Vision statement on open architecture for hydraulic modelling software tools

Rahman Khatibi, Dave Jackson, John Curtin, Chris Whitlow, Adri Verwey and Paul Samuels

ABSTRACT

This paper identifies the philosophy of open architecture as a feasible vision capable of transforming modelling software packages into living products. This vision, invoked within the specific context of software production in the field of flood forecasting within the Environment Agency, promotes the emerging requirements and consensus of users, academics and software producers. In the past, the philosophy of closed architecture dominated the use, development resources and investment in modelling systems by producers and users. As closed architecture encourages the development of monolithic software products with limited scope for innovation by third parties, investments often do not return the value of their full potential. A consensus is emerging that this is no longer tenable. The time is right: for the producers of hydraulic and hydrologic software tools to move from the culture of ‘doing things better’ to ‘doing things better and doing better things’; for users to design their own systems through assembling off-the-shelf software products; and for academics to have a less restrictive environment in which to innovate. The consensus view is rendered viable in a partnering culture undoing many barriers and restructuring many concepts. The paper postulates that software development is a paradigm and shifts through the forming, proliferating, norming and performing stages. This postulate is substantiated by citing evidence for the following associations:

- The forming stage is associated with the development of early computer programs.
- The proliferating stage is associated with closed architecture.
- The norming stage is associated with open architecture to create interoperability.
- The performing stage is associated with open source to freely share and improve source codes.

Key words | software architecture, paradigm shifts, off-the-shelf product, living product, interconnection, stakeholder

INTRODUCTION

The Customer Charter of the Environment Agency for England and Wales has identified the flood forecasting and warning service as a life protecting measure, giving it the highest corporate priority. A range of drivers are promoting best practice in this service, most notably the Bye Report (Bye & Horner 1998) which criticised the disparity in the flood forecasting capabilities among the Environment Agency’s regions. While an Agency-wide best practice is still being formed, one region with a strong track record on flood forecasting is currently procuring the replacement of its Flood Forecasting System (FFS). The needs of this region are driven by the requirement to replace both an FFS and an ageing hardware system, which will cease to have software support. The disparity in the capabilities and the possibility of being burdened with an obsolete system are risks posed against business critical flood forecasting and warning services. The point to be pressed is that, if this risk is not mitigated, the integrity of
this operationally critical process will be undermined. This risk has invoked the determination among the flood forecasting team in the Environment Agency to mitigate it. Other organisations are also vulnerable to similar risks. This paper focuses on solutions to this risk.

This paper uses systems approaches to explain the current status and the future of software applications in modelling hydraulic/hydrologic problems and further employs the concept of ‘paradigm’, as revisited by Khatibi (2005). Kuhn (1962) presented the doctrine of paradigm, where a paradigm is a framework in science and technology. However, Khatibi postulates that a paradigm is formed following the law of natural selection under spontaneous pre-paradigm conditions and shifts through the forming, proliferating, norming and performing stages, each stage associated with appropriate rules. This paper presents software development as a ‘paradigm’, which emerged as a product of Information Technology (IT) in the 1950s and identifies the various rules at each stage to the paradigm of software development.

It is postulated that software architecture is the building block for the shifts from one stage of the paradigm of software development to another, where architecture is defined as ‘the conceptual structure and logical organisation of a computer or computer-based system’. Arguably, the current stage in the development of hydraulic software capabilities has persistently remained in the proliferating stage but the overall context is as follows:

- In the pre-paradigm period, empirical hydraulics reached the limits dictated by manual calculations.
- In the forming stage, computational speeds of IT-based tools offered a selective advantage for the development of computer programs and produced a major impact on science and technology, including hydraulics.
- In the proliferating stage, computer programs are transformed into software products under the umbrella of closed architecture, but it is not feasible to ‘plug in’ innovative off-the-shelf components directly to existing systems.
- In the norming stage, the requirement for open architecture prevails to provide interoperability in software systems, where third-party software engineers can attach off-the-shelf components from different producers into user-designed systems.
- In the performing stage, the open source movement can play a pivotal role in sharing freely and improving software source codes.

The authors promote open architecture, the realisation of which is believed to be feasible and inevitable. Closed and open architectures have strengths and weaknesses and therefore their selective advantages depend on the balance between the opportunities they are capable of creating and the users’ demand for such opportunities. Although the paper considers architectural issues related to computational hydraulics, its specific focus is on flood forecasting. The paper reflects on the past, present and future of the paradigm of software development for hydraulic modelling capabilities. While the hydraulic modelling community awaits the emergence of open architecture, this paper aims to answer the following questions:

- Why has such an architecture not emerged spontaneously?
- Is emergence fostered (top-down initiatives) or spontaneous (bottom-up initiatives)?
- Are initiatives of developing open architecture problem-specific (endemic/pandemic) or do they spread laterally (epidemic)?

**Historic context**

Since the 18th century, diversification of stakeholders in the emerging functions of rivers was an impetus to the development of open channel hydraulics. Although theoretical hydraulics was developed during the 19th and early 20th centuries, computational difficulties dictated the limits of its application. The advent of computers created a further impetus since the late 1950s, giving rise to the emergence of software tools. Those that would have been regarded as a software product in the early days are only of historic value for today’s software users. This signifies waves of changes and three overviews are outlined in Table 1 for capturing the current status of software development. The table presents:


• Khatibi’s (2001) presentation of paradigm shifts in the development of open-channel simulation capabilities.

Although a descriptive account of software generations is informative, it does not fully reveal the generic stages of development and is not normally an explanatory tool. For instance, the forces in triggering apparently sequential generations in modelling are not readily identifiable in Abbott’s account. It is true that Abbott regards the fourth generation as a veritable revolution, with extensive benefits of hydraulics to the benefit of the society, and presents the term ‘hydroinformatics’ to hallmark his expected fifth generation modelling systems. Inherent in Abbott’s philosophical doctrine is the fusion of ‘hard’ systems in the form of computational hydraulics and ‘soft’ systems in the form of emerging artificial intelligence. However, it is not clear whether this fusion is spontaneous or through a period of restructuring by a foresight for consolidation and improvements. Similarly, Brooks’ categorisation of developmental stages in software development does not offer a generic tool for creating transparency between the developmental stages in software developments and other paradigms. Arguably, Brooks’ categorisation is closer to postulating software development as a paradigm, as presented in this paper.

The aim of this paper and the above historic reviews is to unravel the role of architecture in software development. The definition of architecture presented above (the conceptual structure and logical organisation of a computer or computer-based system) should be complementary to Brooks’ definition that architecture is ‘the complete and detailed specification of the user interfaces’. However, user interfaces become an issue in open architecture but not normally in closed architecture. Currently, worldwide software architecture in hydraulic modelling is based on the capability of proprietary packages (developed for in-house use or sale) or bespoke packages (normally proprietary packages with additional user-specified enhancements). As long as there is one package which meets the needs of one organisation, the software architecture is not a problem. This is often not the case, as there are two problems to be solved:

(i) Re-organisation, especially in the public sector, can lead to the new organisation inheriting several packages, each with its own culture from its predecessors, such as the Environment Agency which inherited several river modelling and flood forecasting packages.

(ii) The various software packages are not compatible and therefore user-designed systems are not possible through client selection of the features from off-the-shelf software components.

**THE PARADIGM OF MODELLING SOFTWARE AND ITS SUBSEQUENT SHIFTS**

There is no denying that existing software products are versatile tools, which have made a remarkable contribution to science in general and to hydraulics in particular. However, there are also problems stemming from their architecture. Notwithstanding this, there is a remarkable willingness to address these problems, e.g. the HarmonIT project (http://www.harmonit.org/)—a fifth Framework Program (FP5) project within the R&D program of the European Community.

In order to understand inherent problems, the flexibility of existing software capabilities needs to be critically appraised. Figure 1 illustrates the framework for such an appraisal, which is a two-dimensional examination of a software tool for its capability for (i) admitting off-the-shelf modules at a systems level (with three possibilities of open, quasi-open and closed) and (ii) the ability to undertake heterogeneous, quasi-heterogeneous and homogeneous modelling. Heterogeneous modelling refers to the ability of using datasets from third-party software products. In quasi-heterogeneous modelling, the choice of modelling datasets is restricted to those from a proprietary sub-system but in homogeneous modelling the choice is limited to only using native model datasets. Closed architecture software tools
Table 1 | Review of the various schools of thought on explaining the development of hydraulic software tools

<table>
<thead>
<tr>
<th>Attributes of the classification of software generations in hydraulics by Abbott (1991). Note: these generations are similar to those illustrated by Long (1994) in relation to the development of computers and information technology.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1st generation</strong></td>
</tr>
<tr>
<td><strong>2nd generation</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>3rd generation</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>4th generation</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>5th generation</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes of Sayers’ (1941) Classification of Stages of Software Development</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stage 1</strong></td>
</tr>
<tr>
<td><strong>Stage 2</strong></td>
</tr>
<tr>
<td><strong>Stage 3</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes of Brook’s (1995) Classification of Stages of Software Development</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stage 1</strong></td>
</tr>
<tr>
<td><strong>Stage 2</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Stage 3</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes of paradigmatic stages in open-channel simulation capabilities—Khatibi (2001)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pre-paradigm</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Forming</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
The pre-paradigm period and the forming stage

The forming stage of the paradigm of software development has been spontaneous, although it drew on an already existing culture of computation and formalised empirical design procedures. Prior to the IT revolution, ‘human computers’ were employed to work out calculation tables in as early as the 19th century, who were simply good at arithmetic operations without necessarily delivering customised solutions. Figure 3 illustrates the main features at each stage of the paradigm of software development.

Table 1  |  Continued

<table>
<thead>
<tr>
<th>Paradigm Stage</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proliferating</td>
<td>- Software tools emerged with selective advantages of being general-purpose, data-steered, nodular, robust, having front-end and back-end facilities and later with GUI, with error traps</td>
</tr>
<tr>
<td></td>
<td>- The breaking up of the profession into software producers, modellers and innovators</td>
</tr>
<tr>
<td>Norming</td>
<td>- Modelling served as proactive tools to investigate system performance in diverse</td>
</tr>
<tr>
<td></td>
<td>- Ease of use of modelling tools and risks of overlooking inherent assumptions/simplifications</td>
</tr>
<tr>
<td></td>
<td>- The requirements for the partnership among software producers, modellers and innovators</td>
</tr>
<tr>
<td>Performing</td>
<td>- The need for tools on sustainable development</td>
</tr>
<tr>
<td></td>
<td>- Integration of modelling towards total modelling capabilities</td>
</tr>
<tr>
<td></td>
<td>- Need for conscientious practices to treat uncertainties, refine models and formalise practice</td>
</tr>
</tbody>
</table>

Various architectures as defined by Khatibi et al. (2001)

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed architecture</td>
<td>This is inherent in many worldwide hydraulic modelling practices, where the various components within one package are normally specific to its native software system and cannot be transferred into other packages without some intervention by their producers.</td>
</tr>
<tr>
<td>Quasi-open architecture</td>
<td>Some features in a number of software packages potentially allow for a limited incorporation of off-the-shelf components, although there are not many such components marketed.</td>
</tr>
<tr>
<td>Open architecture</td>
<td>A fully open architecture is yet to be realised. However, the concept is feasible in terms of users designing their required systems and then assembling together a variety of off-the-shelf components from different producers without the need for any form of re-programming, this is sometimes referred to as the ‘plug and play’ concept.</td>
</tr>
</tbody>
</table>
being good at mathematics. These human computers were also employed in emerging large organisations to look after their accounts. This was also paralleled with an emerging design culture in the various disciplines of engineering, including civil and water engineering. Inevitably, there were limitations on their contributions and therefore other solutions were developed, such as developing various approximate methods and graphic solutions in water engineering. Through approximate and graphic methods, human computers or formalised design procedures have created a professional culture toward the forming stage of the paradigm of software development.

The development of early computer codes was not easy and needed dedication, but offered selective advantages owing to their speed and the ability to repeat the same procedure with different sets of input data. These selective advantages were not immediately obvious in the early endeavours of writing computer programs. Different generations of programming languages emerged, where the early generations employed text-based command-driven interactions. Dedicated experts often wrote modelling programs mastering the full lifecycle of their code and model development, implementation, running and results processing.

In addition, the codes were project-specific, and not every institution could afford them. The programs were often developed in reaction to important or fundamental problems. These programs marked the formation of the paradigm of software developments. Figure 4(a) illustrates a retrospective view of this stage with a seemingly monolithic outlook for innovation, program writing and using the programs.

The proliferating stage—the emergence of closed architecture

The paradigm of software development, formed by the IT revolution, has its seed in the early programs, which had to be modified from one application to another. The key selective advantages of a computer-based modelling over manual calculations were speed and the ability to repeat the same calculations with different input data. Since the late 1950s and up to 1990, generations of IT solutions proliferated together with rapid development activities. Universities, research and professional organisations were preoccupied with transforming many approximate hydraulic methods of solutions into working tools. By modern standards those tools were often rudimentary, project-specific and required dedication. Equally, they are seen retrospectively as user-hostile tools, where troubleshooting of runtime problems was an important preoccupation. Although early computational hydraulic programs of the 1970s served as tools, the ability for repetitive running was often restrictive, as the data was often

Figure 1 | Appraisal of openness of existing products.
Selective advantage for a combination of certain building blocks

Formation of a paradigm

Proliferating Stage: Selections of many workable arrangement of the same components And only one-way flow of information

Forming Stage:
A selection of a workable arrangement of components

Proliferating Stage: Selections of many workable arrangement of the same components And only one-way flow of information

Norming Stage: Two-way flow of information and the emergence of hierarchies and control and regulation

Performing Stage: 2-way flow of information in all the components to align the components to maintain steady state & to adapt

Figure 2 | Paradigmatic stages.

Figure 3 | The paradigm of software development and its subsequent shifts.

<table>
<thead>
<tr>
<th>Paradigm Shifts</th>
<th>Generic Features</th>
<th>Software Development Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-paradigm periods</td>
<td>Fertile ground for many possibilities each with little selective advantage</td>
<td>Human-computers and empirical design procedures a need for speed and tools</td>
</tr>
<tr>
<td>Formation of a paradigm</td>
<td>Selective advantage for a combination of certain building blocks</td>
<td>Early project-specific programmes with a potential for for speed and repetition</td>
</tr>
<tr>
<td>Proliferation Stage</td>
<td>Niche opportunities and changing fortunes - opportunism</td>
<td>Closed-architecture Versatile tools- 1-way flow of info Users &amp; Developers Innovators</td>
</tr>
<tr>
<td>Norming Stage</td>
<td>Prevalence of a foresight interconnectedness, intelligence and strategy</td>
<td>Open Architecture 2-way flow of information delivering living products</td>
</tr>
<tr>
<td>Performing Stage</td>
<td>Customised solutions capable of living and growing</td>
<td>Open architecture &amp; source- free flow of information living and growing products</td>
</tr>
</tbody>
</table>
embedded in the program and even the program had to be amended from one project to another.

By the early 1980s some of these programs gained commercial value and there were incentives to make them data-steered and general-purpose. In the early 1990s front-end editors and back-end graphics were also provided. Equally some institutions released similar modelling software products, nominally free of charge and often referred to as ‘freeware’. An outcome of these software tools was the ability to repeat, speed and convenience for customised solutions. A tool may be seen as a catalyst, facilitating the production without being exhausted and not altering it from one application to another. These attributes are embodied into software tools, defined as programs that direct the activities of the computer system by changing input data but not the source code. At this stage, the ties among the innovators, software producers and model developers/users was redefined, by splitting into different skill bases through a one-way flow of information as illustrated in Figure 4(b).

Existing hydraulic modelling (and flood forecasting) systems are largely proprietary or bespoke software packages with some freeware but all follow the closed architecture philosophy. This architecture emerged spontaneously, responding to a niche market for tools to treat a host of practical problems. However, this architecture may pose three types of problems:

(i) Organisations such as the Environment Agency use software tools as operational tools and therefore must manage the risk of their software capabilities becoming obsolete, which can happen if the producers cease to support their products.

(ii) While consultants developing models are exposed to the same problems as model users such as the Environment Agency, in addition they have to maintain a store of proprietary software products.

(iii) Each of these products often contain many similar facilities that are repetitive but not transportable from one software environment to another, e.g. GIS or editor facilities within different packages. This incurs substantial costs that can be trimmed in a user-designed system. However, the other side of the coin is that competition among software vendors stimulates the development of creative and efficient support tools.

Software producers frequently transfer the same piece of software from one package to another and sometimes, in
order to maintain their competitive edge, they have to produce a product from scratch that other producers specialise in. One explanation of this problem is that source codes written for a particular task may inherently be modular but under the culture of closed architecture the modularity structure is not transparent among different producers, leading to the following problems:

- Software products are normally monolithic in the sense that they prescribe a set of governing equations for each set of boundary, locally distributed, spatially distributed and control processes without being interoperable in precluding the usage of non-native modules.
- Model datasets cannot be transferred directly from one to another proprietary software product, i.e. model datasets are only intelligible in their native systems.

The absence of published interfaces is the root of the problem. The testing of new products then becomes a formidable task, as the number of combinations in testing is often very large. Thus, in a closed architecture environment, (i) interoperability among different software systems is not feasible as their modularity is discrete and lacks transparency; (ii) as such, they are in essence a top-down approach and inherit many barriers.

A comparison of commercial software packages with freeware modelling products reveals an important fact. The availability of freeware did not redistribute market forces and did not turn the tide against commercial interests but it emerged as another choice in the market. Since both commercial software and freeware capabilities followed the closed architecture philosophy, they shared the same generic strengths and weaknesses. Thus, other than costs, modelling freeware did not offer any selective advantage or any breakthrough.

The norming stage—the call towards open architecture

Many of the problems discussed above can be solved through open architecture and this characterises the norming stage of the paradigm of software development. There is no novelty in open architecture and it is accepted as a norm in other fields of IT. For instance, Holtom (1994) states that ‘A selection of off-the-shelf hardware and software exists, which can be combined to make powerful, low-cost SCADA systems. Municipalities need not be restricted by proprietary systems when open-architecture systems are achievable at lower overall cost.’ Open architecture is also feasible in hydraulic and hydrological modelling and its first building blocks have already prevailed through highly modularised software tools even though they are not offered as off-the-shelf products.

The actual procedure for the realisation of this architecture will be explained in this paper but in summary it comprises an off-the-shelf ‘open shell’ with published interfaces and a whole range of modules attachable to the open shell to process data, run models, post-process results and communicate with external systems. Open architecture is one step towards transforming software products with ‘dead-end’ prospects to living products. The Environment Agency has specified open architecture through an open shell to underpin its flood forecasting systems and an effective step towards the realisation of user-designed systems.

The realisation of user-designed systems through open architecture is overdue and it appears that a spark is needed to set off the process. This may be set off from the consensus among operational users, modellers, software producers and innovators. This ideal will be underpinned by the open shell and published interfaces, on the one hand, and the formation of a critical mass subscribing to the ideals of open architecture and transforming their software products into modules through published interfaces on the other hand. It is evident that the norming stage, as outlined here, depends on a two-way flow of information among innovators, software producers and end users, as illustrated in Figure 4(c).

The performing stage—the future of software development

‘Open source’, as a vehicle towards the performing stage, is defined as the software source code freely shared and
improved. Harvey & Han (2002) hold that open source results in potentially significant benefits by encouraging widespread interoperability and rapid development. They contrast open source with closed or proprietary source, where software is written for sale or for in-house use. Quoting Raymond (1999), they inform us cautiously that as much as 95% of codes are written in-house, which is almost always treated by coding organisations as being a primary asset. Open source is a natural outgrowth of the Free Software movement through the following milestones:

- Richard M. Stallman (usually referred to as RMS) started the process in 1971.
- RMS formed the Free Software Foundation (FSF) to administer funds and resources for this development in 1985.
- A large quantity of free software, referred to as General Public License (GNU), became available for Unix-compatible operating systems but these needed a proprietary core to run.
- Linus Torvalds wrote and released the source code for a Unix-compatible kernel (called Linux) under the FSF General Public License (see FSF 1991). This finally freed GNU from proprietary kernels and represented a viable alternative to Windows.
- RMS developed a set of values (Stallman 1998) stating that closed source infringed on the user’s rights and set about writing a complete operating system, providing a platform for such rights.
- The open source label came out of a strategy session held on 3 February 1998 in Palo Alto in relation to the release of the Netscape browser source code in the hope of attracting external programming talent.

Harvey & Han (2002) argue that Torvalds harnessed a massive talent pool willing to work for mutual benefit, rather than immediate financial gain, 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. The fundamental principles of open source are that, when a piece of software is released as open source, it includes the full source code and permission for any user to modify it. There is often a moral obligation to return the modified versions back to the FSF for distribution, although this is not a legal requirement. The FSF may or may not charge for the new issue. These are considerable freedoms when compared with the typical proprietary software, but they come with a moral obligation, often under a licence agreement, that any changes made to the software must be given back to the community. They further argue that the open source development process is not anarchic but analogous to publishing peer-reviewed scientific journals. Arguably, open source software products may be transformed into organically growing products only through institutionally progressed arrangements.

This paper emphasises the roles of FSF, open architecture and published interfaces in an effective proliferation of open source. Similar to modelling freeware, which did not replace commercial software, choices in open source can proliferate but cannot be transformed into an organically growing environment without the FSF, open architecture and published interfaces. It is noted that the realisation of FSF, open architecture and published interfaces encourages a multi-directional flow of information among innovators, software producers and model developers/model users, as illustrated in Figure 4(d). In this empowered flow of information the spirit of partnership is of paramount importance and this will be discussed further in this paper.

Overview

The shifts from the formation of a paradigm to its performing stage are not automatic and that from the forming to performing stage gets harder and harder. Although the building blocks (components) of a paradigm at the formation and proliferation stages bind to one another the components are treated one-at-a-time at the expense of losing their inter-component synergies. Khatibi (2003) argues that the forming and proliferating stages of a paradigm are driven by the law of natural selection, which is a blind architect capable of creating niche opportunities and changing fortunes. On both accounts—treating the components one at a time and following the law of natural
selection—paradigms at these stages are opportunistic, meaning that there is a lack of foresight towards future directions.

The shift to the norming stage is associated characteristically with interconnecting the components with one another for a more pronounced two-way flow of information, with the immediate outcomes of gaining an insight into the inter-component synergies and of obtaining a strategic overview. The outcomes of the interconnectedness are essential for the emergence of a degree of intelligence to minimise opportunism and develop a strategy for complying with multiple objectives. The necessity to shift from the proliferating to the norming stage can often be in sight from the early stages but obstacles need to be removed before realising the shift. For instance, the performing stage of the paradigm of modelling software is visible now even though this paradigm is at its proliferating stage.

The norming and performing stages are conscious processes to identify the deficiencies and pitfalls and treat them. The performing stage often opens up the paradigm to customised or user-designed solutions. The ability to customise a solution needs a greater knowledge and many well-tested technical measures. In this culture many barriers are undone and many concepts are restructured. There are significant barriers to the realisation of the norming and performing stages of the paradigm of modelling software:

- For an effective culture of open source, the barriers among users, producers and innovators need to be removed; an institutional arrangement is required to ensure the integrity of the software tools and an intelligent community has to be formed for the organic growth of the products in their common interest.

BUILDING BLOCKS OF OPEN ARCHITECTURE IN MODELLING

The shift from closed to open architecture is not a trivial task. The capability of modularisation through published interfaces is a minimum requirement but the emergence of an effective open architecture depends on the development of hierarchies and interfaces among modelling systems across different disciplines. These requirements call for the consolidation of science but its full presentation is outside the scope of this paper. This section presents the architectural building blocks of modularisation, interfaces, publication of interfaces and hierarchies. An overview of the architecture is illustrated in Figure 5, breaking down computationally-based systems (e.g. for flood forecasting) at an architectural level into ‘open shell’, input/output modules, utility modules and computational/forecasting modules and storage spaces for datafiles and datasets. This paper is not concerned with the layers below the open shell but the open shell and the layers above it are detailed below.

Open shell

The open shell may be defined in terms of a modular software tool capable of:

- incorporating, operating or otherwise using the computational modules;
- facilitating the processing of raw or previously processed data through a range of input/output software modules, such as editors;
- facilitating utility modules such as inundation mapping.

Open architecture is viable through published interfaces. Although currently there is a lack of such interfaces, this is just an obstacle, not a problem. There is already a procedure to remove this obstacle, as discussed later in this paper.

Although open architecture is viable, innovative applications still have to be developed containing many complex interactions between the various computational modules—the organisation of such modules is usually taken care of automatically in closed architecture environments.

The environment for published interfaces also depends on stable and flexible institutions and partnering among the stakeholders.

- Open architecture is viable through published interfaces. Although currently there is a lack of such interfaces, this is just an obstacle, not a problem. There is already a procedure to remove this obstacle, as discussed later in this paper.
- Although open architecture is viable, innovative applications still have to be developed containing many complex interactions between the various computational modules—the organisation of such modules is usually taken care of automatically in closed architecture environments.
- The environment for published interfaces also depends on stable and flexible institutions and partnering among the stakeholders.

• For an effective culture of open source, the barriers among users, producers and innovators need to be removed; an institutional arrangement is required to ensure the integrity of the software tools and an intelligent community has to be formed for the organic growth of the products in their common interest.

BUILDING BLOCKS OF OPEN ARCHITECTURE IN MODELLING

The shift from closed to open architecture is not a trivial task. The capability of modularisation through published interfaces is a minimum requirement but the emergence of an effective open architecture depends on the development of hierarchies and interfaces among modelling systems across different disciplines. These requirements call for the consolidation of science but its full presentation is outside the scope of this paper. This section presents the architectural building blocks of modularisation, interfaces, publication of interfaces and hierarchies. An overview of the architecture is illustrated in Figure 5, breaking down computationally-based systems (e.g. for flood forecasting) at an architectural level into ‘open shell’, input/output modules, utility modules and computational/forecasting modules and storage spaces for datafiles and datasets. This paper is not concerned with the layers below the open shell but the open shell and the layers above it are detailed below.

Open shell

The open shell may be defined in terms of a modular software tool capable of:

- incorporating, operating or otherwise using the computational modules;
- facilitating the processing of raw or previously processed data through a range of input/output software modules, such as editors;
- facilitating utility modules such as inundation mapping.
Software details of the open shell are design issues and outside the scope of this paper. If the open shell is not available as an off-the-shelf product, its full potentials will not be realised. The open shell is particularly viable through published interfaces.

**Modularisation of computational facilities**

In an open architecture environment, modularisation must penetrate every part of the system and this includes computational facilities. This is a formidable problem, as there are numerous alternative equations to describe each of the boundary, locally distributed and spatially distributed processes. It is not attractive to transform each possible set of equations into a module, attachable to the open shell. Against this background are hundreds of possible interfaces to be built into the open shell but some sort of housekeeping order is essential, for which categorisation can serve as a generic tool. For instance, there are numerous conceptual modelling techniques, each with an input/output requirement. Publishing one single interface to collectively represent the diversity of these techniques is a step towards the formalisation of their inputs/outputs and a significant contribution towards best practice in software development and professional practices but this step is yet to be taken.

Implementations of interfaces involve the design of agreed schemas and their associated data exchanges. Exchanges of data may be carried out using extendible markup language (XML) but the design of schemas are not discussed here. A method has been developed in the Environment Agency to categorise modelling techniques for flood forecasting and these are outlined by Khatibi (2002) and Khatibi et al. (2002a, 2002b) and depicted in Figure 6. Each category is associated with a number of techniques, which can conveniently be used to develop one interface per each category and each category of the techniques is called a computational ‘module’, as
discussed by Khatibi et al. (2002b). Figure 6 depicts the categories of modelling techniques discussed above and for hydraulic/hydrologic models of open channels they are: rules of thumb, empirical approaches, blackbox models, conceptual models, hydrological routing models, kinematic and hydrodynamic routing. Computational hydraulics for coastal forecasting, groundwater flow prediction models or even empirical approaches for snowmelt modelling are possible. Thus, categorisation may be used as a science-based tool for designing schemas.

Published interfaces

The legacy practices on input/output formats of software modules are ad hoc and this needs urgent attention. The route from ad hoc to standardised practices is often long and difficult. The first step in the trajectory towards standardisation in software development is published interfaces. The role of published interfaces is fundamental, as the shells from different producers become interchangeable by being designed to fit interfaces. By the same token, different modules by different producers for a particular interface become interchangeable. The route from ad hoc to standard practices is related to the process of consolidating and institutionalising consensus among stakeholders.

Standardisation of a practice or a product is not spontaneous and is the outcome of waves of authoritative research findings or best practice. It is developed, maintained and enforced often through institutional arrangements with a legal requirement to comply with the standardised products. It is in the common interest to
uphold open standards and the benefits are so important that similar measures are often developed even if standardisation does not seem imminent. It is a good sign that a need is perceived to move towards published interfaces of computational modules, as this signifies that there is a community concerned with the problem and sooner or later a movement will be formed. Developments towards standardisation may progress through the following steps:

- the emergence of interacting user and producer communities;
- publishing interfaces or guidelines to formalise the practice/product;
- the emergence of champions and consensus builders to formalise the practice through manuals or codes of practice;
- designated authorities or standardisation institutes, taking note of the emerging good practices and standardising them.

This paper argues that there is an intelligent community of users, producers and innovators in hydraulic modelling practices and all are awaiting the realisation of user-designed systems. It is in the common interest to invoke the formation of a movement towards the culture of user-designed systems. The movement can be promoted by learned societies, scientific journals and conferences. Organisations such as the Environment Agency can champion this cause, e.g. by supporting research committed to publishing their software interfaces.

### Combining forecasting modules

User-designed systems are highly desirable, as innovative modular products become system-independent and the systems become ‘living products’. The delivery of the open shell is one step towards assembling a range of computational modules to carry out flood forecasting but communication among the assembled computational modules is another obstacle to be overcome. A number of possible approaches are outlined below.

(i) If the computational system comprises the open shell onto which various computational modules are attached at published interfaces and these modules produce output to published interfaces, off-the-shelf products can be developed to run these modules in a certain sequence, creating a sharing medium among their inputs and outputs. Such off-the-shelf products will be referred to as adapters. Adapters are external managers of the forecasting modules. Although they are technically feasible, they do not exist at present or, if they exist, they are yet to penetrate the market. An example of this approach in the context of flood defence design and planning was the EUROTAS prototype system shell developed under EC research in the Fourth Framework Program (see the project report compiled by Samuels (2001)).

(ii) If the computational system comprises the open shell onto which various computational modules are attached at published interfaces and these modules produce outputs to published interfaces, off-the-shelf products can be developed to run these modules through an internal assemblage. Such off-the-shelf products may be referred to as couplers or controllers but they do not exist at present, although precedents exist for this approach, e.g. the ICA (2000) developed by the CEH Wallingford.

(iii) The FP5 project, HarmonIT (http://www.harmonit.org/), is currently underway and is formulating an appropriate architecture to meet the following aims: to provide the means for implementing the Water Framework Directive; to underpin integrated modelling systems; and to emphasise managing ecology and water quality—acknowledging socio-economical dimensions (Gijsbers et al. 2002).

### Hierarchy

Modularisation through categorisation is only a solution for each modelling discipline but this falls short of maximising the synergy among different disciplines. This is the subject of a separate paper (under preparation) but one of its important aspects is the hierarchy introduced in Table 2. A system of equations can be formulated to handle prediction problems, calibration problems, etc., but the modelling engine varies according to the modelling problem. Interoperability of modelling software products based on open architecture depends on hierarchically structuring the modelling engines.
This table is a generalisation of a similar one given by Dooge (1969), as described in Mahmood & Yevjevich (1975), where the terms used above are defined as follows. ‘Static dataset’ is any input data that consists of any boundary and initial values remaining unchanged during the whole modelling/forecasting period. Any input data in the form of time series that are updated at set time intervals are referred to as ‘dynamic datasets’. ‘System data’ comprises a set of physically quantifiable values describing some catchment attributes, such as survey data. System data also includes parameter values used as part of describing certain physical or flow processes, e.g. calibration parameters. ‘Control data’ describe some physical values that may be set manually or automatically in interaction with the hydraulic behaviour of the system, e.g. gate openings. ‘Output data’ is the values of the dependent properties determined through modelling/forecasting, such as depth, discharge, velocity or gate opening values.

It is clear that each of the above modelling problems can drive the modelling engine composed of a selection of computational modules. Without establishing hierarchies there is the risk of developing a computational module for each of these modelling problems and proliferation of modules. However, by recognising the role of hierarchies and creating appropriate flexibility within each module, this risk of such an unwarranted proliferation can be mitigated.

Some clarifications on terminology

The term ‘computational module’ is a compromise term, as:

- some of these modules are truly based on complex differential mathematical equations, which represent physical, hydrological and hydraulic processes,
- some of these modules are computerised but do not involve differential equations, e.g. blackbox or regression models, and
- even simple techniques such as ‘rules-of-thumb’ can be modules, which do not use a modelling engine but can be a significant tool in the computational environment.

It is also noted that many of the above modules are called models. Some of the connotations of the term ‘model’ include modelling datasets, a set of governing equations, computational modules and prototypes of the physical system. A liberal use of the term model with these connotations in one context is not unusual and, without intelligible qualifiers, should be avoided.

**DISCUSSION**

This paper has posed questions about the spontaneous emergence of open architecture, its top-down or
bottom-up emergence or its growth pathways (problem-specific or lateral spread). The answers are provided on the basis of the shifting stages of the paradigm of software development. However, the interrelation of open architecture and open source needs to be explained. As these two concepts are not the same, the various combinations are presented in Table 3.

This paper regards open source as one mechanism to shape the current practices towards the performing stage of hydraulic modelling capabilities. However, without open architecture, open source is not a viable proposition. Thus, open architecture is of paramount importance and with an immediate impact. With open architecture, modelling software capabilities are transformed into user-designed systems and become living, but not organically growing, products.

This paper sheds light on past, present and possible future modelling software capabilities. It argues that the developments in the norming stage are not spontaneous and they can only be achieved through conscientious practices. The building blocks of open architecture cited already have formed and it is hoped that the initiative of the Environment Agency will be the spark to set off this process. The norming stage should be both a bottom-up and top-down process. Consensus building is required to achieve promotion of the ideals of open architecture through advocating a culture of partnership among model users, model developers and software producers. Finally, when a paradigmatic concept is formed in one discipline of science, it is spread laterally. Open architecture, as discussed in this paper, has been transformed into working tools in other disciplines of science and technology. The paper argues that the realisation of these ideals is also feasible in the field of hydraulic modelling and there is an end user demand for it.

Attention is drawn to the process of the formation of movements on open architecture/source. Individual initiatives are of paramount importance but doomed to be trivial if they remain isolated. Interconnection and networking and other means are effective in creating a critical mass of communities as a reflection of the movement. If the Environment Agency is to promote the movement, care is needed to foster a critical mass of open architecture movements. Creation of networks and a website are some of the possible measures. However, a key measure would be that organisations such as the

Table 3  A matrix of possibilities-architectures versus computer source codes

<table>
<thead>
<tr>
<th>Closed source</th>
<th>Open source</th>
</tr>
</thead>
<tbody>
<tr>
<td>The status quo in hydraulic modelling—alogous to feudalism</td>
<td>A non-viable combination</td>
</tr>
<tr>
<td><strong>Strength:</strong></td>
<td><strong>Strength:</strong></td>
</tr>
<tr>
<td>• Commercial interests create stable markets and stable products</td>
<td>• Limited improvements</td>
</tr>
<tr>
<td>• Clear definition of responsible organisation</td>
<td><strong>Weakness:</strong></td>
</tr>
<tr>
<td><strong>Weakness:</strong></td>
<td>• Major recoding necessary</td>
</tr>
<tr>
<td>• Existing vast software resources remain untapped</td>
<td>• Organic growth not feasible</td>
</tr>
<tr>
<td>• Unconnected new products sink vast new investments</td>
<td></td>
</tr>
<tr>
<td>• Risks of a product becoming obsolete</td>
<td></td>
</tr>
<tr>
<td>• A continual re-invention of the wheel</td>
<td></td>
</tr>
<tr>
<td>Makes user-designed modelling systems feasible—alogous to human mind or nation states</td>
<td>Makes organically growing modelling systems feasible—alogous to globalism</td>
</tr>
<tr>
<td><strong>Strength:</strong></td>
<td><strong>Strength:</strong></td>
</tr>
<tr>
<td>• Makes living products feasible</td>
<td>• Makes living and growing product feasible</td>
</tr>
<tr>
<td><strong>Weakness:</strong></td>
<td><strong>Weakness:</strong></td>
</tr>
<tr>
<td>• A paradigm shift towards open architecture has to be fostered</td>
<td>• A paradigm shift toward open source to be fostered</td>
</tr>
<tr>
<td></td>
<td>• Voluntary software foundations need to be formed</td>
</tr>
</tbody>
</table>
Environment Agency should only express support to research activities with an output of software tools if allowance is made to use published interfaces and to publish their interfaces for their software outputs if one such interface has not been published. The Environment Agency is already pressing ahead with this initiative.

Successful launching of new products is normally associated with risks. However, for the introduction of the open shell and computational systems (and Flood Forecasting Systems) the risks seem to be limited. It is believed that the capability, envisaged in this paper, is already available at least with the major software producers in one way or another. Releasing off-the-shelf open shell does not pose any particular technical challenge. Similarly, modularisation of the various modelling approaches and publication of their interfaces does not pose particular risks. Arguably transforming these capabilities into off-the-shelf modules are feasible but have yet to be realised. However, the cost effectiveness of these new developments is an issue but opportunities for off-the-shelf products are greater than these tangible but minor risks. This is because the potential market within the Environment Agency and outside it, on the international scale, is enormous. Some of the benefits of open architecture include:

- using existing model datasets whatever the proprietary packages,
- combining products for user-designed systems,
- flexibility of using off-the-shelf innovative components, and
- creating competitive environments.

These substantial benefits must, however, be considered alongside the need for organisations, such as the Environment Agency, to deliver a reliable operational public service in flood forecasting and warning. Such benefits may be accrued by promoting open source but these depend on the following fundamental requirements:

- the quality assurance and validation of innovations through open source,
- the custodianship of a definitive reference version of products based on open source,
- institutional support for the long-term maintenance of open source from many innovators.

CONCLUSION

The emerging views of software users, producers and innovators are reflected in this paper. In spite of substantial progress in software development, a great deal remains to be done to transform them into living products. The concept of ‘paradigm’ was used here to explain the roadmap for transforming existing capabilities into living products. This paper postulates that computational hydraulic software is a paradigm and, since its formation, its development shifts through proliferating, norming and performing stages. The shifts from the forming stage of a paradigm to its performing stage are not automatic. The forming stage was traced to the pre-paradigm periods and it was also shown that early programs offered a selective advantage over procedures of traditional empirical hydraulics through speed and the ability to repeat the same operations. The proliferating stage, driven by natural selection, created niche opportunities and changing fortunes. The end result is that hydraulic modelling is dominated by proprietary or bespoke packages with strengths and weaknesses. This paper associated the status of the current software production philosophy with closed architecture and argued that this architecture is poor at meeting a specific number of requirements of model users at the present and in the near future.

The norming and performing stages are conscious processes to identify the deficiencies and pitfalls. The intelligence prevailing in other disciplines of IT proves that software capabilities can be transformed into living products. This paper has taken a strategic overview of the potential developments through the norming and performing stages of the paradigm of software development and identified feasible solutions. Open architecture was presented as the vehicle towards the norming stage and open source as the vehicle towards the performing stage.

This paper holds that the delivery of open architecture for computational hydraulics is feasible and the building blocks for migrating to this architecture from existing closed architecture are already obvious. No problem-solving or research with uncertain outcome is envisaged for the delivery of the open shell, as the engine of open architecture. A major obstacle was identified, this being
the diversity of modelling techniques, but categorisation of modelling techniques and the formation of a partnership among the users, software producers and innovators are capable of removing these obstacles. With the categorisation of modelling techniques, an interface can then be designed and adopted through an extensive consensus building process. When a final outcome is identified every software product and model are revised to adjust to the published interface without the sense of losing or winning and this is not possible without partnership.

ACKNOWLEDGEMENTS

The authors wish to thank Dr Hamish Harvey of Bristol University for his thorough review of this paper and for offering valuable comments and we also wish to thank Dr Alistair Borthwick of Department of Engineering Science, University of Oxford for also reviewing the paper and offering his comments.

Gratitude is also expressed to Mr Jim Haywood for his contribution to this paper and for his leadership on the National Flood Forecasting and Modelling Systems (NFFMS) project, commissioned by the then National Flood Warning Centre of the Environment Agency. Thanks are due to Mr Asghar Akhondi-asl, Mr Tim Harrison and many members of the NFFMS project board, for their contributions into, respectively, managing national and regional projects on flood forecasting systems. Figure 1 of this paper was derived from Mr Harrison’s ideas. Mr Alan Knot and Mr Peter Wade, from project managers Parsons Brinkenhoff, are also thanked for their contributions to the NFFMS project.

This paper puts forward a case for good science and a vision shared by the authors for the benefit of good science. The opinions expressed here are personal and do not necessarily represent those of the affiliated organisations of the authors.

REFERENCES


HarmonIT http://www.harmonit.org/.
Harvey, H. & Han, D. 2002 The relevance of Open Source to hydroinformatics. J. Hydroinf. 4, 219–234.

Khatibi, R. H. 2003 Systemic knowledge management in hydraulic systems: 1—a postulate on paradigm shifts as a methodological tool. J. Hydroinf. 5(2), 127–140.


