Team-Based Collaboration in Model-Based Design

Saurabh Mahapatra[1] and Jason Ghidella[2]
MathWorks, Natick, MA, 01760

and

Gavin Walker[3]
MathWorks Ltd, Cambridge, UK, CB4 0HH

In this paper, a new tool is introduced that enables team-based collaboration in Model-Based Design. The use of this tool enhances the engineer's productivity by maintaining primary focus on the design tasks, encourages the adoption of configuration management tools through the abstraction of complex file and source control tasks, and improves knowledge transfer across the organization. The key challenges engineers face today and how this new tool addresses these challenges are described in this paper. Best practices for motivating organizations, large and small, to jumpstart team-based development initiatives are also described within the context of the tool's capabilities.

I. Introduction

Modeling and simulation has been the cornerstone for Model-Based Design in the aerospace industry [1; 2]. With advances in computing technologies, recent trends in aerospace modeling have focused on building higher fidelity models that result in better and more robust designs. The result has been the proliferation of large-scale projects that involve multidisciplinary groups of engineers that must collaborate to realize these models. Associated with these workflows are inherent complexities arising out of engineers having to organize, manage, and revision control files that contain algorithm implementations, data, utilities, and associated artifacts. Consequently, the complexity faced by the engineer is two-fold: design and file management. With attention divided between these two tasks, the productivity and effectiveness of the engineer is diminished.

At the team level, interdependencies occur as engineers contributing to a single design in a project-based setting complicate the matter further. The result has been a trend towards ad hoc project management where engineers have to learn to work with source control tools or depend heavily on a configuration management specialist within the team for basic tasks [3]. This can lead to process bottlenecks being created, or the abandonment of the process altogether. Lessons and best practices learned from one project are lost or not transferable to other projects.

In this paper, a design-centric approach, in which the file and project management tasks are exposed to the engineer from within the design tool is described to address the problem. By providing flexibility to connect the design tool to various source control tools via an authoring application program interface (API), the amount of the latter tool's exposure for common tasks engineers perform can be managed, while other critical project management tasks still remain with the configuration management specialist. Using Simulink® [4] as an exemplary environment for Model-Based Design, these issues are specifically addressed with a new tool called Simulink Projects.

II. New Approach to Team Collaboration

This section outlines the broad goals and architecture of Simulink Projects.

A. Broad Aims

The motivation behind the Simulink Projects tool was to provide an environment within Simulink to ease the engineer's transition from an individualized to a collaborative design environment. This required satisfying the following:

- Provide repeatability to specific file management tasks

[1] Saurabh Mahapatra, Simulink Platform Marketing, 3 Apple Hill Drive, Natick MA, AIAA Member.
[2] Jason Ghidella, Simulink Platform Marketing, 3 Apple Hill Drive, Natick, MA.
[3] Gavin Walker, Simulink Development, 3 Apple Hill Drive, Cambridge, UK.

- Improve accessibility to configuration management.

This would lead to the standardization of workflows among all the engineers working on the same project through the adoption of best practices such as component-based modeling, peer review workflow, and simplified configuration management.

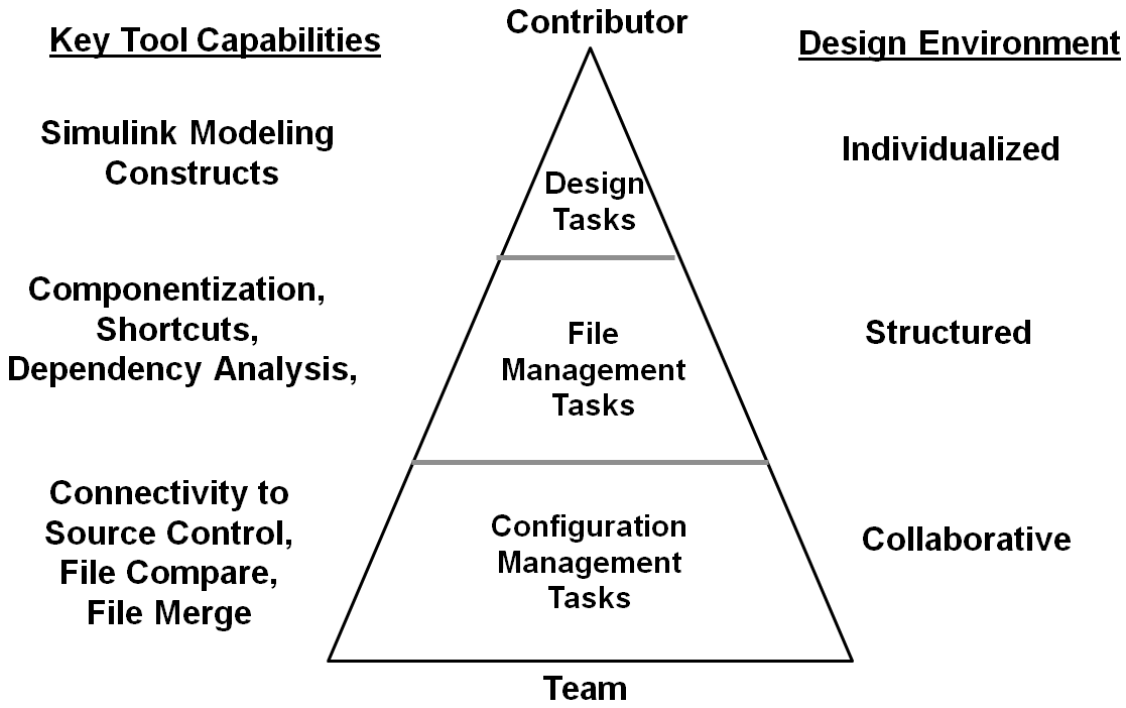

Figure 1. Mapping of project tasks and design environment characteristics for transitioning from an individual contributor to a team-based setting.

B. Tool Architecture

The tool design is modularized according to the design environment characteristics shown in Figure 1. The three levels described show the progression of the tasks complexity as the team size grows. In a highly individualized workflow, the engineer's focus is on their design and the design tool functionality itself. As the engineer's project scales, they will require a structured approach to file management such as creation of utilities, design componentization and folder organization. As multiple engineers work on the same project it necessitates revision control of files in a source control repository.

The tool provides engineers with an interface where they can define associations between files and folders in the project. By accessing standard views of the project, the engineer can streamline their file management tasks. Since this approach is design tool centric, there is flexibility to change the source control tool without disrupting the design workflow. To facilitate this, an application program interface, (API), and software development kit (SDK) were developed for authoring *adapters* that allow connectivity of Simulink Projects to a source control tool.

III. Addressing Key Challenges for Facilitating Team Collaboration

The primary use case for the tool was to enable design engineers familiar with Simulink but having minimal file and configuration management knowledge, to easily transition into a team-based environment. This section outlines the key challenges faced by engineers working in a team-based environment:

- Design partitioning
- Artifacts management
- Management and accessibility of project-level utilities
- Label management
- Accessibility to source control tools

- Workflow standardization
- Knowledge retention

Examples are presented to show how Simulink Projects can address these concerns.

A. Design Partitioning

Ad hoc approaches to partitioning often result in unwieldy models that are difficult to break down into smaller components. The component representation in the software tool itself is a key factor that determines the efficiency of the collaborative workflow. For example, Simulink models containing design components can be stored as a single file. However, such a monolithic approach can result in maintenance cost as all changes from all users would need to be merged into the same file. To alleviate these issues, design components can be represented in separate files that can be combined together to represent the overall design [5]. In Simulink, the modeling constructs that enable this are libraries and model referencing. Though similar, differences exist that impact their usage.

A library is simply a container for blocks. When libraries are used to hold design components, they typically contain subsystems. Subsystem libraries enable design reuse and centralization as when they are copied and placed in a design model, a link to the library is preserved allowing for updates of the design component to be propagated across all instances of that subsystem whether contained in one model or multiple models. There are some drawbacks to using libraries to contain design components, for example, since every instance of the subsystem library block is a copy of the component, the scalability of using them is limited. Components that are algorithm intensive and are used often in the model will consume system memory and will deteriorate editing and simulation performance significantly. Libraries are best used as a palette for lightweight components [6].

For algorithm-intensive components that are edited frequently and owned by a primary contributor, the use of model referencing is recommended [6]. Unlike libraries, model referencing forms a reference to a design component that is stored as a separate model file, the model is not copied. This significantly helps with the scalability of designs. Additionally, referenced models are context independent and require a strict definition on the I/O and software interface specification.

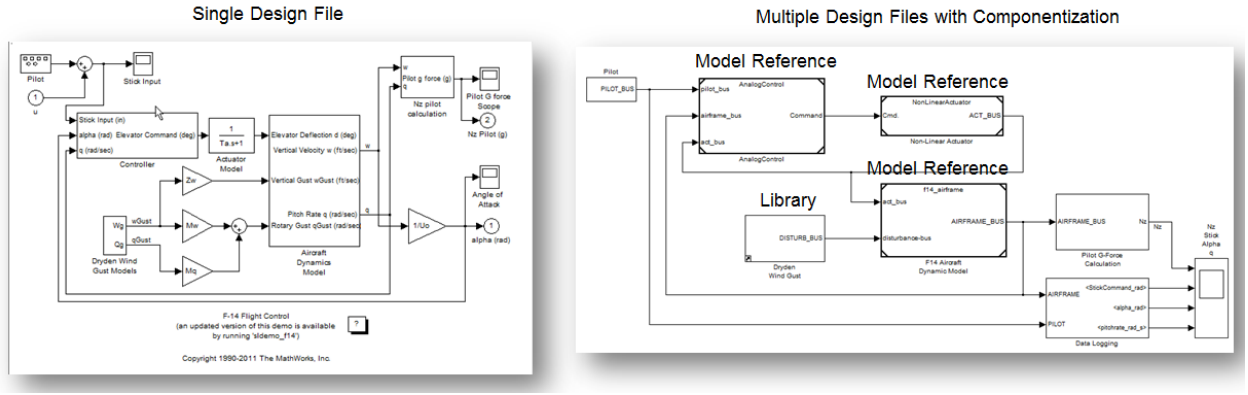

Figure 2. Monolithic design in a single file shown on left and it's componentized equivalent shown on right.

The use of these two modeling constructs in componentizing a model is shown in Figure 2. The wind model subsystem is stored in a library while the plant, actuator and controller components are referenced models. The creation of additional files necessitates file dependency analysis. Using the built-in Dependency Viewer in Simulink, it is possible to generate an architectural view of the design components.

The top level model *slproject_f14* shown in Figure 3 contains references to 3 additional models and a link to a library. The *f14_airframe* component model contains a reference to the *vertical_channel* model.

To define the interfaces between these components, *signal buses* provide a convenient mechanism to define the contained signals. Buses allow the definition of a full set of input and output signals, which is particularly useful when performing top-down design. Buses can be defined in a file and placed under source control, making it easy

to change the interface definition by modifying the file instead of making structural changes to the design model, such as adding or removing ports and signal lines.

It is helpful to discuss model architecture, criteria for componentization and file ownership early in the project [7]. This discussion can help establish a common understanding among team members. For example, a project charter that grants exclusive component ownership can require team members to respect component boundaries. Purposeful design is top down rather than bottom up.

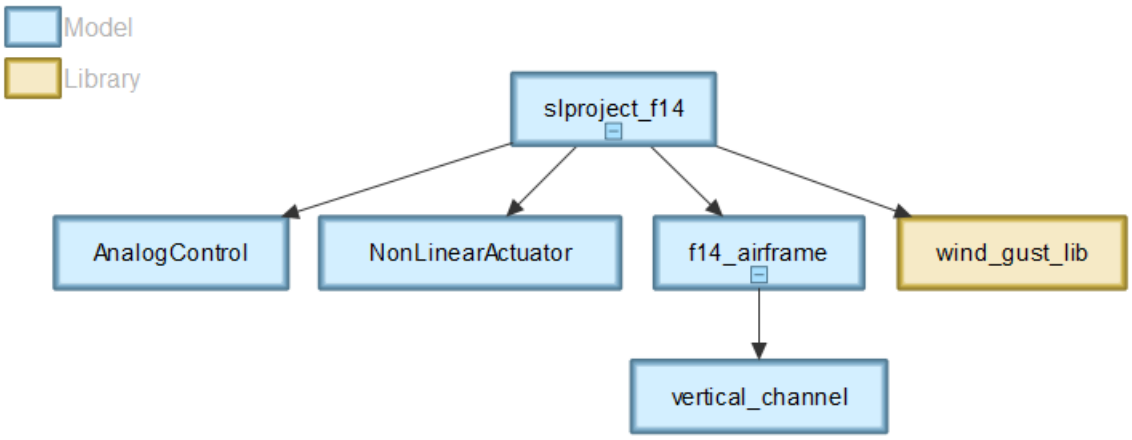

Figure 3. Dependency viewer for top level model *slproject_f14*

The following guidelines can be used to create components that enable team-based collaboration:

- Base the component boundaries on those of the real system – especially if the model contains both embedded systems (for control, signal processing, or communications) and physical systems (plant and environment).
- Define components distinctly so that only one engineer at a time needs to edit a component.
- Subdivide components that are too big and those that could become too big as the design is elaborated.
- Group components with the same sample rate.

These guidelines are a good starting point for defining component interfaces for a new project:

- Model all subsystems from the start. If your model will contain sensor models, include them as an empty subsystem that either passes signals straight through, adds a unit delay, or performs a name conversion.
- Keep component reuse in mind.
- Consider using a signal naming convention.

Engineers often store data files for specific components as part of a configuration. They frequently treat other data as global data that resides in the base MATLAB workspace. This practice can cause confusion during component integration because there is a risk that engineers will assign the same name to different parameters stored in configurations for two different components. Naming conventions for parameter files or a data dictionary of unique parameter names and definitions can help prevent confusion around storing local parameter data. Simulink provides two methods for preventing this confusion

- Data scoping through workspaces: Base, Model, and Mask
- Storage of parameters in separate files

B. Artifacts Management

Standards governing which artifacts to store in the repository at various stages of the development process can vary by project. These standards also depend on the working practices of individual organizations, companies, and industries.

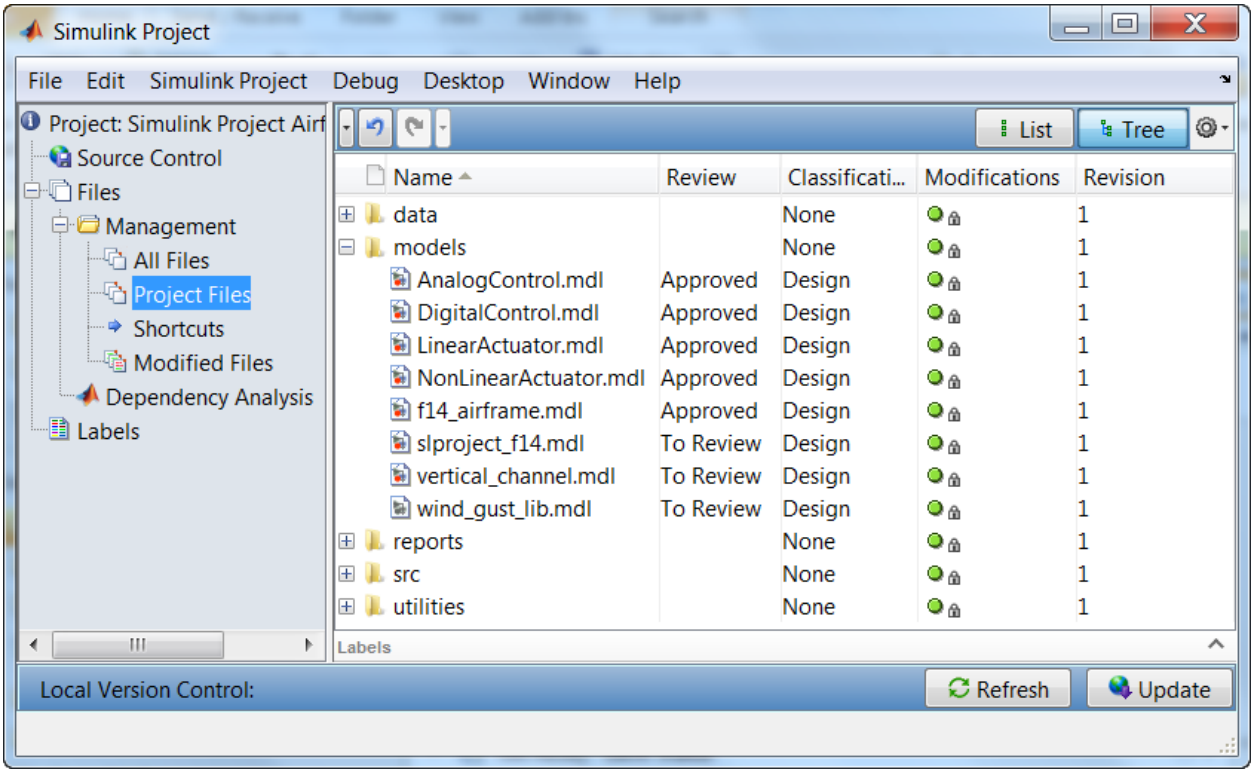

Figure 4. Simulink Projects user interface showing the componentized design Files with associated folder hierarchy.

A team might manage the following artifacts in the configuration management system:

- *Core functional files:* This artifact is the minimum set of files that represent a fully functional project. Keeping this core set of files under configuration management allows new team members to be able to open the project and be fully functional. The definition and hence number of files required to remain fully functional can change as the project progresses. For example, initially it might be sufficient for the project to simply simulate. Later, additional files or libraries may be needed to support code generation, or verification and validation activities.
- *Derived files:* To enable future reproducibility and root cause analysis, it is common for a team to also store derived files at specific points within the development process. Examples of derived files include the generated code, simulation and test results, model validation results, and linearized approximations to a nonlinear model.
- *Documentation:* Some teams opt to include artifacts relating to the development process. These artifacts can include model-checking reports that demonstrate review readiness, code generation logs, test reports, and model coverage reports.

It is also important to consider directory structure as an extension of componentization because it provides the abstraction for separating categories of files. The Simulink Projects user interface containing the design files and the folder structure is shown in Figure 4. The *data* folder contains the bus object definitions and model parameters, associated with the design. The *models* folder contains the componentized design files. The *reports* folder contains documentation artifacts generated from models. The *src* folder stores handwritten C code that is represented as components inside the design. The *utilities* folder contains scripts for initialization, and cleanup tasks.

C. Management and Accessibility of Project-Level Utilities

As the project evolves, knowledge is created by team members in the form of utilities such as automating project environment setup, report generation, regression testing or model standards checking. Accessing key files in a large scale project can be a daunting task. The lack of a unified collaborative environment increases the risk of losing this knowledge and impedes accessibility. Simulink Projects addresses these issues by providing a mechanism for users

to aggregate utilities and key files centrally in a special view called “Shortcuts”. It is also possible to assign special actions to certain shortcuts to execute at project startup or shutdown. Transitioning an existing project into this environment requires an audit of all utility and key files. For an ongoing project, these shortcuts, when created, are made available to all team members because this information is contained in the project definition files, which are stored in the source control repository.

Figure 5 shows that the interface definitions contained in the *buses.mat* file and the model parameters contained in the *f14_digital_data.mat* file are loaded at project startup. The *slproject_f14.mdl* file is a key file for launching the system level model. The project environment is setup at project startup by *set_up_project.m* file. This is cleaned up at project shutdown by *clean_up_project.m* file. The script file *rebuild_s*_functions.m builds manually authored blocks in MATLAB or C on the engineer’s machine.

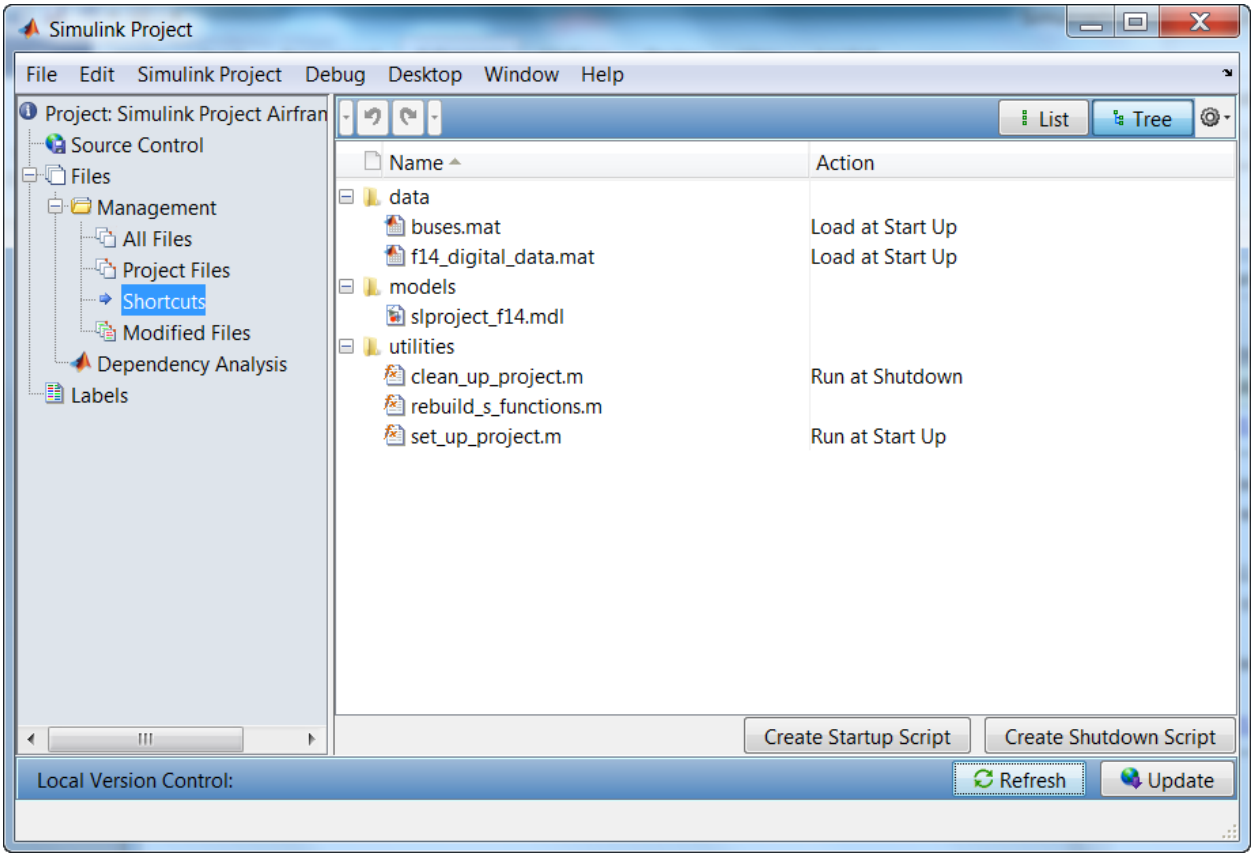

Figure 5. Key files and utilities accessible from the Shortcuts centralized view.

D. Label Management

Project teams typically need to store additional file information, depending on what task the models are being used for, the stage in the development cycle, and the organization's standards. For example, the team might include the following:

- Project status such as *under development*, *approved*, or *rejected*.
- Application classification such as *test harness*, *rapid prototyping*, or *production code*.
- Distribution classification such as *internal use only*, *suitable for export*, and *imported from customer*.
- Versions of products used to develop the files

These additional attributes can be identified by creating entity relationship models and converting them into a tabular form such as relational database where each entry record corresponds to a file. In Simulink Projects, this correlation is made possible through the use of “Labels” which are predefined tags that can be associated with a file. As shown in Figure 6**Error! Reference source not found.**, each user can see a tabular view of a desired set of

labels in column view. It is possible to define complex filters using regular expressions that can return names of files matching complex criteria.

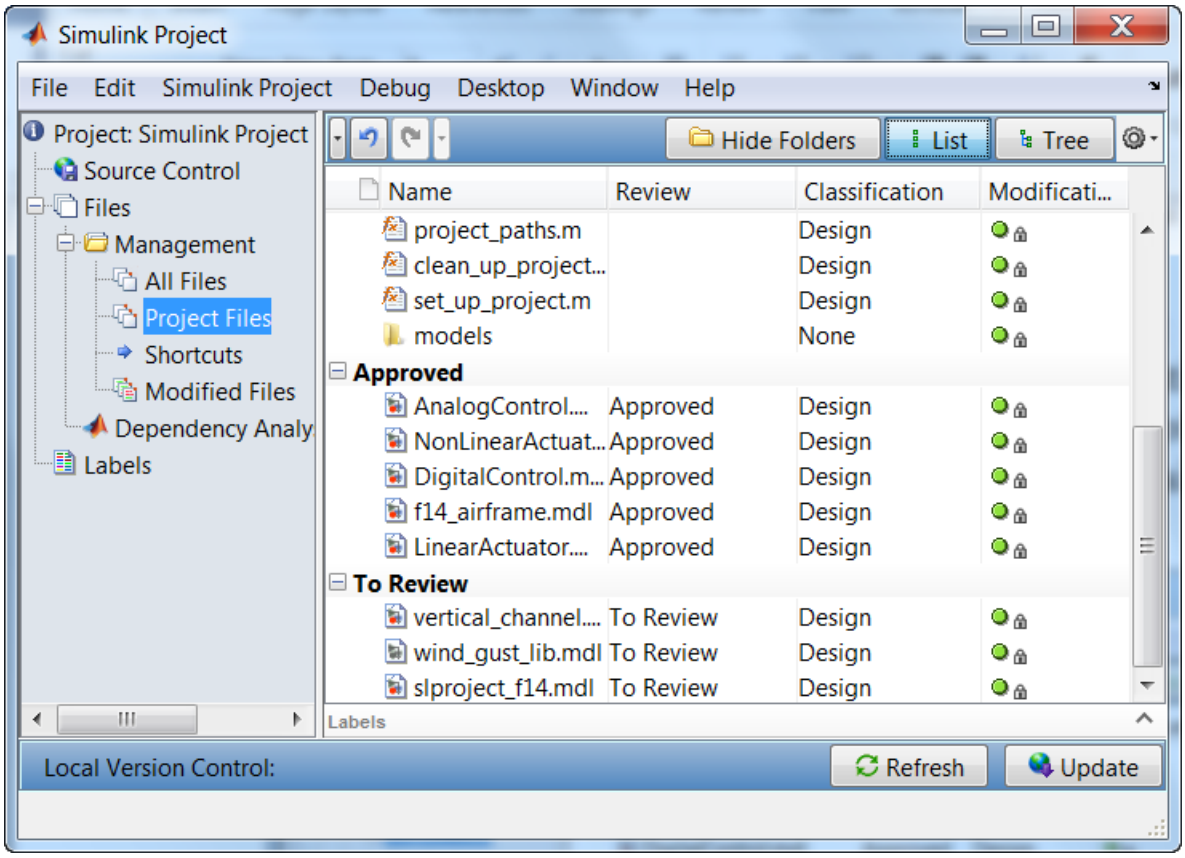

Figure 6. Simulink Project Labels shown as columns that have been grouped by review status.

E. Accessibility to Source Control Tools

In recent times, the adoption of source control tools has risen significantly due to the popularity of open source software projects such as Subversion® and Git. Based on anecdotal evidence gathered over several years, the authors would like to claim a trend towards Subversion as the preferred choice for groups working on a small budget or practicing lean methods. This popularity can be attributed to software engineers who have shared their positive experiences with their modeling and simulation engineers. Newer releases of Subversion are providing features comparable to commercial vendors. This trend offers several opportunities for teams to jumpstart collaboration. Workflows can be prototyped to gain early insight into its feasibility. To facilitate this, Simulink Projects offers out-of-the-box connectivity to Subversion. It also provides a Java-based software development kit (SDK) that can be used to create connectivity to other source control tools. Such modular tool architecture – where design and file management features can be decoupled from those of configuration management – allows the source control tool to be replaced with another without disrupting the existing workflow.

F. Workflow Standardization

Standardizing a workflow that consists of design, file and configuration management tasks across the entire team is critical for project success. In Figure 7, the *pre-submission review-based workflow* is highlighted. In this workflow, engineers ensure that files are not checked in without passing a set of regression tests and peer review.

Development work for the *pre-submission review-based workflow* involves creating a local copy of the project onto the design engineer's machine, checking out required files, modifying the design models, testing through simulation, and conducting peer reviews of the new changes. These changes are then submitted into the central repository where

regression tests and additional tasks such as automatic code generation and builds are performed by other team members.

In contrast, for the *committer-based workflow*, engineers check in files after conducting initial tests but leave the definition and testing of new configurations to a designated *committer*. In both processes, a primary objective is to prevent untested or problematic models from entering a working configuration, thereby enabling engineers to retrieve the latest configuration known to be good.

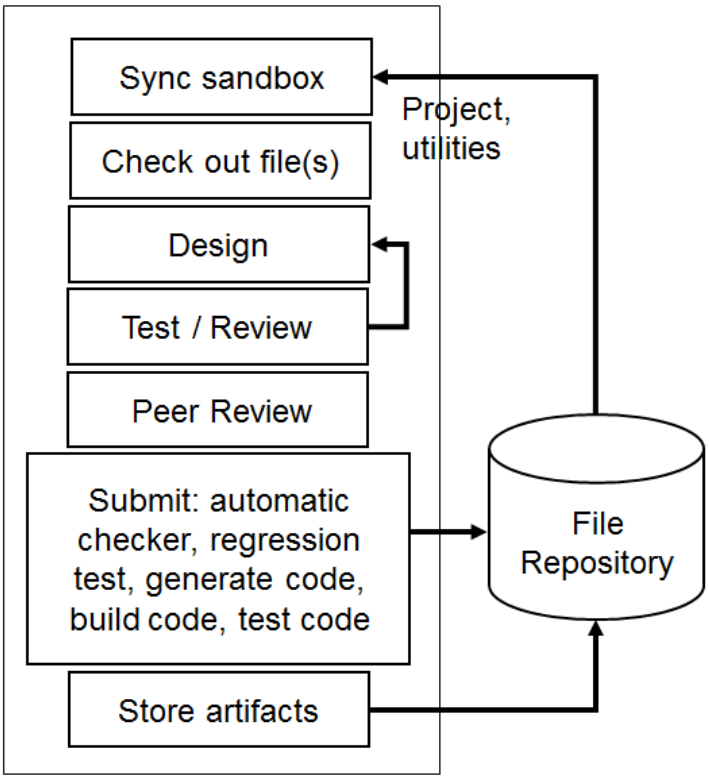

Figure 7. The Pre-submission Review-Based Workflow

In Simulink Projects, all the actions described above are made available to the user through file context menus or views. As shown in **Error! Reference source not found.**Figure 8, the "Modified Files" view shows all the project files awaiting check-in. They can be labeled for peer review, and also analyzed to detect whether there are no new files missing from the project.

It is important to mention that contractual rules for component ownership within your team should be established early in the process. There are two options: exclusive and mutual ownership. In exclusive ownership scenario, each component is worked on exclusively by only one user at a time. This ownership avoids the costs associated with merging changes from multiple users. However, exclusive ownership can slow parallel development as users needing to work on the same file, can only do so serially. While this limitation is removed in the mutual ownership scenario, mutual ownership incurs the cost of merging changes. Componentization and the selected ownership scenario chosen, requires understanding the tradeoff between development cost and the cost of merging changes.

As team size increases, establishing a collaborative framework that facilitates peer review is essential to the success of the project. The peer review consists of various activities such as:

- Comparing different versions of models and resolving conflicts.
- Generating a system design description (SDD) report.
- Using web-based renditions of the models to show system architecture.

Simulink Report Generator® [8] provides functionality in support of these activities. The tool provides the ability to compare XML text files exported from Simulink models. This comparison not only provides a comparison of the blocks within the model, but also the underlying configuration settings such as the type of solver being used, the data logging settings, and code generation settings. Simulink Report Generator also provides the ability to create the SDD report for any model [9]. The SDD report contains information on the model such as I/O signals, the associated data types, and dimensions. Web-based renditions of the model allow the model to be shared with others, such as management or adjacent groups, who need to be aware of the overall design but do not need to run the simulation.

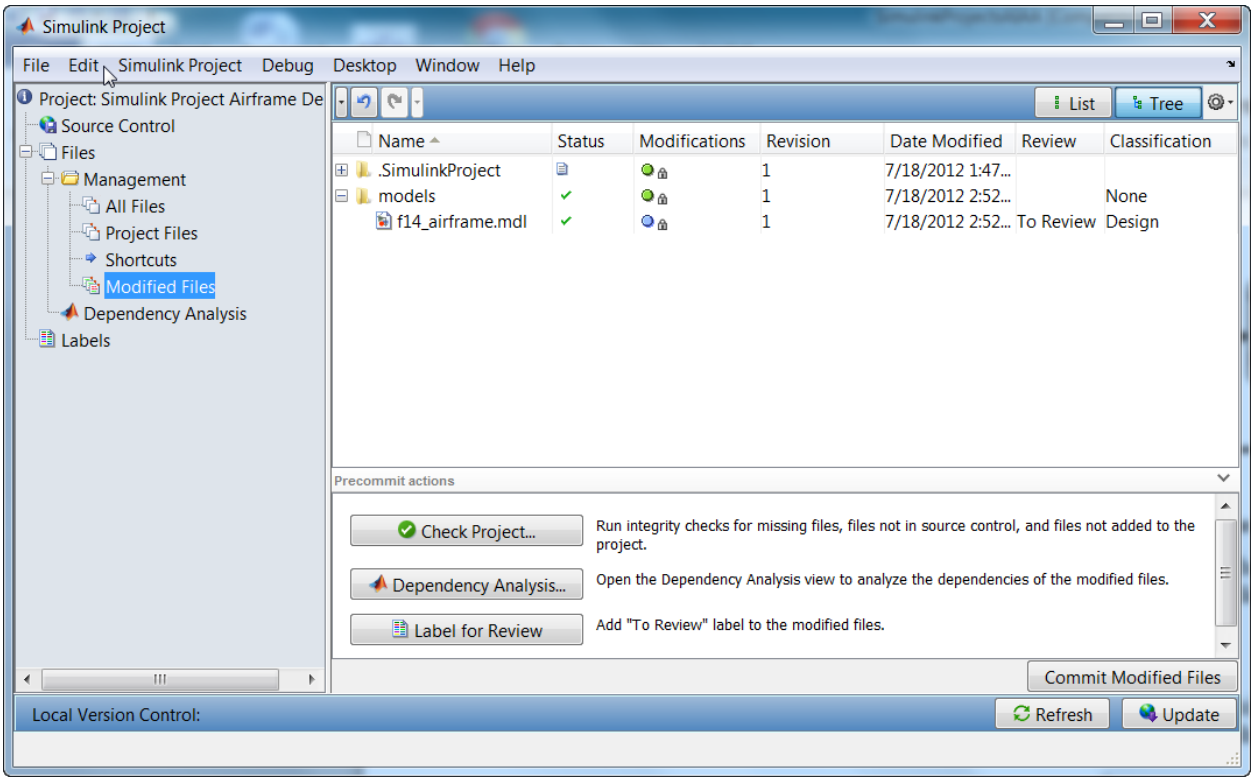

Figure 8. In the Modified Files view, files can be labeled for review, compared to earlier revisions, checked for project integrity and file dependency analysis prior to check in.

To create a release using Subversion, there is a facility to tag where the configuration of the project is copied over to the *tags* directory. This is very useful for creating releases and storing stable configurations that can be rolled back in the future.

G. Knowledge Retention

There are several aspects of a project that may be transferable to other projects inside the organization. This transfer results in standardization of several aspects of the development process that helps build organizational memory. Opportunities abound in this area such as reusing design architecture, utilities, libraries, folder structure, and legacy code. The Template Manager in Simulink Projects provides the ability to use "Templates" for adding and sharing project elements within your organization. For example, the folder structure and the associated utilities of a given project could serve as a standard for creating other similar projects elsewhere in the organization. To do this, the project definition and the associated files can be packaged using the Template Manager and stored centrally in a network location. The same tool can be used by any project initiator inside the company to use the same template as a starting point for further development.

IV. Conclusion

The ideas encapsulated in the Simulink Projects tool and presented in this paper offer many possibilities for organizations using Model-Based Design. By structuring the design and file management process, compliance with a standardized workflow can be achieved, leading to greater efficiency and effectiveness in a collaborative setting. Organizational memory of best practices can be increased through knowledge retention. Accessibility to

popular source control tools such as Subversion will help accelerate development for some groups directly, or be used to prototype workflows. Alternatively, the Adapter Software Development Kit (SDK) offers immense potential for vendors to create connectivity to their tools from Simulink.

References

[1]Barnard, P, *Graphical Techniques for Aircraft Dynamic Model Development,* AIAA Modeling and Simulation Technologies Conference and Exhibit, 2004, Rhode Island.

[2]Aberg, R and Gage, S, *Strategy for Successful Enterprise-Wide Modeling and Simulation with COTS software.* AIAA Modeling and Simulation Technologies Conference and Exhibit, August 2004, Rhode Island.

[3]Walker, G, Friedman, J and Aberg, R, *Configuration Management of the Model-Based Design Process,* Society of Automotive Engineers (SAE) World Congress, 2007, Detroit.

[4]MathWorks. *Simulink User's Guide.* MathWorks, 2012, Natick.

[5]Anthony, M and Friedman, J, *Model-Based Design for Large Safety-Critical Systems: A Discussion Regarding Model Architecture,* AUVSI's Unmanned Systems North America, 2008, San Diego.

[6]Grand, K, et al.,*. Large-Scale Modeling for Embedded Applications,* Society of Automotive Engineers (SAE) World Congress & Exhibition, 2010.

[7]Ruff, R, Stephens, C and Mahapatra, S, *Applying Model-Based Design to Large-Scale Systems Development: Modeling, Simulation, Test, & Deployment,* American Institute of Aeronautics & Astronautics (AIAA) Modeling and Simulation Conference, 2012, Minneapolis

[8]MathWorks. *Simulink Report Generator User's Guide.* MathWorks, June 2012. Natick.

[9]Mahapatra, S, *Automatic Report Generation in Model-Based Design,* Society of Automotive Engineers (SAE) Commercial Vehicle Engineering Congress, 2010, Chicago.