How to create and support families of deliverable software products from a common IP pool with high levels of software reuse in a workflow platform

D. G. Thomas, S. J. Cummins and P.W. Cleary

[a] *CSIRO Data61, Private Bag 10, Clayton South, Vic, Australia*
Email: david.thomas@csiro.au

Abstract: The development process for new components for in-house modelling and simulation software frequently involves either writing code from scratch or copy-paste-modifying code that already exists for similar components. These components could all be quite different from the application point of view, but from a software perspective they often have much in common. This paper explores how the use of a workflow system such as Workspace with generic higher-level components can foster high levels of re-use, thereby accelerating and reducing the cost of development, improving maintainability, and improving software quality and robustness. Also important to IP commercialisation is the nature of the Graphical User Interface (GUI) designs and their usability. Rather than build a single monolithic interface possessing substantial complexity the approach followed here is to create simple, custom user interfaces capturing just the capabilities that are needed and presented in a form that is most intuitive to the expected users of the software project. This leads to having bespoke software products for each domain or sub-domain which raises the question of how to easily construct families of related software products from a common IP pool but with simple bespoke user interfaces and how these can be maintained. The basic strategy used is shown diagramatically in Figure 1. The philosophy behind this approach to software product development is explored within this paper. Practical methods for enabling easy customisation of products and user interfaces are also provided. These methodologies, enabled by use of Workspace and Workspace Professional Environment provide a low-cost pathway for the commercialisation of research IP that would otherwise be stranded.

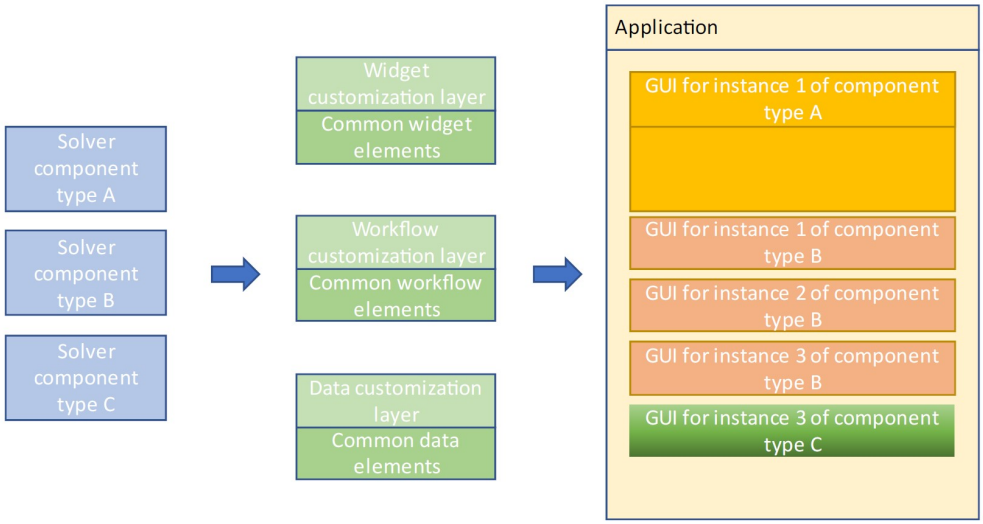

Figure 1. Conceptual flow of customisation process. Left column represents, for example, solver components such as sample planes (these consist of data structures which feed into input files for the solver to consume). The middle column represents the main components we create to encapsulate any of the component types A, B, or C, with each consisting of a common layer and a customisation layer. The right-hand column represents the elements as embedded in a graphical user interface in a custom-application.

Keywords: *Workspace, workflow, commercialisation, software product, re-use, customisation, IP*

1. INTRODUCTION

The most usual method of developing new components for in-house software based on scientific IP seems to be either to write code from scratch or to copy-paste-modify code that already exists for similar components. The disparate solver components, from the application point of view, could all be quite different, but from a software point of view they often have a high level of commonality. Being able to harness this commonality to reduce development and maintenance costs is very desirable. Through use of Workspace and Workspace Professional Environment (WPE) we provide a software framework capability that already enables a good level of re-use thereby accelerating and reducing the cost of development, improving maintainability, and improving software quality and robustness. Despite this base level of reusability, it is still common for developers to build separate instances of similar workflow functionality. A key question is how can we further separate/abstract-away the software commonality from disparate appearing application components to further enhance re-use and to further reduce development and maintenance costs?

A second issue critical to IP commercialisation is the nature of the GUI designs and their usability. Rather than follow the traditional software productisation approach, as used by commercial software vendors, where a monolithic interface of substantial complexity is presented to the user restricting usage to experts in that form of modelling and analysis, the WPE approach is to support the creation of simple, highly custom user interfaces (UI). These allow the capturing of just the capabilities that are needed for a given application, and presented in a form that is most intuitive to the expected users of the software project. This is a fundamental change from the usual paradigm for attempted commercialisation of research IP, and has two major advantages:

1. It enables bespoke, per-application GUIs that significantly improve the user experience and user productivity, and significantly reduce the expertise level required of the user.
2. It is much less expensive to build and maintain. This is critical since a key obstacle to commercialising research IP (particularly modelling and simulation software) is that the target audience is usually small and specialised so high productisation costs typically make commercialisation unviable.

The adoption of a workflow approach with high levels of re-use and substantial ability to customise applications reduces the cost to levels that make such commercialisation viable and often attractive. This provides expanded opportunities to translate existing valuable research IP into usable commercial products.

For research IP which is typically stranded (meaning that it has no commercial path from developer to user), particularly in the modelling and simulation domain these workflow software options provide new possible paths for commercialisation (Cleary, 2017, 2019).

This paper addresses both types of issue and is based on the Workspace workflow platform and the Workspace Professional Environment (tools built on Workspace to support commercialisation). The common IP modelling pool being used to showcase this software design approach is the CSIRO Particle-Based modelling solvers (Cleary, 2004, 2009, 2020a) and their supporting pre-processing, post-processing and visualisation infrastructure which are all built on the Workspace platform.

2. WORKSPACE AND WORKSPACE PROFESSIONAL ENVIRONMENT

Workspace (Workspace, 2014) is a mature Scientific Workflow System, intended to support large scale workflow and application development, is operating system agnostic, and was developed in such a way as to avoid being bound to any specific scientific, deployment or application domain. It has been under continuous development at CSIRO since 2005 and was publicly released in 2014. It continues to be developed with two main user categories in mind:

1. Users who want to create and share scientific workflows in one coherent, easy-to-use environment, where much of the core software development is provided by the platform.
2. Developers or organisations who want to make their scientific software available as standalone products, web services, plugins or components that can be freely or commercially distributed or mixed with capabilities from collaborators, covering the full range of deployment scenarios.

Workspace Professional Environment is a collection of higher-level applications and workflows that are built on Workspace, and which specifically support commercialisation of research IP. Workspace is built on Qt (Qt) and users directly interact with components such as Qt Designer (for interface design and customisation) as well as making indirect use of the underlying Qt framework.

3. PHILOSOPHY FOR SUPPORTING CUSTOMISABLE FAMILIES OF WORKFLOW BASED SOFTWARE PRODUCTS

From a user perspective, many of the elements making up the GUI for a solver-based simulation application look very different from each other. However, despite visual differences, there are often a lot of commonalities in the underpinning data structures, data flow, and graphical organisation.

From an IP licensing perspective, a valuable capability is to be able to make a family of closely related products that vary in a range of ways, without having to construct each application from scratch. End-product software applications based on common IP can vary in a range of ways:

1. Products that vary only in their branding and appearance –
    a. The easiest branding changes to enumerate and effect are simple colour changes, logos, and copyright texts. These are achieved through existing Qt functionality (by setting style sheets for example), embedding customer-specific images into resources which are built into the final application, and (usually) text files containing copyright information.
    b. Other visual changes can be made to graphical widget components (e.g., buttons) via stylesheets via Qt. Such styles can be specified up-front when using a Workspace wizard to generate an application or variation of a base application (Wizards are a sequence of dialogs, launched via the Workspace editor, that walk the user through choices about the component they are creating (e.g., a new widget). These produce "boilerplate" code which even inexperienced users can extend with minimal effort).
    c. Translated text – Qt provides a framework to aid with the supply of translations of text in the application.
2. Products based on the same underlying IP / code pool, but which draw on overlapping but potentially quite different subsets. One example, for a commercial customer, involves two separate but related applications – one of these is restricted (by the nature of the simulation) to a single fluid inflow (a solver-based concept), while the other is allowed to have an unlimited number of such items. In these cases, the level of code re-use can be made to be high for these items – the graphical representation of a single inflow is re-used in each instance. A recently developed WPE component, the "Solver Component Container Framework" (SCCF) is used in the second application to handle multiple instances of such inflows. The SCCF is discussed further in section 6 below. This product-variant category is more complex to design and handle and this is the primary focus of the Workspace and WPE ecosystem. The core intent of which is that everything common can and should be reused. Success in meeting this aspiration is tested by determining what fraction of common capability is manifested through re-used components. The ongoing mission is to continue to build more capability in the framework to support an ever increasing ratio of reuse of common components, with a corresponding decline in bespoke code which has higher development and maintenance cost.

4. WORKSPACE APPROACH TO SUPPORTING FAMILIES OF WORKFLOW PRODUCTS

We start with the intent to provide a higher level of re-use with a collection of operations and workflows that have a lot of commonalities in their internal data structures and data flow. We only want to write each of these once, and then reuse them across all workflow operations that share this common general structure.

Once an area of commonality is identified, we start by extracting common code and workflows. This first level of reuse is the extraction of blocks of code, functions, and / or scripts into packaged operations and workflows. Software engineering skills are highly advantageous at this stage, but the aim of this stage of the process is to produce Workspace components that can be used by the more novice programmer in a simple manner. For example, usage is much simplified by providing wizards to walk the developer through the process of selecting component type, data structure names etc., and to produce code.

An example of an application developed with Workspace is GF-Mill (Cleary et al. 2020), a generic milling simulation software application. This software shares, from a conceptual point of view, a lot in common with other solver-based application, (both actual and nascent) in terms of the visual rendering of simulation construction/setup and of the simulation results, and the analysis and rendering of data.

In the traditional realm of bespoke scientific software, tools are built according to a specific user scenario, and we can have many instances of these user scenarios. We want to avoid just copying large blocks of code from one place to another, which invariably produces highly expensive maintenance hazards. This typically produces a much larger body of code than when using the re-use paradigm. As with any code base, this needs to be

maintained and updated – with any work required to be done on one instance usually needing to be manually replicated across the board – so, the modification of a common core aspect means that each instance of where the code has been copied must also be identified and updated accordingly. Typically, local variations in the code implementation in each different usage case creep in over time, the consequence of which makes rolling out general changes or bug fixes extremely onerous. Although it's possible to design tools (another development overhead) for the automation of some aspects, the software custodian is still left with labour intensive processes to identify, review, and debug a large code base. The associated workload grows exponentially, becoming practically unmaintainable very quickly unless the team has unlimited resources.

The provision of a relatively easy route for code and component generation represents a high value investment return and also helps enforce re-use by being easier to do than to write from scratch. Of course, it is critical that the existence of such components is made visible to the user – one such route we take with the Workspace ecosystem is through the provision of developer wizards (Workspace (2014)).

Despite the intent of the developer to build components with the reuse concept in mind, the reality in many cases is that bespoke special cases / instances tend to be built - mainly as a result of early lack of understanding of requirements, and the time-evolution or drift of these requirements. Despite best intentions, the re-use intent often fails to manifest, and a mass of code is produced with many bespoke copies of components, each varying slightly from the other. This should be addressed to reduce maintenance and future development costs. We try to design up front to limit making such mistakes in the future through re-use of specifically designed components that automatically enforces a valuable level of standardisation. In the Workspace framework, these components take the form of macros, tools and support infrastructure (e.g., wizards), base widget classes, etc., but we also avoid over-designing before the event. For example, say we have one set of application components (code, workflows etc.) to represent a solver item (e.g., a model, a data collection component, etc.) during development. Later, during further stages of development it is determined that another solver item needs to included (perhaps which did not exist at the start of the project, but which has been added since according to the evolving scientific need) and that it has some exploitable level of similarity so that it makes sense to abstract out the commonality and to use the same generalised component in both solver items. Invariably, abstracting before the event (the event being having more than one solver item to abstract from) is a mistake as it is very easy to make the wrong or too many abstraction choices (the latter meaning we end up with needlessly abstract and meaningless developer components that do not create actual increased levels of component re-use).

The Workspace design aim is to enable production of components that are interlinked in a layered way so as to hide their complexity from the user. The development tools heavily involve and leverage Workspace components. For example, widgets connect to Workspace-based data structures on a workflow, with two-way data transfer undertaken by Workspace's framework. Workspace serialisation is also used for data storage.

5. METHODS TO FACILITATE PRODUCT CUSTOMISATION FROM GENERIC SOURCE

Two customisation pathways are taken in the WPE approach:

1. C++ configuration controls. This is the simplest customisation path from an implementation point of view and is achieved by adding customisation controls to a given class – these take the form of, for example, a Boolean flag which tells using code whether a certain element should be included or not (e.g., should a surface mesh of a data set be visualised). As an example, for the display of simulation results, there are many flags and settings that are available in our visualisation tool which not only determine what should be displayed, but how to display them (what shaders to use, what to call the items, depending on simulation objectives and the nature of the overall application in which it is embedded). This route is usually chosen as the first method of customisation during development of a code base. These controls can be used by one of two methods:
    - Manually (direct coding) – the simplest method requiring no extra work other than setting the desired values in the calling code,
    - Via a wizard – this allows a developer to use a provided wizard to lead them through the configuration options. These choices are then reflected either via code generation or another route such as resource generation. In our existing WPE framework, provision of this route is a goal for all new components.
2. Qt designer property controls. A Workspace developer is highly likely to make significant use of the Qt Designer application (Qt Designer). Qt provides a "Property System" (Qt Property System) which allows configuration options (or "properties" in Qt parlance) to be exposed in Qt Designer. These controls are exposed as check boxes, line edits, etc. in the properties interface for any given widget in Qt Designer (the widget needs to be exposed to Qt Designer via the Qt framework – Workspace developer wizards produce code for this purpose to make this type of integration very simple

(Workspace 2014)). For example, a widget may be coded up such that the title can be customised by entering the desired text into the relevant property when viewed in the Qt Designer application. These properties feed through into a .ui file (XML format). A .ui is used by Qt, at the application generation stage, to generate widget code (this includes the main widget / window as well as any sub-widgets). This method of customisation is very useful for presentation to developers of software who may be less comfortable with C++ coding, but it is also used by more experienced developers as it is generally quicker and easier to use when available.

6. EXAMPLE OF RE-USE PRINCIPLES

An example of putting these principles into practice is the recent addition of the SCCF (referred to in section 3 above) to our in-house software repository for applications built on our in-house solvers (although it is general enough to work for other similar, non-solver based items or for other solvers). This framework consists of C++ code, accessible via simple macros (which can be coded by hand or generated by a wizard), along with Workspace workflows and Qt-based widgets.

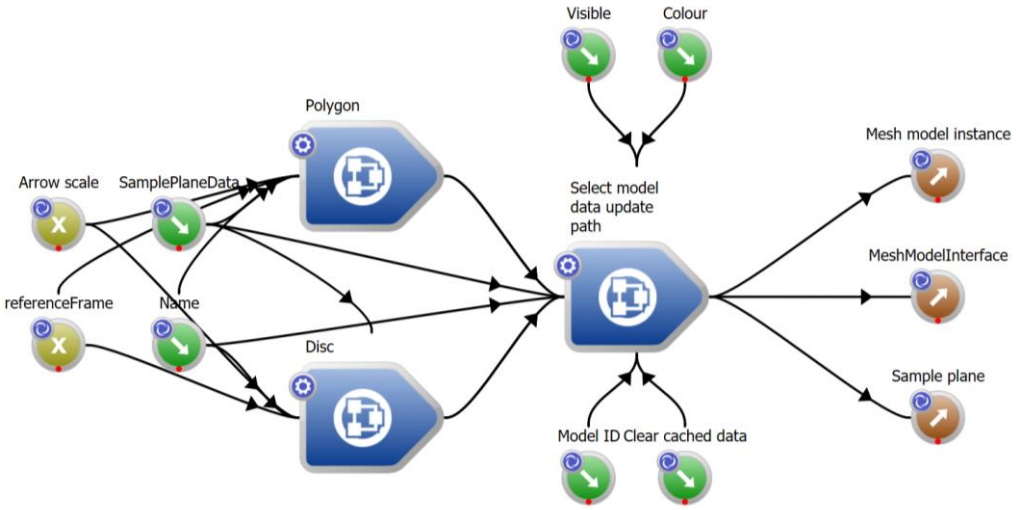

Figure 2. Workflow components for a single sample plane, enhanced for use in a multiple-sample plane environment but still usable in a standalone manner.

Our solvers have components such as sample planes, inflows, etc. These can be included in a simulation for a variety of situations and may or may not be required in any given application developed for a class of problem or end-user. The addition of a single one of these components, because of our re-using and leveraging Workspace components already developed for each individual item, was already straightforward. However, prior to this recent work, adding multiple items still involved some bespoke copy-and-paste of existing components (widgets, workflows, etc.).

An example is the sample plane solver component, used to gather data (in a customisable fashion) during the run of a simulation. Prior to the recent work, each sample plane (the workflow for which is shown in Figure 2) would have to be individually connected to the simulation. With these new framework elements, the developer is now provided with the ability to add, in a relatively simply manner, a general (customisable) number of sample planes to any given solver object. The workflow which can be used to add a collection of these sample planes is shown in Figure 3. The result is that any application using this does not need to maintain its own lists of sample planes, or sample plane settings etc., nor does it need to be concerned about how to, for example, display the sample planes in the simulation construction view. It is also relatively easy to change the GUI view of the sample planes – e.g., they may be displayed in a tab-view layout or a list-view layout. The upshot of this is an increase in the ease of customisability (at the moment this is restricted to the choice of display with items in tabs or in a list), a faster route to producing similar functionality for other components, and bug fixes that improve all code using these components.

This example (sample planes) is for one of the much simpler solver components and is used for the purposes of demonstration. Other solver components with higher levels of complexity benefit even more from using this framework with benefits scaling faster than the increase in component complexity due to the strongly non-linear interactions (both within the workflow and between it and its calling entities). This recent addition to the

WPE framework has made the bundling of such solver components into general containers (both visually and in a data flow) much easier to achieve and is used in currently under-development commercial applications.

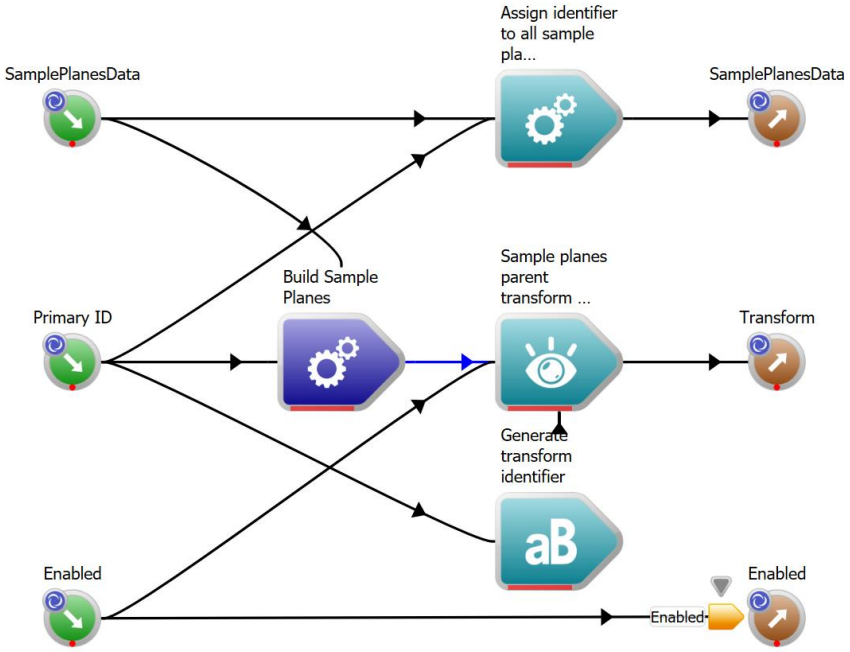

Figure 3. Workflow for the collection of sample planes. This deceptively simple workflow hides behind-the-scenes work to manage the display of the sample planes – the purple operation (labelled "Build Sample Planes") does this work. This workflow is used (and re-used) in various applications. Other parts of the new framework components are data structures, widgets, and naming conventions.

7. HOW THE GRANULAR FLOW IP POOL PRODUCTS ARE STRUCTURED USING WPE TO ACHIEVE RE-USE AND CUSTOMISATION GOALS

In-house, Workspace is used to create a large number of applications leveraging stranded IP (Cleary, 2017, 2019). We deliberately structure our components for use by these applications. We consider the various high-level application building blocks:

1. Pre-processing – i.e., the processing of the input data from, for example, customer models etc., to produce digestible data for the underlying solvers.
2. Running a simulation. Such applications invariably have a "Run and monitor simulation" tab widget. Perhaps unsurprisingly, there is a very high degree of commonality of this component throughout the solver-based applications. This is helped significantly by the solver itself which is developed in such a way as to support this very high-level of re-use at the application level. To get to this practically 100% re-use level, a very high degree of alignment is needed between the underlying source code, workflows, and the application design is extremely useful. This can be either built in at the start or progressively engineered into the software base during ongoing development which then needs to also consider what is needed to support productisation of that type of solver.
3. Visualisation of results – Workspace provides extensive capability options in 2d and 3d rendering, allowing us to build higher-level visualisation applications with broad-based capabilities. These capabilities are exposed to the user on an application-specific basis. (e.g., picking (of geometries) may be needed in one application but not another, or the ability to use a different (or bespoke) graphics shader may be desirable in a particular application). At an abstract level there are two parts to this:
    - The Workspace-based visualisation platform. The visualisation capabilities are mainly bundled in a single, standalone, highly customisable application. We achieve a very high level of re-use as this application is then embedded in the majority of our solver-based applications (wherever 3d visualisation is needed), being customised on an as-needed basis.
    - Application-specific customisation of the visibility of capabilities – we use our customisation capabilities to determine how we expose any given capability. For example, a capability may be exposed in application A, but not in application B (such customisation is discussed in section 5 above). But underneath it is still the same common visualisation application which

has the very significant benefit that only one such application needs development and extension and only one needs to be maintained.

4. Post-processing of results – The primary components for this are constructed as a whole suite of standalone, Workspace-based utilities targeting specific types of solver data or output, and we now have dozens of these applications. They are specifically designed, using the principles we have discussed, to be both standalone and embeddable, and easily customisable. The choice of building many standalone utilities is our design decision aimed at organising and systematising the disparate range of possible data outputs and corresponding data output processing. This decision provides us the top level capability to customise the data analysis options – i.e., the decision is either to include the utility or to exclude it in a specific application product. This is a top-level application design decision. This deceptively simple aspect provides a large and critical capability to build bespoke modelling applications and particularly to have families of related applications that can be built and maintained at the same cost of doing only one. To facilitate this choice, our framework provides one of the simplest coding (C++) interfaces possible, namely a single method call to add each such utility to an application. This, along with the abilities to customise the software appearance and function at several levels, is critical to meeting the goal of being able to have as minimal and specific as possible GUI's for each application in the family.

8. CONCLUSIONS

In this paper we have considered the problem of how to structure software components within a workflow environment to create high levels of re-use at workflow, workflow component and application scale. This allows the building of multiple software products based on a common IP pool with relatively low investment and ongoing maintenance cost. Critical to this is the ability to customise the appearance and function selection of each software product. This allows multiple products with bespoke, simple user-friendly GUIs to be created and maintained efficiently. These methodologies, enabled by use of Workspace and WPE provide a low-cost pathway for the commercialisation of research IP that would otherwise be stranded.

REFERENCES

Bolger, M., Cleary, P. W., Cohen, R., Harrison, S.M., Hetherton, L., Rucinski, C., Sankaranarayanan, N., Thomas, D., Watkins, D., and Zhang, Z., (2016), Workspace: a fast and low cost methodology for delivering commercial applications based on Research IP, eResearch Australasia Conference, Melbourne, VIC, Australia, October 2016, Link: http://hdl.handle.net/102.100.100/89557?index=1

Cleary, P.W., 2004. Large scale industrial DEM modelling, Engineering Computations, 21, 169-204.

Cleary, P.W., 2009. Industrial particle flow modelling using DEM, Engineering Computations, 26, 698-743.

Cleary, P.W., Watkins, D., Hetherton, L., Bolger, M. and Thomas, D., 2017. Opportunities for workflow tools to improve translation of research into impact, 22nd Int. Congr. on Modelling and Simulation (MODSIM 2017), Hobart, Tasmania, Australia, 3-8th December.

Cleary, P.W., Hetherton, L., Thomas, D., Bolger, M. and Watkins, D., 2019. Translation of stranded IP to commercialisable software applications using Workspace, 23rd Int. Congr. on Modelling and Simulation (MODSIM 2019), Canberra, Australia, 1-6th December.

Cleary, P.W., Sinnott, M.D., Cummins, S.J., Delaney, G.W., and Morrison, R.D., 2020a. Advanced comminution modelling: part 1 - crushers, App. Math. Model., 88, 238-265.

Cleary, P. W., Thomas, D., Hetherton, L., Bolger, M., Hilton, J. E., and Watkins, D., 2020b. Workspace: a workflow platform for supporting development and deployment of modelling and simulation, Mathematics and Computers in Simulation, 175, 25-61.

Qt, https://www.qt.io/

Qt Designer, https://doc.qt.io/qt-6/qtdesigner-manual.html

Qt Property System, https://doc.qt.io/qt-6/properties.html

Subramanian, R., Cleary, P. W., Thomas, D. and Cummins, S. J., 2021. Extending the capabilities of applications built using the Workspace architecture, MODSIM 2021, Sydney, Australia.

Watkins, D., Thomas D., Hetherton, L., Bolger, M. and Cleary, P.W., (2017). Workspace – a Scientific Workflow System for enabling Research Impact, 22nd Int. Congr. Modelling and Simulation (MODSIM 2017). Hobart, Tasmania, Australia, 3-8th December.

Workspace (2014), https://research.csiro.au/workspace/

Workspace Introductory Tutorials, (2019), https://research.csiro.au/static/workspace/docs/workspaceusertutorials.html

Workspace Developer Tutorials, (2019), https://research.csiro.au/static/workspace/docs/workspacedevtutorials.html