TOOKUIL: A Case Study in User Interface Development for Safety Code Application

CONF-961192--

D. L. Gray, C. K. Harkins, et. al.

November 1996

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States, nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

KAPL ATOMIC POWER LABORATORY
SCHENECTACY, NEW YORK 12341

Operated for the U. S. Department of Energy by KAPL, Inc. a Lockheed Martin company
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
TOOKUIL: A Case Study in User Interface Development for Safety Code Application

David L. Gray
Cynthia K. Harkins
Jeffrey G. Hoole
R. Cary Peebles
Richard J. Smith
KAPL Inc.

Abstract:
Traditionally, there has been a very high learning curve associated with using nuclear power plant (NPP) analysis codes. Even for seasoned plant analysts and engineers, the process of building or modifying an input model for present day NPP analysis codes is tedious, error prone, and time consuming. Current cost constraints and performance demands place an additional burden on today's safety analysis community. Advances in graphical user interface (GUI) technology have been applied to obtain significant productivity and quality assurance improvements for the Transient Reactor Analysis Code (TRAC) input model development. KAPL Inc. has developed an X Windows-based graphical user interface named TOOKUIL which supports the design and analysis process, acting as a preprocessor, runtime editor, help system, and post processor for TRAC. This paper summarizes the objectives of the project, the GUI development process and experiences, and the resulting end product, TOOKUIL.

1.0 Introduction

The majority of present day computational tools used for nuclear power plant systems and safety analysis share some common characteristics. They are fairly general in their applicability and system representation capabilities. They have a great many options which can be invoked (typically on the order of 1000). There are many interdependencies among the options, the various system modeling features, and engineering objectives. These characteristics, while necessary for accurate and effective nuclear power plant (NPP) system modeling, result in a very large number of input parameters and an extremely complex set of rules for generating these input. Traditionally, NPP and safety engineers have had to manually sift through large quantities of data, learn and understand volumes of manuals, and build a high level of expertise in applying manual rules, interfacing with computer operating systems, and processing input and output. The end result is that the cost of modeling and analyzing NPPs is very high. Minimizing the cost (resources) associated with such efforts while maximizing analysis quality is critical to effectively carrying out these tasks.

This paper presents an effort undertaken by KAPL Inc. to improve plant analysts’ productivity and model/analysis quality assurance when using the Transient Analysis Code (TRAC, developed
at the Los Alamos National Laboratory) (see References 1 and 2). KAPL Inc. has invested in the development of a graphical user interface (GUI) for TRAC called TOOKUIL because, as numerous studies have shown, GUI technology significantly improves productivity and reduces the number of errors compared to standard text editing. TOOKUIL is a combination of computer programs and methods for enabling the engineer to effectively interact with computer systems. It provides a more intuitive approach to model development and has the ability to incorporate TRAC modeling expertise. Furthermore, TOOKUIL supports the engineer throughout the entire power plant analysis process from input model development to processing and visualizing the output.

The TOOKUIL user interface was developed following a software development life cycle, whereby user needs and requirements are defined; a conceptual design is developed and implemented as a prototype; a final design is coded, tested and released to the user community; and the end product is supported and enhanced as dictated by user needs. The TOOKUIL development process is described in further detail, along with KAPL’s experience with graphical user interface development.

2.0 What is a Graphical User Interface

With the advent of window based interfaces, more computer programs are providing graphical user interfaces. A graphical user interface (GUI) uses one or more windows, or panels, to present and collect data. A variety of graphical objects, such as push buttons, scrolled lists, menus, toggle buttons and sliders, may be used to present data. The user interacts with these objects using the mouse and keyboard. The GUI gathers information and passes it on to the underlying program.

A GUI often extends beyond the capabilities of the analysis program to which it is coupled. This extended functionality can include encapsulating expertise for making the system easier to use, adding features to support the whole analysis process, providing an extra layer of quality assurance and quality control, and enhancing data and file management to reduce user burden.

A GUI is not a glorified text editor. Rather, a GUI interprets & analyzes (parses) the input file, has encoded knowledge of the input manual logic and is able to create/interrogate interrelationships among the data as required. It can write various formatted output files for use by the analysis code, data post-processors, visualization tools, etc. Because it has knowledge of the data and logic, expert systems-type functions may also be incorporated to help users with modeling and analysis decisions.

2.1 Ease of Use

The NPP analysis process typically requires the engineer to describe the power plant configuration, geometry, initial conditions, specialized component actions and boundary conditions, and the power plant control system. This information is usually conveyed to the analysis program via a text file containing thousands of lines of input data. Dealing with this large and complex data file presents an array of issues related to data management, intuitiveness, change control, model sharing and the like.
A GUI can be used to organize and present data in a more intuitive manner. For example, an engineer may define a model by drawing a schematic rather than entering blocks of numerical data. This makes it easy to see relationships among model components and the total plant layout. Each component in the schematic can be automatically populated with default data. Model data can be entered through pop-up windows. These pop-ups can be organized to present data in a logical manner. The user interface can dynamically check the data being entered to make sure it is valid and also to determine if there is a relationship between the entered value and other input values. If the entered value causes additional information to be needed, the user can be prompted accordingly.

A GUI can also provide a mechanism for sharing data and supporting top-down design. For example, a component library may be used to supply a standard set of components from which users can build their models. The component library may also include customized components and submodel components. These specialized components can be developed and shared among power plant analysts.

A good GUI provides support for multiple levels of users. Access to functions via menus is useful to the novice user, whereas keyboard shortcuts are convenient for the advanced user. On-line help, tutorials, and guidelines can help users get started or obtain additional guidance during the modeling process.

Another major benefit of a GUI is that it eliminates the need to understand the input file formats and interact with the computer operating system. Once the input has been entered, the required input files for the analysis code as well as geometry information for later post-processing/visualization can be automatically generated. Options for running the analysis program can be included so that the engineer does not need to leave the user interface environment in order to make runs. This can include capabilities for running the program in a batch mode. In this case, the required job control information is automatically generated. The engineer is not burdened with details of creating input files, job control files, and interfacing with the computer on which the analysis is run. The engineer’s time is spent doing engineering rather than data processing.

2.2 Support for the Plant/Safety Analysis Process

A well designed GUI creates an environment that supports and seamlessly integrates with the analysis process. This process includes researching and creating models, making numerous sensitivity runs to evaluate changing model conditions, processing and interpreting the output, and documenting the findings. It also includes managing changes to the input model, tracking and managing input and output, and managing links to databases and other components of the analysis process. It may also include linking/integrating the target analysis code with other codes in the overall analysis process.

The task of creating a model may require research into what models have been created in the past and it may also require information from other design and analysis programs. The model building task can be supported by providing access to sources of information such as a database of existing models (e.g., CAD systems) or results from other analysis programs. The user interface can also include a library of standard components, specialized components, submodels and models to use
as building blocks. Artificial intelligence and expert system technology can be utilized to help the engineer build the model more intelligently and to guide the analysis process.

As an engineering tool, a key feature of a safety code GUI is to be able to generate and run parametric cases from a base case model. The base case model may be a plant operating at steady-state. The parametric capability can be used in many different ways. For example, the parametric runs may restart from the steady-state calculation and perform multiple variations of a given transient scenario. Another possibility is that parametric runs use the steady-state restart as a starting point to reach other steady-state operating conditions. The parametric capability may be used to adjust plant operating conditions for a given plant model or may be used to adjust the plant model itself (i.e.- geometry, component characteristics, etc.) for a given operating condition or transient scenario. Consequently, the GUI needs to be flexible enough to adjust multiple input parameters for each parametric case yet remain simple enough for the user to easily understand what parametric studies have been defined and saved. The GUI can directly support the Code Scaling, Applicability and Uncertainty (CSAU) methodology (Reference 5) by automating the calculational matrix for the sensitivity analysis. Such calculational matrices can be made up of hundreds of runs. Each of the runs adjusts input parameters or combinations of parameters over their uncertainty ranges.

The parametric analysis can be further enhanced by integrating optimization software and artificial intelligence technology. This software is particularly useful in the design process where one needs to evaluate the results from a given run, adjust appropriate input parameters and perform further runs so as to arrive at the desired design configuration. It can also be useful in a probabilistic risk assessment role by having the system automatically search for the worst case accident scenario.

A GUI can also improve overall process efficiency by facilitating the coupling of the plant/safety analysis to other analyses. For example, the plant/safety analysis code may be used to drive downstream calculations such as detailed Computational Fluid Dynamics (CFD) calculations or structural analyses. The GUI automates the process of gathering the appropriate output from the plant analysis and coupling it to the downstream analysis. Furthermore, the GUI can be made to automatically initiate the downstream jobs and process their output as well.

Post-processing is another area in which GUIs are useful. The output generated by safety analysis codes is large and not always easy to read. In some cases, the output is in binary format, so that the only way to get at the data is via another program that extracts the data and creates an ASCII file. These output files generally require the engineer to sift through large quantities of numbers and do not give the him/her a clear picture of what is happening in the model. A GUI can solve this problem by reading the output data file and presenting it to the user in an understandable manner such as an x,y plot, contour plot, or a 3-D animation. Access to third party or proprietary visualization and analysis tools can be provided without the user ever leaving the user interface environment. The GUI can be coupled to a statistical software package, as well, for generating performance fits and response surfaces. The user is freed from having to know the numerous file formats used by these visualization tools or how to access these tools. Rather than spending a great deal of time scanning numerical output files, the engineer is able build a deeper understanding of the phenomena and processes occurring in the power plant.
2.3 Quality Control and Quality Assurance

Quality control and quality assurance are important issues when dealing with safety analysis codes. One reason for this is the size and complexity of the input files. A change to a single parameter can change the input requirements for other portions of the model. Checking for data dependencies and making sure a model is correctly and consistently specified is a cumbersome process which can easily be performed by a GUI. As the engineer enters parameter values, a GUI can check the validity of the value and its impact on the model’s input requirements. Visual cues, such as prompts or highlights, can be used to indicate changes to the input required. Model inputs often consist of table or array data. Entering this data into a text file is tedious and error prone. A user interface can make this task easier by providing functions to read data from various sources such as files, databases, or other tables. Visualization techniques such as x,y plots or contour plots can be incorporated right into the input pop-up to help the user verify array or table data being input. These features help guarantee correct model input.

Safety analysis codes often contain redundant information. For example, the user may be required to input parameters that could be derived from other input parameters. The user must ensure these parameters are always consistent. A change to one requires a change to the other. Redundant information is often contained in component boundaries. For example, the flow area at the end of one component must match the flow area of the component to which it is connected. A GUI can reduce the amount of redundant data by automatically calculating derived values and updating duplicate values.

Building an input model to an NPP code such as TRAC requires the user to translate a set of plant characteristics into a set of parameters that the safety analysis code understands. This translation process may not be intuitive since there may not be a direct mapping of physical characteristics to model input. The plant analyst may have to set many parameters in order to achieve a single model characteristic. A GUI can help the user by providing a standard set of components with default values from which they can build their model. User defined components having specific characteristics can be added to the standard set. The interface can provide on-line help to guide the user through the model building process. A more sophisticated user interface may incorporate expert system technology so as to allow the engineer to specify higher level or physically based characteristics. The interface then automatically generates the representation required by the NPP code.

Query tools and report generators can be provided so that the user can quickly view the input data. Methods can also be incorporated so that the model is self documenting. In addition to user comments/documentation and the general input variable annotation, facilities can be developed within the GUI that interpret the model input, particularly hardware action scenarios, and print a report describing the sequence of events that are set up in the model. This capability is particularly useful for archival purposes.

3.0 TOOKUIL: A Case Study

TOOKUIL is a graphical user interface developed by KAPL Inc. for the Transient Reactor Analysis Code (TRAC) (see References 3 and 4). This GUI was developed to support the design
and analysis process by providing features for creating TRAC models, running TRAC calculations, managing the resulting output files, and post-processing the output data.

3.1 TOOKUIL Features

The TOOKUIL Main Window is shown in Figure 1. It consists of a component library containing the standard TRAC components (e.g., pipe, valve, tee), an "infinite" workspace for drawing a model schematic, and a message window in which warning, error and status messages are printed. At the top of the Main Window is a menu bar that provides functions such as opening and saving models, editing global options, running TRAC, opening on-line documentation and accessing post-processing tools. References (3) and (4) provide detail on the functionality of the Main Window.

Once a model is complete, the engineer can run a TRAC calculation using the model by selecting the Submit option from the Runtime menu (Figure 2). The user specifies runtime options such as where to run the TRAC job and where to store the resulting output files. The ability to perform restart calculations is also supported through the submit window. A restart selection pop-up window allows the user to specify which components in the model (if any) are to be resupplied (i.e. - redefined) for the transient restart calculation. The components selected to be resupplied are highlighted in the model schematic (Main Window). Before submitting the TRAC job, TOOKUIL checks the model to make sure all required inputs have been specified. Finally, TOOKUIL provides tools for monitoring the progress of jobs that have been submitted.

Once a model is run, the engineer needs to evaluate the results and make decisions based on these results. TOOKUIL provides automated links between the TRAC graphics file and various data visualization and processing tools. The TOOKUIL Output Processor (Figure 3) displays a list of the graphics data available from a specific run (the graphics data catalog). The user selects the data to be extracted from the graphics file and specifies how to format this data. The Output Processor extracts this data from the graphics file and reformats it as appropriate. Post-processing and visualization tools (Figure 4) are then accessed automatically from within the TOOKUIL Output Processor.
FIGURE 1. TOOKUIL Screen
FIGURE 2. Runtime Editor
FIGURE 3. TOOKUIL Output Processor
3.2 TOOKUIL System Summary

TOOKUIL is a stand-alone program running in a UNIX environment. In its current configuration, there is no direct (i.e. interactive) communication between TOOKUIL and TRAC, which can be run on super-computers or workstations. TOOKUIL generates a TRAC input file and invokes TRAC via a batch queuing system. All job control information is handled by TOOKUIL so that input and output files are transferred and stored correctly.

TOOKUIL, which was implemented in C, uses the X Window System and Motif widgets to perform all graphical user interface capabilities. The GUI requires the X11R5 version of the X Window System. It has been implemented and tested on both Sun and SGI workstations.

3.3 TOOKUIL Data Flow

Figure 5 shows how data flows through the TOOKUIL system. The annotation in the figure shows the order of the data flow. (1) The TRAC model is created via TOOKUIL. This model may be created interactively by reading in an existing TRAC input file or accessing TOOKUIL model...
files. (2) When the Submit option has been selected, TOOKUIL generates a TRAC input file and job control file. These files are transferred as appropriate to the computer on which TRAC runs. TRAC will read in the input file, which may be a base model file or a restart file. (3) TRAC produces several ASCII and binary output files. The binary files are generally stored on the machine on which the run was made. The ASCII files are returned to the engineer’s workstation area. (4) If the user wishes to use TOOKUIL’s Output Processor, an extraction job is run to extract a catalog of graphics information. (5) The catalog extraction program generates an ASCII catalog file that is returned to the user’s workstation. This catalog is then loaded into TOOKUIL’s output processor. From the output processor window, the engineer can select the data to be extracted and specify the data formats. (6) The output processor runs a second extraction job to extract and format the requested data. (7) The formatted data files are returned to the workstation, where they may be loaded into the appropriate post-processing tools.

4.0 TOOKUIL Development Process and Lessons Learned

There are many textbooks that define software development methodologies. The standard software development life cycle roughly follows the following steps: (1) user needs and requirements are defined; (2) a conceptual design is developed and implemented as a prototype; (3) a final design is defined and coded; (4) the software is implemented and released to the user community, and (5) the software is continuously supported and enhanced as dictated by user
needs. Throughout the process, the software is tested and user input is sought. In practice, the lines between the steps of the development process are blurred, particularly once the software is initially released. It should be noted that if existing GUI products are available to use as the starting point for further development, the upfront phases of the process may be substantially reduced.

4.1 Defining the Project Team

The first step in any development effort is to pull together the project team. This team will most likely evolve over time. The team should include someone knowledgeable about the safety code and how it is being used within the safety analysis community. The team also needs someone with knowledge about graphical user interface technology. It is useful to have a team member who is familiar with current industry trends and software development tools. Software project management, database, and visualization/post-processing tools skills and knowledge may also be required. In the earliest phases of design, the team is best kept small in order to effectively manage their work. Once the project has been defined, it may be useful to have additional programmers, depending on the size of the project and the schedule.

The TOOKUIL development team consisted of two software engineers familiar with GUI development, a lead engineer responsible for overall project management and an engineer familiar with the details of TRAC use. During the early stages of TOOKUIL development, this team proved to be very effective.

4.2 Process and Requirements Analysis

A big step when designing any software system is to understand the safety analysis process and how the GUI can/should fit into that process. Because many different communities may use the safety analysis code in different ways, the project team must be sure to get input from all of these communities. During this stage, the project team can identify parts of the existing process that may be cumbersome or unnecessary and redefine the process to eliminate these items. The team also needs to determine if the GUI will be specific to a particular safety analysis code or if it will be capable of interfacing with multiple codes.

Another part of requirements analysis is determining hardware and software interface requirements. Some of the questions that need to be addressed include: (1) how is the GUI going to be coupled to the safety analysis code and how will they communicate; (2) are there any other external programs that the GUI must interface with such as a database; (3) from what hardware will the user be accessing the GUI; (4) will the GUI be implemented on a computer network; and (5) what potential hardware platforms may be used in the future and what operating systems and software will this hardware be using? Other issues which need to be addressed at this stage include the organization’s policies toward use of third party software and software support, protection of proprietary information, and access to the Internet. All these issues can greatly impact the GUI implementation.

Once the process and requirements are understood, a formal requirements document should be written. This document must be reviewed by the user community to ensure that nothing is missing and that the process and requirements are understood by the project team.
During this stage, the TOOKUIL development team worked with TRAC users at KAPL to understand how TRAC was being used. Some of the requirements that came out of these discussions included the ability to more easily generate a correct TRAC input file, reduce model development time while increasing quality, and provide easy mechanisms to run TRAC, including support for parametric studies. Portability across UNIX workstations was also required. This lead to the decision to use X/Motif to develop the GUI rather than a proprietary windowing toolkit. User requirements did not dictate the need for the GUI to interact with the code while it was executing. Thus, TOOKUIL was designed for a batch interface with TRAC. It generates a TRAC input file and provides post-processing for TRAC output. It was also determined that TOOKUIL would initially only provide an interface specific to the TRAC code. The TOOKUIL team generated a comprehensive requirements document that was reviewed by the current user community.

4.3 Conceptual Design

During this stage, the project team must determine how the GUI will meet all the requirements identified in the previous stage. This is the most creative stage of the development process. A small core of developers and users (i.e. users of the safety code) discuss how the GUI might look. There are usually several radically different proposals and “whiteboard” designs. Early decisions such as hardware platform(s), operating system platform(s), etc. are formulated during the conceptual stage. The various approaches are presented to users for feedback. Often, “storyboarding” is used to walk users through each conceptual GUI. These storyboards may be in the form of pictures or prototypes. During this stage, the users and developers can make sure that all the requirements are adequately met.

4.4 Prototype

Once the developers have an idea of how the GUI should look, they may need to do some “proof-of-concept” prototyping. This allows the development team to try out new technology, development tools, and get an idea of what works and what does not. A small subset of the features of the safety code are actually modeled in the prototype and error checking and recovery are not considered (unless that happens to be part of the feature scope). Besides giving developers a test-bed for trying different ways of implementing GUI features, a prototype gives users an early opportunity to test drive the system and provide feedback.

There are several important decisions made during the prototyping stage regarding platform as well. This stage will most likely determine the implementation (programming) language for all further development, the mechanism for interfacing with the GUI platform (i.e. X-Windows library calls, 4GL GUI builder, etc.), and internal data organization, representation, and management. The selection of programming languages and tools should also consider the potential effects of using proprietary software on the longevity of the GUI. The prototype stage will also, formally or informally, introduce the beginnings of the application style.

TOOKUIL started as a prototype. The initial version included just the PIPE, BREAK and FILL components of TRAC. The intent of the prototype was to try out different methods of drawing a schematic using Xlib and Motif. This prototype became the basis for the production version of
Because the prototype initially dealt with very simple TRAC components, the development team ran into some obstacles when some of the more complicated components, such as the heat structure, were developed. This required major rework that could have been avoided if the time had been taken to develop a more comprehensive design. In addition, the functions for displaying pop-up tables were developed specifically for inputting geometry data. It was difficult to use these functions for creating more general purpose tables. As a result, TOOKUIL contains many different routines for creating tables. In hindsight, it would have been helpful to identify the generic objects needed for the GUI. This experience made it apparent that a fair amount of design work needs to be done in the prototype stage. A prototype should not necessarily include the easiest elements to implement, but rather those elements that may be the trickiest. Error handling must be included during this stage because it is very difficult to back fit it into the code.

4.5 Initial Development

The prototype is usually used for demonstrative sessions with users and management. After prototyping, initial or “Phase-I” implementation begins. The term “Initial” is used here because it is usually not realistic to produce a working GUI that meets the entire set of requirements in the early releases. The initial implementation expands the prototype in scope to include enough features to run the safety analysis code. During this stage, added error checking makes the GUI more robust. As with the prototype, user feedback is very valuable at this stage and is one of the principle reasons for this development stage. The user feedback obtained during this stage may actually change the requirements or at the very least, the method of implementation for some of the requirements.

Undoubtedly the most difficult step in the initial development stage is deciding which of the features identified in the requirements analysis stage to leave out of the GUI. The importance of different features can vary dramatically depending on the user and how they are using the analysis code. For example, some users may have very complex models and require extensive model building capabilities. Other users may need to make hundreds of runs so runtime features, such as support for parametric studies and data management, take a higher priority. Another set of users may need output processing support for report generation.

The TOOKUIL initial development effort has incorporated most of the features necessary to create a model, run TRAC and perform 2-D and 3-D visualization and spreadsheet related post-processing. Features include reading an existing TRAC input file and converting it to TOOKUIL model files, building a schematic of the model, automating TRAC input manual logic in the data entry panels, TRAC input file generation, and TRAC job submittal and data access tools for use in post-processing. Data entry panels were developed for the most commonly used components. Components such as the TURBINE and VESSEL, as well as TRAC control procedures, were implemented as text editors in which the user was required to enter the component data by following the TRAC input manual specifications. Other features that were also put off for future development included several schematic editing features such as zooming and group editing, more sophisticated model checking and model management, schematic-based control procedure definition, and support for parametric runs.

TOOKUIL development during this stage progressed more slowly than the prototype development. There is much more to a GUI than appears on the screen. Most computer-aided GUI
development tools can improve the developer’s productivity when designing the layout of a GUI. However, the development team must still contend with designing efficient data structures; memory management logic; input file parsing and generation logic; cut, copy, paste, drag-drop, and undo logic; file management (i.e. autosave, creation of backup files, etc.); and the like. All of this development must be done and maintained regardless of the complexity of the GUI’s interface with the user. Thus, there was a relatively large initial development cost to arrive at TOOKUill’s current level of capability. Future enhancements, however, do not need to incur such a heavy cost, as they can build significantly off of the existing infrastructure.

Prior to the initial release, TOOKUill was tested out by the development team. This testing focused on two major categories. First and foremost, the TRAC features that were implemented in TOOKUill were tested to show that they were indeed operational. The members of the development team that were experienced TRAC users were crucial for this testing. The second type of testing performed by the development team focused on trying to find incomplete logic (i.e. trying to break the system). This involved performing operations in various sequences, pressing various key sequences, etc. Many bugs were found and eliminated through this testing. At this point, TOOKUill was released to a subset of users. These power plant analysts were used to identify bugs which could only be found through real production use. This stage is extremely valuable and close user-developer interactions are key to its success.

Once the first version of TOOKUill was released, several things became apparent. First and foremost, the GUI had a significant positive impact on bringing new users up to speed on the code. TRAC is a very complex code and the manuals are daunting. TOOKUill encapsulates the manual logic/requirements within it and helps guide new users in model creation and data post-processing. However, the initial release of TOOKUill met with a significant level of resistance from engineers who were experienced with using TRAC. These users were very proficient at editing the input deck by hand (with a text editor), interfacing with the various computer operating systems, and utilizing the available post-processing tools. Initially, the GUI actually slowed down these experienced users. Furthermore, these users also utilize the safety code in an advanced (efficient) manner that was not supported by initial releases of the GUI. The experienced users desired features such as model management, parametric analyses and expert systems type capabilities, many of which are being worked on now. Most of the users issues associated with the initial release of TOOKUill have been resolved. Advanced development work is proceeding in parallel with production use of the GUI.

Finally, a ripple effect of the release of the GUI was that new development essentially stopped for a period of time following the first release of TOOKUill. During this period, a great deal of time was spent fixing bugs and helping users learn the new system. Team members spent a lot of one-on-one time with users as well as giving group demonstrations. A walk-through tutorial in the TOOKUill User’s Manual would have been helpful for users who were basically familiar with the TRAC code and creating TRACIN files.

4.6 Advanced Development

Initial development work is focused on getting a robust, capable working system into production. Once the GUI has been in production for a period of time, the developers get a sense for what the next step should be. The next set of features to be implemented can be identified and an
implementation plan developed for the next set of releases of the GUI. During the more advanced stages of development, more consideration can be given to issues such as process integration, integration with other codes, model data management, version control (of both the safety code as well as the GUI), model sharing, etc.

These considerations are very broad in scope and difficult to design in adequately during early development stages. However, once this advanced point in the development cycle is reached, substantial system changes are more difficult, costly, and functionality may need to be downgraded through compromise. During advanced development phases, user needs are reassessed, development priorities are solidified, and functional capabilities updated to incorporate key computer system and safety code upgrades that have occurred. This work often must take precedence over larger usability related development tasks. As usage grows, the development plans must begin to consider legacy data where they had not previously. Coupling the GUI to the design process or implementing a data management scheme may require changes to the storage of both the data for the safety code as well as the enhancing data the GUI captures and maintains. It must be decided whether the GUI will be backward compatible with these older formats or whether a conversion utility should be developed. Incorporating advanced functionality becomes increasingly complex.

TOOKUIL is currently in an advanced development stage of its life cycle. Most of the basic model building capabilities exist, along with support for running TRAC and performing data post-processing functions. TOOKUIL fully parses the input deck and all user manual logic has been encoded. This positions the GUI to implement expert system type functions to act as an Engineering Assistant. TOOKUIL has the potential to act in an interactive manner with the code as dictated by user requirements. TOOKUIL has access to substantial 2-D and 3-D post-processing capabilities and tighter integration between the model generator and these visualization tools is possible. The data structures in TOOKUIL are designed for implementation on top of a database (library) of component/model information which will facilitate additional user capabilities and increased configuration control of modeling information. A model validation database was implemented which uses a commercial database system (Oracle) and TOOKUIL's knowledge of the data in the verification model database to dynamically assess a user’s model against the base code validation matrix. Model diagnostic information is also available to help users assess the complexity of their model against others of a similar type. TOOKUIL's schematic drawing capability also lends itself for use as an interactive control procedure generator to automate use of the TRAC signal variable and control block input.

During the advanced development phase, there is increased pressure on the development team due to support/maintenance related needs. In order to keep both production code support and development moving forward concurrently, it is crucial to ensure that the development team is staffed properly and has the necessary mix of skills (e.g. GUI programming, safety code use and model building, data management, process integration and mapping, and software development for maintainability).

4.7 Software Validation

Verification and testing of an interactive GUI such as TOOKUIL is a challenging task. The process is dependent upon the type of capability to be tested. There are two primary categories of
capabilities which require different types of testing. The first is a change to TOOKUIL to support enhancements to the underlying code, TRAC. The second is a change to elements of the GUI which are independent of TRAC. In both cases, modifications are tested individually and in conjunction with other modifications and the base code.

Modifications to TOOKUIL to support new features in TRAC require several common types of changes and as such, several common testing procedures. Typically, new TRAC modifications require a new input. The internal data structure in TOOKUIL needs to be modified to include a place to hold the value of this input. The routines that parse and build the input deck (TRACIN) must be modified to understand this value and write it out properly. These changes are usually made before any GUI elements of TOOKUIL are changed. This allows for easier testing since this portion can be coded, compiled and tested completely without significant GUI changes. Once the new input is able to be read in from the input deck, stored internally, and rewritten to the input deck, the changes to the GUI can begin to allow the user the ability to view / update the value of this input. Once completed, this new GUI input field capability can be verified visually and through inspection of the resultant input deck.

There are numerous routines in TOOKUIL that define the characteristic nature of the system and are not related to TRAC input model options. For example, the routines that generate a pop-up dialog for querying geometric inputs from the user or routines for cutting and pasting components are TOOKUIL features that have no bearing on the nature of the communication with TRAC. Changes of this nature to portions of the system that are used in only one location of the GUI are relatively simple to test. For example, when rearranging the layout of a pop-up dialog window, testing the changes can be done by displaying the window. If however, a generic routine was developed to automatically layout a pop-up dialog window, a change to this would require more testing. Cut and Paste logic is an area that is generic to all components but requires extensive testing. This logic must account not only for the differences between the component types, but must also account for the states of the components. For example, if the user attempts to “Cut” out a component and a pop-up window associated with this component is currently open, the cut logic must be able to detect the presence of this window and know how to eliminate it.

The overall validation process involves several steps. The first level of testing is referred to as unit testing. Unit testing is how the individual programmers test specific code modifications. Typically a release of TOOKUIL involves the integration of several modifications, potentially from multiple developers. Prior to a release of the system, the developers use the system with all of the changes integrated and retest specific updates as well as many of the (base) general system functions. This is usually referred to as an integrated system test. The purpose for this is to ensure that the modifications from the multiple enhancements do not conflict with each other or improperly impact other GUI functions. After an integrated system test, the system can be given to users for testing. TOOKUIL development has typically made the new version of the system available for a limited number of users initially for a Beta testing period. Following successful completion of user testing, the system is then released for production use.
5.0 Summary

It is generally recognized that a well organized and functionally comprehensive graphical user interface can significantly improve the productivity and quality assurance of data entry and processing tasks. This observation provides the rationale for the development of an integrated graphical user interface for nuclear power plant and safety analyses codes such as TRAC. An overview of the requirements and potential benefits of a TRAC GUI have been discussed in this paper. A graphical user interface developed by KAPL, TOOKUIL, has been described and the development process reviewed in terms of TOOKUIL’s requirements, architecture, key features, and implementation. TOOKUIL provides an integrated environment for engineers to build a TRAC input model, interface with computers in a client/server environment, and process and visualize analytic results. Stages of the development process include team formation, process/requirements analysis, conceptual design, prototyping, initial development, and advanced development. Key components in a successful GUI development effort include development of a detailed and comprehensive design which accurately addresses both short-term and long-term requirements, constant user review and feedback during all stages of the process, and proper project staffing, including both number and skill mix.

REFERENCES


DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.