implementation, and a reference implementation must be open to be useful. The mailing list discussions can then refine both the standard and the reference implementation until the consensus is that the former is correct, and the latter is an accurate implementation of it. At the end of this process there exists a fully working implementation of the standard, which can be used by anyone, at least in prototyping work (since, depending on the license, it may not be possible to use the reference standard in proprietary software).

The availability of full source code to the framework, even if initiated by a company, can act as a reassurance to competitors that the company, once it achieves widespread adoption of the framework cannot begin to abuse that adoption by exacting high fees or simply by abandoning the development of the foundations of another's product.

THE SCIENCE CASE FOR OPEN SOURCE

The technical merits of Open Source discussed above are increasingly well understood and accepted. Another important issue, not peculiar to the field of hydroinformatics, but certainly not common in the current range of Open Source projects, is how the Open Source development method integrates with the scientific method. The processes involved in Open Source software development and in the scientific process are closely related—unsurprisingly, since the Open Source method

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

grew out of the early computer science community, which valued the open exchange of information as highly as any scientific field. In this section we look at this relationship and consider how it can be used to advantage, consider how this may be of advantage to the field, and examine some problems which may arise at the interface between software and science development.

Parallels between Open Source and science

The principles by which Open Source development works, and the ethical viewpoint of the FSF, are often described as being communist in nature. Himanen (2001), however, points out that communism is a centralised (statist) model, which the free software community dislikes—the level of anti-censorship feeling on such hacker hangouts as Slashdot (*http://www.slashdot.org/*) makes this clear. In fact, free software is no more communist than academic research. It is a model of sharing based on the belief that this sharing is in everyone's best interest, just as the scientific publishing model developed precisely because it maximises the efficiency of scientific development. It has been observed (Raymond 1999) that most Open Source projects start when a single developer decides to start working on a problem that they regard as interesting, and grow as more developers who share that interest begin to collaborate. In choosing to work on Open Source at all, these developers are seeking peer recognition—the vast majority have day jobs and thus do not develop Open Source software for survival reasons. Raymond draws a parallel between the world of Open Source and the anthropological idea of a gift culture, where the basics are taken care of and social status is gained by giving things away, rather than by accumulating. In a similar fashion, most developers of scientific information (such as many of the readers of this journal) work on problems they find interesting and are driven to publish their research in order to gain recognition for the quality of the information they are producing. As with developers of Open Source software, academics operate to a great extent within a gift culture.

Open Source software development and academia essentially share a common model, that of 'an open and self-correcting process' or, in more detail (Himanen 2001, p. 68):

> 'The scientific ethic entails a model in which theories are developed collectively and their flaws are perceived and gradually removed by means of criticism provided by the entire scientific community.'

In addition, along with rights come obligations, and in the case of science, as in that of Open Source software, the right to use, criticise and develop a solution (theory or program) comes with two (Himanen 2001, p. 69):

> '. . . the sources must always be mentioned (plagiarism is ethically abhorrent), and the new Solution must not be kept secret but must be published again for the benefit of the scientific community.'

When looked at in this light it is no longer so surprising that the Open Source development model has proved to be so successful at creating software of high quality—the model on which this development works is a tried and tested one. In addition, it can be argued that (in terms of the quality of software produced and the rate of development and integration of new ideas) closed source software development must be expected to be less efficient, since this self-correcting cycle of study, development and publication is broken.

Quality control

It is not enough, however, to simply create some software, or a scientific theory, and publicise it to gain the social status that is being sought; some form of quality control is required. Without it there would be no way to reason about the validity of any theory or piece of software without extensive testing. Both Open Source and academia have an effective quality control mechanism and here the similarities between the two become even more striking.

The process of publishing the results of academic work in peer-reviewed journals is a primary element of the academic quality control mechanism. The reviewers will decide if a given paper is relevant to the journal's readership, and if its quality is sufficient for publication. If the latter is not true they may decide to reject it or to require modifications to be made before further consideration is

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

given. The readers of the journal can then expect a minimum quality level in all the papers published in that journal. The status of a particular journal is largely a function of how well the community at large thinks that the reviewers do their job. If a journal loses status, then the reviewers may be replaced or competitors to the journal may emerge—either way, the community ensures the continued relevance and quality of published work.

Likewise with Open Source, there is an ongoing process of review. The originators of a project are the reviewers at the outset, and in most cases retain that position for as long as they maintain an interest in the project. In the interests of the community it is expected that if a project leadership loses interest they should pass it on—if a project seems to have been abandoned, a new leadership may emerge, though they should first try to obtain the support of the old as a matter of etiquette. The project leaders have control over what changes are made to the main source code repository. In general, newcomers to the project will send patches for changes they make to the project leaders. The leaders may then decide to incorporate those changes or not. As the leaders gain confidence in the quality of the contributions made by a particular developer they may allow them to commit changes directly to the repository, which makes the process smoother.

In the case that a significant number of potential contributors find that the leaders of a project are not interested in the work they are doing, they may decide to abandon the attempt to get it included in the official project. This is the situation in which *forking* occurs. Forking means that, from the point at which the decision to form a fork is made, two separate sets of source are kept, one by the original team and one by the new. There is then a level of competition between the two branches of the project. Tradition will tend to favour the original, unless the forked variant offers something significant that the original does not, in which case developers and users may switch. Before they do, however the new team will generally have to display competence and commitment to the project, and show that their blocked contributions are indeed valuable. Note that the Open Source licenses generally allow forking, but that it happens very rarely since the project leaders are unlikely to persistently reject high quality contributions, and most contributors would rather spend their time making useful contributions to the original project than trying to wrest control of the project from the incumbent leadership.

Benefits of Open Source in a scientific field

Schmidt & Porter (2001) suggest that Open Source can be effective within communities with unmet software needs, citing science as one such area. Their argument is that the market represented by scientists is not large enough for the software industry to be interested in, which is clearly true in general, so scientists have a very good reason to produce Open Source software if they do not want to leave research and start a specialist software company. We would add to this the observation that the vast majority of all the scientific development work undertaken in hydroinformatics takes a long time to filter through into general practice, if it ever does so at all. This suggests that, even having, as we do, our own software industry, we are inefficient at utilising the results of scientific research. Two reasons for this are that the software produced by research groups is, in general, closed source and standalone, and that it is unable to integrate with the systems in general use in the industry. The barriers to entry in the software market are high, so much of this software never gets past the stage of being experimental, even if binaries are distributed at no cost.

In addition to this issue of the transfer of science from research into practice, releasing the software produced in the process of scientific research as binaries only interferes with the open scientific method and stifles development. Much work in the fields of hydrology and hydraulics, and thus work that is fundamental to the success of hydroinformatics, is in the development of new and (it is hoped) improved models of the processes of interest. These models are generally implemented as computer software. If, as is often the case, models are the result and subject of ongoing scientific investigation, then in theory it would be possible for multiple developers to implement that model from the scientific literature alone. Indeed, for such models to be properly peer reviewed (for the science to be validated by independent verification of the results) then

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

independent researchers must either have access to the code used in the original research, or they must re-implement it. The latter option is a waste of resources. The former can be partially achieved by giving a closed source version of the implementation freely to researchers while charging for commercial use—this free access is 'free as in beer' and as such might include the source code under a non-disclosure agreement. It must be understood that releasing source under a non-disclosure agreement is not Open Source, since it is fundamental to Open Source that anyone is allowed to make and distribute modifications. This limits the effectiveness of the scientific method, as modifications cannot be made, released and further tested without generating increasingly complex web of agreements and cross-licenses.

One possible solution to this problem is to release such implementations as Open Source. It is recognised that researchers who hope to capitalise on any potential future success of their model may be reluctant to do this, since the assumption would be that commercial users are unlikely to pay for software they can obtain at no cost. We hope the arguments presented below and in the references about business aspects of Open Source may convince some model developers otherwise. Although in an ideal world it would be possible to persuade some companies to spend the money they would otherwise have spent on buying software licenses on research without placing restrictions on the results of that research, realistically this is unlikely to happen. A lot of research culminates in negative rather than positive results, and industrial sponsorship of research tends to focus on areas with a low risk of failure and a direct industrial application, rather than blue-sky investigations, so its application is restricted. On the other hand, if companies and agencies could be persuaded to fund and participate in academic projects producing Open Source software, the openness of the process may generate significant improvements in the closeness of interaction between funder and fundee.

Open Source knowledge encapsulation

So, Open Source has the potential to accelerate the pace of scientific development by opening the related implementations of the science (the software source code). There is a problem to be overcome, however. The unmodified Open Source approach to model implementation development tends to enhance the peer review of the software engineering in the implementation, but could bypass the scientific peer review. Acceptance of uncontrolled code (by which we mean code implementing changes to the encapsulated science, without associated publication and scientific acceptance) into the implementation will render the implementation scientifically dubious and limit the effectiveness of the scientific process as much as a closed source model release. Note that, depending on the vigilance of the project leaders and the scientific understanding of the contributors, this may not be an actual problem. However, the software so produced will not be used if it is even a potential or perceived problem.

There is perhaps a deeper problem here in the use of software implementations of models described in peer-reviewed academic journals. Without access to the source code of such implementations, that they are faithful to the model must be taken on trust. We do not wish to challenge the integrity of purveyors of closed source model implementations; rather merely to point out that while the bugs that tend to surface in software which is not a model implementation may affect the reliability or adherence to a particular standard of that software, those in implementations of models may affect the very validity of that implementation. Both of these types of bugs can be more effectively found and eliminated in Open Source software, but in the latter case weakly controlled changes are problematic.

This is a matter of quality control. It is an important issue—an obstacle even—in the adoption of Open Source in this area, but it is not insurmountable. The community must develop a means to certify modifications and cross-reference them to the appropriate literature. Open Source projects already limit contributors on the basis of the perceived quality of the software modifications they supply. Here we could limit it to those whose science is trusted or, better, to have a core team of scientifically trusted individuals vet modifications to the code before it is moved into the production version. The technology to achieve this is at least nearly available in source code

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

versioning systems such as the Concurrent Versions System (CVS), but an effective implementation may require the creation of an open source science portal to facilitate this quality control process, along the lines of SourceForge (*http://www.sourceforge.org/*) but more targeted.

THE BUSINESS CASE FOR OPEN SOURCE

The current model

It has long been the case that the most profitable part of a computing business may not be its most visible. O'Reilly (1998) notes that Digital Equipment Corporation earned more of their $15 billion per year revenue from support and services than from selling computers. With the rise to power of Microsoft, we have perhaps lost sight of this, and got used to the idea that the only way to make money from software is by selling it. Indeed, we also forget that by far the majority of software actually written is for use in-house and never sold to anyone, for example in the form of custom business support systems or in embedded controllers for products. When we buy a washing machine, we do not consider it as an IT purchasing decision! Before the rise of the desktop computer, even software that was neither supplied with hardware nor written in-house was generally not bought outright, but came as part of a service agreement with the supplying company in much the same way as hardware.

VisiCorp with its VisiCalc spreadsheet for the Apple II, then Lotus with 123 for the IBM PC, then Microsoft with MSDOS, Windows and a host of 'applications' created the shrinkwrapped software business from nothing. That it is now a multi-billion dollar industry does nothing to alter the fact that selling software is not the same as selling, for example, a wardrobe or a car. The costs of software production are almost entirely in its ongoing development, not its reproduction. While boxed software incurs some cost in the manufacture of packaging, hard-copy documentation and the delivery medium containing the software itself (such as a CD-ROM), these costs are marginal in comparison with maintenance costs, and with distribution methods based on the global Internet the real cost of obtaining a copy of a piece of software tends to zero. At the same time, a company selling packaged software usually generates its entire income from the sale price of the software package. It is difficult to imagine a business model based on a purchase time charge for software (formerly known as 'shrinkwrapped' software, a term that has lost its currency with Internet distribution) that can be sustained without relying on either a constantly expanding user base, or regular sales of upgrades to nearly all current users. The former is clearly limited—the number of new users buying a piece of software will become fewer over time. The latter can be achieved either by persuading the users to upgrade, or forcing them to. This lack of sustainability is almost certainly why Microsoft are pushing towards the 'software as service' model (Microsoft 2000, 2001) in which the user pays an annual or per-use fee, rather than a one-off charge.

In hydroinformatics, this problem is not yet biting. In the case of software not written for sale—and a large portion of hydroinformatics software, as experimental installations testing developing ideas, falls into this category—the development is funded by a particular project grant or sponsor, and since the software is the 'apparatus' for the experiment it is dismantled after use and incurs no ongoing support costs. With software written for sale, such as catchment modelling systems, the pace of development—to some extent in the encapsulated science, but mostly in usability and integration—has been such to date that the users are quite willing to pay the upgrade costs to obtain the features in the newest versions. We might assume, however, that the cracks now showing in the business model of software funded by sale value (one-off charges for boxed products) in the general computing world will show too in hydroinformatics in time, if we do not take evasive action.

Another problem with the closed source model of software production is that, although competition (where it exists) encourages development, it really only encourages gradual change. Revolutionary ideas tend not to emerge from the established players, since their market value is hard to perceive. Price (2000) states that 'commercial software developers will only develop what their customers want.' This statement, apparently made

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

in support of the closed source approach to software production, has at least two hidden assumptions, and in our economic analyses as much as in our modelling we should be careful to make such assumptions explicit. These are that

- what the customer wants is what the customer needs,

and in the longer term that

- the customer has the vision to drive the development of the next generation of software.

In other words, that providing what our customers ask for will lead us along the fastest, or at least a sufficiently fast, development path. These assumptions are then refuted in the same paragraph with the statement, 'Interestingly, as yet, the (commercial) [parenthesis in original] demand for properly integrated modelling of collection-treatment-river impact is still muted.' This is despite the fact that the belief that such integrated modelling will be essential to the effective management of water resources is almost universal in the academic community and among the research-led software houses and consultancies. This statement, then, encapsulates a fundamental problem with the commercial closed source model of development: competition on this model (if there is sufficient competition in the first place) will lead to relatively slow improvement in the available tools, since this improvement is driven by those who see the present tools and project improvements, rather than seeing the leading edge of technology and imagining what is possible. While competition in the software field encourages innovation, it does so only as a side effect of the fight to remain profitable.

Alternative economic structures

The relevance of this to Open Source is that, once we have abandoned our preconceptions about how money is made from software, the fact that the source code is freely distributable (and hence it is nearly impossible to make money by selling software as such) is rendered less of an obstacle to the commercial exploitation of the model. The recent collapse of a number of companies attempting to commercialise Open Source has led some commentators to assert that the model is failing. This is a rather naive argument, since the pain felt in the entire IT sector recently has been considerable and Open Source companies are far from unique in their suffering—witness the number of commercial 'dot-bombs' over the last two years, which were also largely the victims of a flawed business model.

Raymond (1999) includes a study of the economics of Open Source software entitled 'The Magic Cauldron'. Raymond is a free marketeer, and presents his arguments purely from the point of view of economic benefits. It has been explained above that the current economic model of software sale value is unsustainable, and with Open Source the social contract that enables distributed development and generates the key Open Source benefits essentially makes it impossible to make money from direct sales of software. Raymond argues that we can look at two basic models for a solution: use value funding, and *indirect* sale value funding. Both of these can work with Open Source, and perceived problems of funding become less serious in this light.

These two approaches are suited to different types of software. The first, use value funding, is demonstrated by the Apache project, discussed above, and partially by GNU/Linux itself. A lot of funding for development on these projects comes from companies who use them, and essentially *share the cost of development* such that, for a small investment, each company receives a large return. In addition, the investment by these companies in, for example, Apache is protected: Apache cannot be made closed-source, and its continued availability is not conditional on the commercial success of a single company or the continued involvement of the current primary developers (or, indeed, the commercial success of the software itself—a company is unlikely to continue development on a loss-making piece of software). These two aspects are referred to by Raymond (1999) as *cost-sharing* and *risk-spreading*. In the case of Apache, one of the organisations committing resources (programmer hours) to the project is IBM, which suggests that the financial case for collaborating rather than developing yet another competing web server, was very good indeed. Of course, IBM has

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

already been bitten once (losing out to Microsoft at the start of the PC revolution) in failing to move with the times, and is perhaps more keen than before to keep abreast of the changing nature of the computing business.

For hydroinformatics, this approach may well be effective for certain classes of software. In particular, the frameworks or middleware discussed above that will be critical to the development of truly integrated and flexible hydroinformatics tools are candidates for this kind of cost sharing, risk spreading funding model. As our understanding of distributed decision support systems improves and they get to the stage where they can be constructed from ready-made components, the integration framework for these components might be another. Indeed, any company whose primary business is in consultancy rather than software engineering would be well advised to consider this option.

The indirect sale value model of funding is that used by many Open Source companies. Among the most visible of these are the Linux distribution companies such as Red Hat and SuSe. Both take freely available source code, build complete bootable installation CD-ROMs of GNU/Linux and sell them. In many cases, the complete contents of the CDs are also available for free download, but this business model still works, for a number of reasons. In part, it is because many home computer users do not have the network bandwidth to download a complete GNU/Linux distribution, but a major reason is that what these people are buying is the brand, with the implied warranties, time-limited installation support and further support deals available at extra cost. Thus while the same software is available for free, people will choose to pay for it for the sake of ease of use and peace of mind. There are a number of other variations on this model, many of which are catalogued in Raymond (1999).

The various indirect sale value approaches to the funding of software mostly work by using Open Source software to generate a market for something else, and there is clearly potential for this to work in the hydroinformatics field. The most obvious case is to make money from consultancy—while the so-called 'fourth generation' of catchment modelling tools do not demand software engineering skill from their users, their effective use certainly still demands an understanding of the encapsulated science, and that understanding is a valuable commodity in its own right. Further possibilities are to sell commercial extensions to the software (a careful choice of licenses is required to enable this, and it should be noted that most Open Source licenses do not allow modified versions to be closed), hard copy documentation or support services for the software (similar to the consultancy option).

A slight variant of the Red Hat model described above is a potentially valuable one in the case of science-encapsulating software. Earlier the issue of how to guarantee that a piece of Open Source software is an accurate implementation of the science was discussed as a problem to be overcome in the application of Open Source to scientific fields. This could be turned to a business advantage. An Open Source project is, by default, lead by its originators. In the world of hydroinformatics this means projects would generally be led by the same people who at present author and publish software. Thus, using the quality control procedures outlined above, these same people could maintain a reliable (scientifically peer reviewed) version of a model. Ongoing, unreviewed investigations can take place away from the production code, and only be wrapped in as the science is accepted. The management of this reviewed code base is a value adding operation, and where purchasers of software currently buy the software itself, in this Open Source friendly model, they could buy the a *guarantee of reputability* of the software. Digital signatures and certificates technology (Shaw 1999), now quite advanced and recently made legally binding in the UK (HMSO 2000), may prove useful here.

These two models are not mutually exclusive, and the two can be used both within one company and even with the same software. An example of this would be a company that participates in a cost sharing exercise in the development of a framework, while simultaneously providing a custom modifications service to companies without the internal expertise to do so, and selling closed source plug-ins to work with the framework. They might in addition provide CD-ROMs of the framework and standard modules with a simplified installation procedure, and hard copy manuals (although FSF (2001b) argues that Open Source software should be accompanied by open documentation), also at extra cost.

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

In summary, although the social contract form of the licenses under which Open Source software is released inhibit the potential for generating revenue through direct sale value, this model is flawed in any case. Other models, which the software industry is exploring generally, can and have been successfully applied to Open Source projects, and seem to have potential as funding models for Open Source development of hydroinformatics software. Even established software companies in the hydroinformatics field should examine these ideas closely since, as demonstrated by the explosive growth of the Web, the Open Source development of substantial quantities of software can have the result of encouraging growth in a field and need not be a financially damaging proposal.

THE END USER CASE FOR OPEN SOURCE

The discussion so far has centred on the benefits of Open Source to the producers of hydroinformatics software. Although the potential for producing technically better software is of direct relevance to the end user, there are additional benefits too. The black-and-white distinction between producer and end user in closed source is transformed by Open Source into a fuzzy one, through user access to full source code and a direct channel to the producers, allowing users to improve the software, fix bugs and roll the changes back into the official source code repository. It is safe to assume, however, that as hydroinformatics matures a larger proportion of users will not be equipped to modify the source code themselves. For this group, the benefits of open source are mostly secondary, but still significant. The principal specific benefits to several end user groups are itemised in Table 1. In addition to these, there are several more general benefits which apply equally to most end users.

- When software development is no longer funded by the sale value of new versions of a program, *product developments are driven more directly by demand, not marketing*.
- Lower barriers to entry in the software market allow new, innovative products to compete on more equal terms with those better established.
- New modelling developments from research could be more rapidly added to the toolkit of the hydroinformatics software user.
- Users are better protected from orphaning, where a software supplier ceases development of a package which a user is dependent on.

CONCLUSIONS

In the preceding sections of this paper we have outlined the principles and practice of Open Source and Free Software. We have observed that, while the ethical code behind the Free Software movement may suit many, particularly those for whom writing code is a secondary employment or hobby, rather than a primary source of income, not everyone will agree with it. On the other hand, the practical benefits of the Open Source approach, using the same licensing controls as Free Software, have been clearly demonstrated in a number of projects, and it is our belief that the Open Source development model has a lot to offer to hydroinformatics. While it is recognised that not all hydroinformatics software can be open sourced, as current business models would not provide financially for the development of that software, certain aspects of our work, notably the creation of software frameworks, seem to be prime candidates for experimental 'open sourcing'.

We therefore call on the hydroinformatics community to experiment with Open Source models of software production where possible. The economic models presented will not be applicable in all cases but there are many areas of hydroinformatics where these models could prove beneficial to the originators of the software and of enormous benefit to the community, the field and the users of our products. Software suppliers should consider whether releasing some of their code as Open Source and encouraging competitors to share in the costs, risks and rewards of its development would not in the long run be financially beneficial. Software purchasers could push suppliers in this direction in order to reap the benefits of openness discussed above. We believe that Open Source, judiciously applied, could help us fulfil

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

Table 1 | Benefits of software produced on the open source model to specific end user groups in hydroinformatics

End user type	Benefits
Students	• Availability of source code allows accelerated learning and experimentation.
	• Ability to work with same software used in production work.
Academics	• Software can be better integrated into the scientific method, in particular the process of academic peer review.
	• Dissemination of research results enhanced.
	• Easier to build on the work of others without need to reinvent the wheel.
	• Easier to have research products adopted operationally by consultants, agencies and utilities.
Consultants	• Easier to adopt new academic developments.
	• Changes required for specific projects more readily achieved.
	• Sharing the costs and risks of software development allows consultants to concentrate on core buisness.
	• No need to rely on software (and related services) supplied by a direct competitor.
Agencies and utilities	• Operational systems can better evolve to meet changing requirements.
	• Academic developments can be more readily absorbed, leading to better performance.
Software suppliers	• Open sourcing certain types of software could lead to more dynamic, competitive, and altogether larger market for hydroinformatics products (especially services, including custom integration of open and closed source parts).

our future human responsibilities (Abbott 1998), as well as generating previously untapped business opportunities.

At the same time, we must be careful to remember that Open Source development does not guarantee good software engineering. In general, successful Open Source projects start life as a well designed prototype written by one programmer. The architecture of such project seeds is critical to the success of the project—note that Open Source development requires good modularity in order to evade the problems predicted by Brooks' Law. We must also consider which aspects of hydroinformatics software are best suited to Open Source, and choose carefully those where the process *adds* value for, as the dot com failures of recent times indicate, it is hard to build a sustainable business model on its *removal*.

Since we hope this paper will trigger some debate, and we are sure there will be those who disagree strongly with our point of view, we feel that there would be great benefit in using some of the Open Source Internet infrastructure to facilitate a discussion. At *http://www.cen.bris.ac.uk/civil/research/wemrc/opensource/* we have set up a web forum. You are all encouraged to join in.

ACKNOWLEDGEMENTS

The work leading to this paper was funded by EPSRC studentship number 98314947. We would also like to thank Jim Hall of the University of Bristol for taking the time to read and comment on a draft of the paper.

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest

REFERENCES

Abbott, M. B. 1991 *Hydroinformatics: Information Technology in the aquatic environment*. Avebury Technical, Aldershot, UK.

Abbott, M. B. 1998 Future business opportunities: future human responsibilities. In: *Hydroinformatics '98* (ed. Babovic, V. and Larsen, L. C.), pp. 1171–1175. Balkema, Rotterdam.

Abbott, M. B. & Jonoski, A. 1998 Promoting collaborative decision-making through electronic networking. In: *Hydroinformatics '98* (ed. Babovic, V. and Larsen, L. C.), pp. 911–918. Balkema, Rotterdam.

Asklund, A. & Bendix, L. 2001 Configuration management for open source software. In: *1st Workshop on Open Source Software, ICSE 2001*. http://opensource.ucc.ie/icse2001/asklundbendix.pdf (accessed July 2001).

Babovic, V. 2000 Web clips for hydroinformatics. *J. Hydroinformatics* **2**(4).

Brooks, F. 1995 *The Mythical Man-month: Essays on Software Engineering, Anniversary Edition*. Addison-Wesley, Reading, MA.

Cunge, J. A. and Erlich, M. 1999 Hydroinformatics in 1999: what is to be done? *J. Hydroinformatics* **1**(1), 21–31.

Dafermos, G. N. 2001 Management and virtual decentralised networks: the Linux project. *First Monday* **6**(11). http://www.firstmonday.dk/issues/issue6_11/dafermos/ (accessed November 2001).

Dyson, E. 1998 Open mind, open source. In: *Release 1.0* (ed. Dyson, E.), pp. EDventure Holdings Inc., New York pp 1–2.

FSF 1991 *GNU General Public License*. Free Software Foundation, Boston, MA. http://www.fsf.org/licenses/gpl.html (accessed July 2001).

FSF 2001a *The Free Software Definition*. Free Software Foundation, Boston, MA. http://www.fsf.org/philosophy/free-sw.html (accessed July 2001).

FSF 2001b *Free Software and Free Manuals*. Free Software Foundation, Boston, MA. http://www.fsf.org/philosophy/free-doc.html (accessed November 2001).

Himanen, P. 2001 *The Hacker Ethic and the Spirit of the Information Age*. Secker & Warburg, London.

HMSO 2000 *Electronic Communications Act 2000*. http://www.legislation.hmso.gov.uk/acts/acts2000/20000007.htm (accessed August 2001).

Jackson, T. P. 1999 Findings of Fact, United States of America v. Microsoft Corporation, Civil Action Nos. 98-1232 and 98-1233, December. http://usvms.gpo.gov/findings_index.html (accessed November 2001).

Khatibi, R., Haywood, J., Akhondi-Asl, A., Whitlow, C., Wade, P. & Harrison, T. 2001 Open architecture in flood forecasting systems. In: *River Basin Management* (ed. Falconer, R. A. & Blain, W. R.), pp. 149–160. WIT Press, Southampton.

Microsoft 2000 Microsoft announces new subscription offering for 'Office 10'. http://www.microsoft.com/PressPass (accessed August 2001).

Microsoft 2001 Microsoft simplifies, enhances volume licensing programs. http://www.microsoft.com/PressPass (accessed August 2001).

Netcraft 2001 Netcraft web server survey, July. http://www.netcraft.com/survey/ (accessed July 2001).

OSI 2001 The Open Source definition, version 1.9. http://www.opensource.org/docs/definition.html (accessed November 2001).

O'Reilly, T. 1998 The open-source revolution. In: *Release 1.0* (ed. Dyson, E.), p. 24. EDventure Holdings Inc., New York.

Price, R. K. 2000 Hydroinformatics and urban drainage. *J. Hydroinformatics* **2**(2), April.

Raymond, E. S. 1999 *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Sebastopol, CA. http://www.oreilly.com/catalog/cathbazpaper/ (accessed July 2001).

Schellnhuber, J. & Hulme, M. 2001 *The Tyndall Centre Research Strategy*. The Tyndall Centre for Climate Change Research. October. http://www.tyndall.ac.uk/research/strategy.shtml (accessed November 2001).

Schmidt, D. C. & Porter, A. 2001 Leveraging open-source communities to improve the quality & performance of open-source software. In: *1st Workshop on Open Source Software, ICSE 2001*. http://opensource.ucc.ie/icse2001/schmidt.pdf (accessed July 2001).

Shaw, S. 1999 Overview of watermarks, fingerprints, and digital signatures. JTAP. http://www.jtap.ac.uk/reports/htm/jtap-034.html (accessed August 2001).

Spangler, A. 2001 Open source development: a suitable method to introduce a standardized communication protocol? In: *1st Workshop on Open Source Software, ICSE 2001*. http://opensource.ucc.ie/icse2001/spangler.pdf (accessed July 2001).

Stallman, R. 1998 The GNU project. http://www.fsf.org/gnu/the-gnu-project.html (accessed July 2001).

ten Cate, H. H., Salden, R., Lin, H. X. & Mynett, A. E. 1998 A case study on integrating software packages. In: *Hydroinformatics '98* (ed. Babovic, V. and Larsen, L. C.), pp. 457–464. Balkema, Rotterdam.

Tomicic, B. & Yde, L. 1998 Integrated software for an integrated management and planning of urban drainage and wastewater systems. In: *Hydroinformatics '98* (ed. Babovic, V. & Larsen, L. C.), pp. 465–471. Balkema, Rotterdam.

Yan, H., Solomatine, D. P., Velickov, S. & Abbott, M. B. 1999 Distributed environmental impact assessment using Internet. *J. Hydroinformatics* **1**(1), July.

Yee, D. 1999 Development, ethical trading, and free software. http://danny.oz.au/freedom/ip/aidfs.html (accessed August 2001).

Downloaded from https://iwaponline.com/jh/article-pdf/4/4/219/392502/219.pdf
by guest