Methodology and Demonstration of Git-Based Configuration Control in Probabilistic Risk Assessment

Mostafa Hamza[a], Alp Tezbasaran[a], and Mihai A. Diaconeasa[a]
[a] North Carolina State University, Raleigh, USA

mmhamza@ncsu.edu, *alptezbasaran@ncsu.edu*, *madiacon@ncsu.edu*

Abstract: Probabilistic Risk Assessment (PRA) has been an integral part of large-scale, high-risk industries, like the aerospace, chemical, and nuclear industries. Both fault tree (FT) and event tree (ET) modeling techniques are essential parts of any PRA model. FTs are used to model the possible causes of system failures, whereas ETs are developed to track the progression of postulated initiating events and their associated system responses. FTs and ETs can be built, modified, updated, and tracked for simple systems and associated event progressions with minimal effort. However, as the complexity of the system or event progression increases, their associated models become more complex. For models developed, modified, and updated by a single analyst, increasing complexity does not necessarily present a challenge for model development. However, if multiple analysts contribute to developing the same PRA model, keeping track of the changes becomes critical to avoid conflicting or replicating work. In addition, rigorous configuration control is imperative to keeping track of PRA model modifications during iterative design stages in which system configurations evolve dynamically. Under current configuration control requirements, PRA standards in the nuclear industry, including ASME/ANS Probabilistic Risk Assessment Standard for Advanced Non-Light Water Reactor Nuclear Power Plants, require an explicit process for monitoring design changes, tracking associated model modifications, and tracing model progression. Though some existing PRA tools, such as CAFTA, implement minimal version control, multi-user collaboration and model-tracking capabilities are limited. To address this unmet need, we present a methodology that utilizes Git to fulfill the configuration control requirements and enable preliminary multi-user collaboration. We then demonstrate the presented methodology on representative models using the MAR-D file format for SAPHIRE and CAFTA. Finally, we offer a brief discussion of the limitations within the proposed methodology and propose improvements for future work.

1. INTRODUCTION

Probabilistic risk assessment (PRA) was first introduced within the reactor safety study (WASH-1400) in the 1970s [1]. The associated results introduced in WASH-1400 were highly debated and questioned by the nuclear industry [2]. However, the validity of PRA as an important tool to investigate the safety of nuclear power plants (NPPs) was proven after Three-Mile Island (TMI). During TMI, a small break loss of coolant accident (S-LOCA) resulted in the meltdown of the core proving the expectations of the WASH-1400 study [3].

Ever since TMI, PRA modeling became an integral part of the nuclear industry. Different regulatory documents list the expectations of PRA conducted by different NPPs [4][5]. Not only existing NPPs, but new NPPs are also utilizing PRA as one of their toolsets to prove the safety case of their design. More importantly, with risk-informed designs becoming more dominant in future NPPs, PRA modeling becomes increasingly important and integral to the design and licensing of NPPs [5][6].

While building PRA models, multiple tools are used to answer the three main questions: what could go wrong, how likely is it, and what are the consequences? [7] To assess the likelihood of a scenario, fault trees (FT) and event trees (ET) are used. FTs are detective logic diagrams that are used to trace the possible failure causes of a system, whereas ETs are logic diagram that are developed to track the response of a system to a postulated initiating event [8]. By combining both FTs and ETs analyses, the first two questions of any PRA can be answered [7].

For basic systems, building FTs associated with system failures or ETs associated with system responses is a relatively straightforward task. However, as systems get more complex, the associated FTs and ETs get lengthier and more complex. Within a nuclear power plant, FTs of a single system and ETs of a single initiating event could span multiple pages. This does not introduce an issue in viewing, tracing, or understanding FTs or ETs, however, modifying, updating, and tracking changes becomes less and less attainable by increasing model complexity. Tracking changes is especially important for models that are built by multiple analysts to avoid duplicating or conflicting work. In addition, keeping track of model changes during different stages of design is required under the configuration control requirement of PRA standards [8].

Moreover, with large PRA models, the collaboration capabilities in building, modifying, and updating the PRA model become increasingly limited. Although collaboration in building the PRA model might not be considered essential, the modeling of supporting systems that support/impact multiple frontline systems can be faster by parallelizing the modeling of separate support lines. More importantly, modifying and updating PRA models, during different iterations of the design, require a high degree of documentation and collaboration capabilities. Although the current PRA tools provide limited configuration control capability, collaboration capabilities in these tools are quite limited.

While for light water reactor (LWR) designs the nuclear industry can use PRA to analyze and risk-inform their design process before asking U.S. Nuclear Regulatory Commission (NRC) to certify them, for non-LWRs, a PRA model is expected to be required during the NRC licensing stages. Using the ASME/ANS PRA standard for advanced non-LWRs [8] as an example, section 1.1 discusses the expected level of configuration control that is required of any PRA model. It should be noted that configuration control is also needed from the PRA of LWRs; the non-LWRs requirement is only used as an example of regulatory requirements. Furthermore, section 1.2 discusses the current configuration control of Idaho National Lab's (INL) PRA tool SAPHIRE. Moreover, section 1.3 presents a brief description of the version control system Git that is utilized as the base of the proposed framework. It should be noted that Git is not the only available version control system, however, it is the most widely used system in software development. Finally, section 1.4 describes the MAR-D that is utilized in developing the current methodology.

1.1. Regulatory Requirements

One of the motivations for the current work presented in this paper is to provide a rigorous, easy, and complete track record of the PRA model. This will provide a pathway to support the regulatory requirement of configuration control. During the licensing stage of any nuclear power plant (NPP), a final safety analysis report (FSAR) is presented to the regulatory body. In this FSAR, the results of a full PRA analysis are presented which are expected to follow the standards for PRA.

The objectives of the configuration control program, according to the ASME/ANS PRA standard for advanced non-LWR [8], are to ensure a process of monitoring changes to the plant design, upgrading the PRA model to be consistent with the latest plant design, documenting the impact of changes to the design on the risk profile, and tracing the computer codes and files supporting the PRA. To support these high-level requirements, the configuration control program is expected to include a process for tracking changes in design, operation, maintenance, facilities, industry experience, and PRA technology. It is also expected to document assumptions, changes, upgrades, and industry issues that may impact the PRA model. Finally, it is expected to "ensure that computer codes and associated files [...] are controlled to ensure consistent and reproducible results" [8].

Although the standard indicates that "the configuration control technical element is likely not achievable at" the conceptual design phase, a rigorous documentation process of each element of a PRA model is expected to provide traceability and support the configuration control. Moreover, as discussed previously, design stages require a high level of configuration control to risk-inform the design during different iterations of the design. This, combined with the need for collaboration capabilities during model development, showcase the importance of having a rigorous tracking and collaborating process even during the design stages.

1.2. SAPHIRE's Configuration Control

System analysis programs for hands-on integrated reliability evaluations (SAPHIRE) is a PRA tool developed by INL with funding from the U.S. nuclear regulatory committee (NRC). With version 8.70 being the latest working revision [9], SAPHIRE enables users to create and solve FTs and ETs, perform uncertainty analysis, input basic events data, and generate reports [10]. Despite having gone through multiple revisions, SAPHIRE still does not provide adequate collaboration capabilities. Moreover, despite being a regulatory requirement, SAPHIRE provides minimal configuration control capabilities for PRA models.

Although SAPHIRE gives the capabilities to generate different reports, these reports are mainly cut sets, FTs or ETs' descriptions, or other results, with no reports regarding changes to the model during different iterations. Moreover, version 8.x of SAPHIRE included the capability of integrating two projects into a single project, which allows the user to compare two different projects [10][11]. However, this requires users to work on two different projects rather than one master project. Moreover, the interface only shows new events, identical events, and changed events, however, the interface does not show if some events were deleted, nor does it show what changes occurred in a specific changed event.

1.3. Version Control Systems and Git

A tool that manages and tracks different versions of software or other content is referred to generically as a version control system (VCS). Standard terminology also refers to these tools as Source Code Manager, Revision Control System, and other permutations with the keywords revision, version, code, source, control, management, system, and framework. VCS history goes back to the early 1970s, starting with The Source Code Control System developed for Unix distributions [12]. Over the years, the techniques matured, also applied to many software. The drastic change over the years was moving from a centralized approach to a distributed one. How the files are locked with versions or changes also improved critically.

Nowadays, it is the standard to learn one of the tools alongside learning a computer coding language for the first time. There are four historical categories of VCS. These are distributed, centralized, lock-based, and optimistic VCS. VCSs are developed initially to track changes in a software code, with transparent communication among team members. The state-of-the-art standard for version control is the distributed VCS, where multiple team members can reach a repository from different locations. This study proposes an alternate use for VCS in general. The idea is to version control PRA models and track changes, allowing the development of large models in a shorter time.

There are multiple industry-standard VCS tools, namely, Git, Subversion, Monotone, BitKeeper, and Mercurial. This study uses Git [13] to demonstrate the proposed methodology since it is the most popular option and has advantages in the PRA model development context. Git was born amidst the Linux kernel community and BitKeeper issues and released in April 2005. The name "Git" is referred to as Global Information Tracker, but it is not an abbreviation but simply a practical joke of the original Linux kernel developer Linus Torvalds. It is a well-engineered VCS with many advantages such as distributed development, scaling for multi-user development, speed, immutability, and accountability. These improved features of Git make it a suitable tool for PRA model configuration control. Since PRA model files can be enormous for a single person to manage history and track changes, using tools such as Git is only natural. The needs are similar to that of source code development where there is a need for storing the history of all changes and the ability to revert and work as a team transparently [12].

Branching capabilities of Git allow distributed development where model developers do not necessarily have to work on the same computer. The structure can easily handle thousands of simultaneous development branches so that many engineers can work on various parts of the model simultaneously. Furthermore, the exchanges are fast, and all of the histories of the source code (or PRA model) are stored locally and can be visited or used if necessary. Moreover, the hashing feature of Git promotes and enforces integrity since all the files associated with a project (generally referred to as a repository) are subjected to a hashing function. The outcome of hashing is a unique value that changes even with the slightest modifications in the files.

Git uses this technique to speed up upload/download processes and ensure file integrity via recording every change. Accountability is another essential feature that Git employs. All the changes are associated with whoever made them in their respective branches, making the development transparent among team members. With branching, merging is also introduced which is bringing changes of two branches together. Transparent workflow of Git support clean, fast, and easy merging.

1.4. Models and Results Database (MAR-D)

In this analysis, the NRC-funded and INL-developed models and results database (MAR-D) is utilized. MAR-D presents a way of converting PRA models into a manageable text-based format. Developed to support the legacy version of SAPHIRE, MAR-D allows users to load, store, and extract all the information associated with the PRA model [14]. More importantly, being a text-based, MAR-D gives the ability to store, modify, update, compare, trace, and collaborate in PRA models using the Git system. Moreover, having a legacy structure, MAR-D opens the path to cross-compatibility in multiple PRA tools including EPRI's CAFTA and ABS Consulting's RISKMAN. However, further investigation is required to achieve this, which is out of the scope of the current paper.

Although the use of Git to provide PRA version control was presented and discussed before by INL in 2021 [15], the report fails to acknowledge that PRA models use non-text-based files. Hence, despite providing the capability of storing and tracking different iterations of the PRA models, the proposed methodology given in [15] will require the users and reviewers to run the models locally to track the changes. In other words, the methodology presented in [15] indicates and highlights when a new model is added but does not give a quick way of identifying and comparing whether this model changed and what are these changes.

Hence, the purpose of this paper is to present a methodology for storing, updating, modifying, and tracking PRA models using Git and MAR-D. The proposed methodology presents a way of achieving configuration control that is like the one proposed by [15], however, with the capability of quickly identifying and comparing changes in different iterations of the design. Moreover, the proposed methodology presents a pathway for collaboration in building PRA models. The proposed methodology is tested and showcased using a simple SAPHIRE PRA model. Finally, the limitations of the proposed methodology are identified and the paths for resolving these limitations are presented.

The remaining of this paper is arranged as follows: Section 2 presents the methodology used to utilize Git as a configuration control hub. Whereas section 3 showcases the proposed methodology using a rudimentary FT/ET model developed in SAPHIRE. Furthermore, section 3 also presents the limitations of the proposed methodology and their associated path of resolution. Finally, section 4 gives a summary of the ideas presented in the paper and presents the final concluding remarks.

2. METHODOLOGY

In this section, we introduce the high-level methodology for PRA version controlling via Git or Git-based tools. The workflow is shown in Figure 1, as proposed by [15], which shows the generic Git usage applied to source code development, version control, and PRA model development. The first step of Git-based configuration control includes recognizing an issue. The plant model might be missing some components, or basic event probabilities might need an update. This issue can also be a needed improvement, or in our case, it can be a needed addition to the PRA model or different iteration of the design. After recognizing the issue, the next step is branching from the primary model so that modelers can make changes independently on their branches. The next step is to work on the models, make changes, and commit them. Once the changes are complete and the branch is mature enough to be included in the primary model, a merge of two branches is requested.

The merge request triggers the sanity checks to evaluate if the changes are valid and working as intended without breaking any other model part. In the PRA model configuration control context, the analysis code should be running without problems after changes and represent the latest changes in the design. After verifying the changes, a peer, who did not work on the branch, approves the merge request provided that all checks are successful, and the new model is working as intended. The last two steps

can be automated based on the nature of the models and the simulation tools. The final merge changes the relevant parts of the primary model.

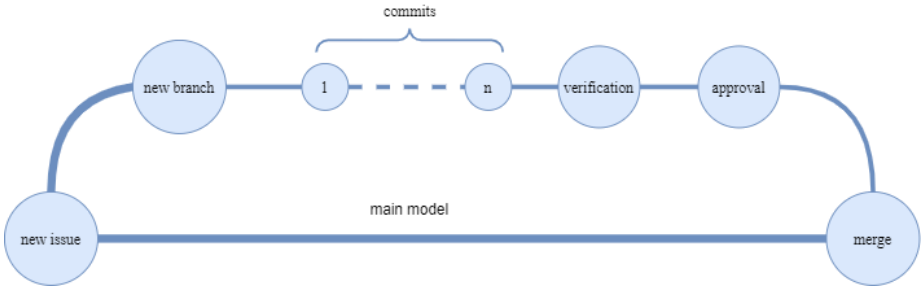

Figure 1 - Generic Git-based configuration/version control [15]

Note that up to this point, the high-level methodology described can be applied to any text-based file or files (source code or input files alike). We can also apply this methodology partially, even if the PRA models are saved and stored in the modeling tool's native file format. If the stored files are not text-based, it is still possible to store, distribute, and version control the models; however, changes cannot be seen or investigated without downloading and importing the model. Using non-text-based model files brings additional excruciating steps toward PRA model development, as [15] suggests. To use all the features provided by Git, the model developers must work with text-based files. Most modern PRA software can import and export the models in multiple formats, including text-based files and native extensions. So, the proposed methodology starts with exporting the model to text format first. It is also worth mentioning that the working directory should be configured as a Git directory; since the relevant configuration files prevent unnecessary ones in the repository and allow exchanging only the text-based PRA model files.

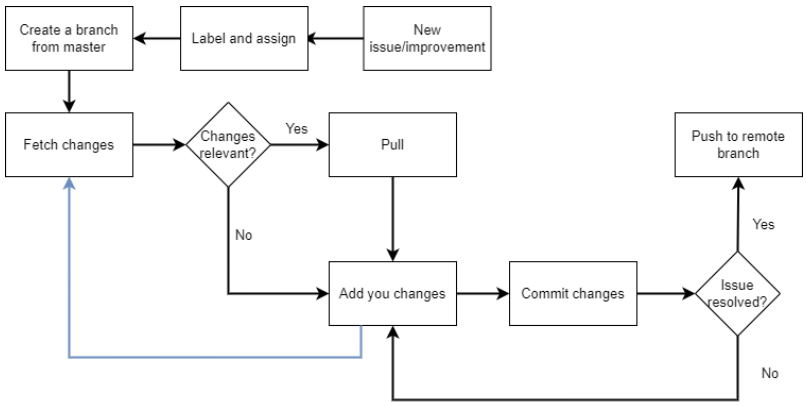

Figure 2 - Individual modeler workflow

Figure 2 shows the basic workflow for an individual modeler. It is worth mentioning that despite the changes being done with analysis tools, the file exchange with the Git repo should be done only with text-based files. The step to focus on, in this straightforward workflow, is to keep up to date with others' changes. If they are relevant and must be included in the current changes, a proper strategy must be followed. Merge between branches or checking out some files from another working branch might be needed. Another thing to consider is not to move relevant files manually since, as mentioned previously, relevant configuration files in the repository will only allow relevant model files to be included, to avoid including any unnecessary files.

Furthermore, an engineer who models and uses version control based on the framework shown in Figure 2 is only part of a larger team. There is also an essential role of the repository maintainer. The workflow for this role is shown in Figure 3. The primary responsibilities are checking the merge or pull requests, triggering the necessary sanity checks, and either approving changes and merging the branch with the primary model or denying the changes and asking for resolutions based on the metrics decided. The metrics can include proper, error-free execution of the modeling tool with the new changes and

parameter values that are expected or calculated before that are not subject to new changes. The metrics are based on the model, and engineering judgment is required.

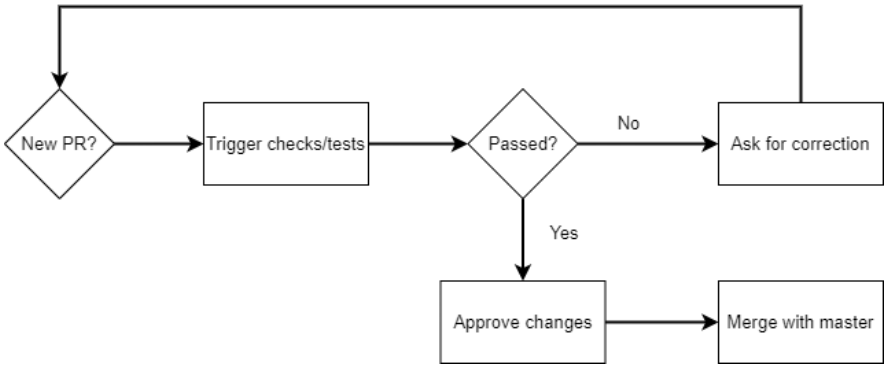

Figure 3 - Repository maintainer workflow

For example, quantification of an unrelated fault tree in the primary model must yield the same result. It is also possible to create tests based on the model files. If some of the model files or parts of those files must remain unchanged, that can be tested. Checks and approval are not necessarily time-consuming if an automated structure is adopted when initializing the model repository. When there is a new pull request, the pipeline can trigger code executions with the model from the corresponding branch and report the results based on the metrics defined. It is, however, a common practice to protect the main branch that includes the primary model (or source code for code development), so the final approval is through a human decision. This way, breaking the model is prevented in cases where the tests and metrics cannot comprehend the changes in the model. In the following section, we demonstrated the methodology introduced and discussed here.

3. DEMONSTRATION AND DISCUSSION

For demonstrating the methodology, a case study was conducted using SAPHIRE and Gitlab. An event tree for medium-break loss of coolant accidents (LOCA) reported in [16] was built using SAPHIRE. This model, shown in Figure 4, was then exported into MAR-D and pushed to Gitlab to initialize the repository. The initial model used only the default values for the top events; hence, two issues were created to update the failure probability associated with each top event and to build the fault tree associated with the reactor trip system.

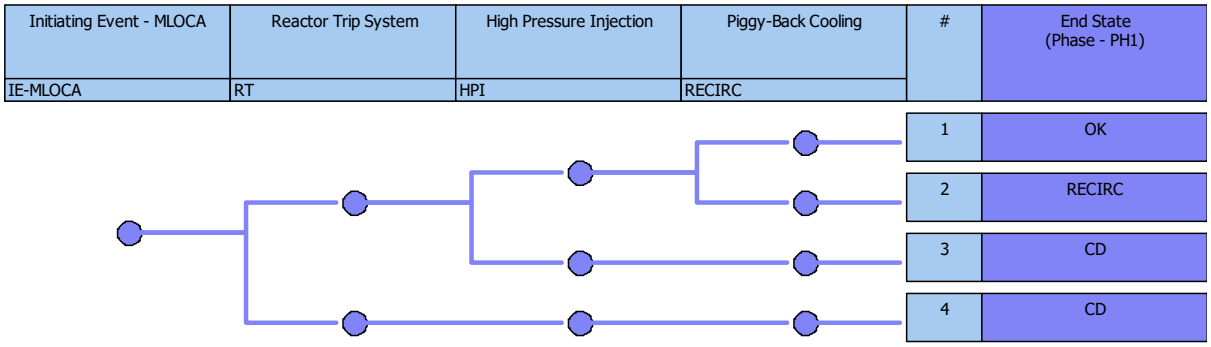

Figure 4 Event Tree for Medium-Break LOCAs at Oconee reproduced from [16]

To update the top event failure probabilities, the fictitious failure probabilities listed in Table 1 and Table 2 were used. Moreover, the simplified FT associated with the reactor trip system is shown in Figure 5. A simplified model was used to demonstrate the methodology; hence, care was taken not to use a detailed model that could cloud the demonstration. However, the methodology, approach, and implementation can be easily scaled up to accommodate any larger model.

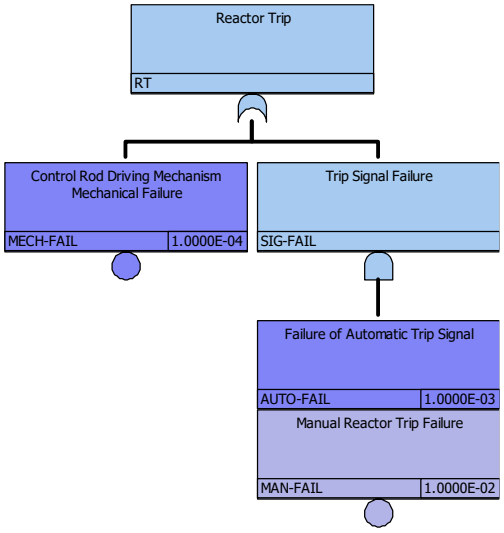

Figure 5 Simplified Reactor Trip Fault Tree

Table 1 Top Event Failure Probabilities

Top Event	Old Failure Probability	New Failure Probability
High Pressure Injection	1.00E+00	1.00E-04
Piggy-back Cooling	1.00E+00	1.00E-03

Table 2 Basic Events Failure Probabilities

Top Event	Old Failure Probability	New Failure Probability
Control Rod Driving Mechanism Failure	1.00E+00	1.00E-04
Failure of Automatic Trip Signal	1.00E+00	1.00E-03
Failure of Manual Trip Signal	1.00E+00	1.00E-02

As can be seen in Figure 6, the two issues created two branches from the project initialization. The contents of these two branches were then pulled locally where the necessary adjustments to the model are made using SAPHIRE. It should be noted that, though SAPHIRE was used in this demonstration, this approach is valid for any PRA tool that exports/reads MAR-D files or any other text-based files. Being a legacy database developed for PRA documentation, MAR-D presented a good basis for demonstration.

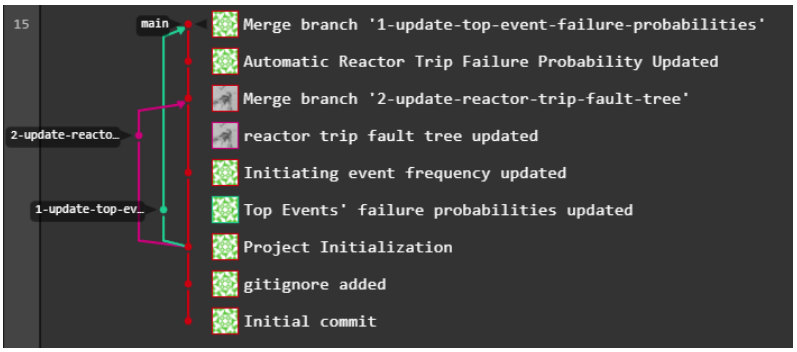

Figure 6 Commit, Push, and Merge History

Moreover, it should be noted that, during the merging of both branches, conflicts arose due to discrepancies between the content of each branch. For example, the main and second branches, shown in red and pink respectively in Figure 6, have different values for the initiating event frequencies. These merge conflicts prevent the user from overwriting previous work; hence, they require input from the user for deciding which changes to keep in the main branch.

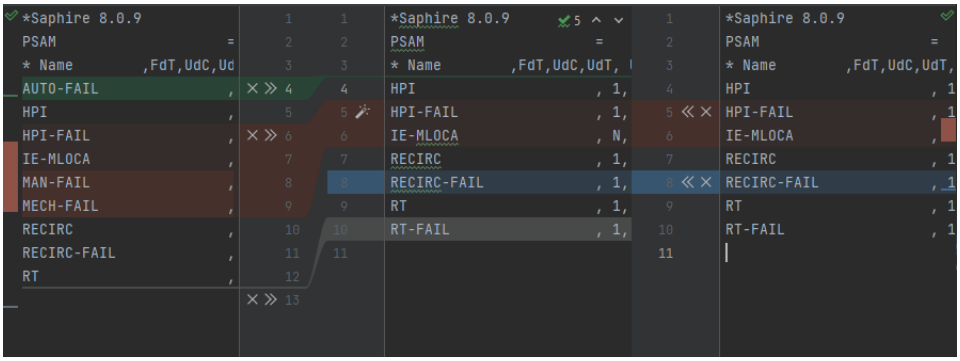

Figure 7 Merge Conflict Resolution

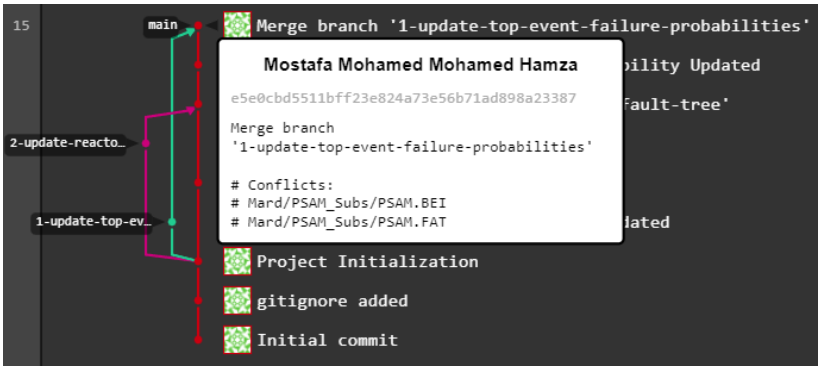

Figure 8 Issue, Commit, Push, and Conflict History in Git

Resolving merge conflict can be done easily using an integrated development environment (IDE). Using the MAR-D allows for merging specific changes from each branch, as can be seen in Figure 7. Each conflict shows as highlighted line, and the user should only merge, or overwrite if needed, only their own changes. By doing this, collaboration within the same FT, ET, and project becomes feasible, which achieves the first goal of this paper.

Moreover, Git provides the capability of tracking all commits, changes, and conflicts, as can be seen in Figure 6 and Figure 8. This provides a high level of traceability of work which enables reviewers to track the state of the model through different phases. By having the entire history of the model recorded, users can return to any previous stages of design to compare or undo changes. Moreover, this record of model changes facilitates meeting the requirements of configuration control for the PRA model which achieves the second goal of this paper.

Finally, it should be noted that it is a best practice that all changes should be made on separate branches. The main/master branch should be only updated in milestones and with the agreement of all responsible parties. This could be translated into each phase of the PRA model having its own separate branch and all modifications only conducted on this specific branch. However, for the purposes of this paper direct merges with the main branch are used. Moreover, as can be clearly seen, keeping a consistent nomenclature for basic events, FTs, and ETs is essential for this task to allow for easier and quicker identification and comparison; such a consistent nomenclature is expected for any professional PRA model.

4. CONCLUSION

The ASME/ANS PRA standard for advanced non-LWR lists the requirements expected from any PRA model associated with non-LWR. Among these, configuration control is one of the elements required by the PRA model. To meet this requirement, Vedros et. al. presented a methodology to use Git as a repository and a version control tool. Git presents multiple tools to store, modify, update, and track the status of the repository throughout its history. According to [15] “The goal would be to create a unique centralized location of the actual plant PRA model in the current plant information technology services

and to use GitLab/GitHub to interface the same model to plant personnel (e.g., PRA analysts, reactor operators, system engineers)".

Although Git presents a solid platform for storing and tracking documents throughout the history of the plant, current PRA tools provide very limited tracking capabilities and no collaboration capabilities whatsoever. Moreover, current PRA tools use binary files that can be stored on Git, however, Git capabilities for tracking changes in these files would be very limited. Hence, to support the goal of [15] in utilizing the capabilities of Git within PRA models, a methodology to use Git as a version control tool for PRA models is needed. Using the legacy MAR-D, a similar approach to that suggested by [15] is used to provide a complete, albeit rudimentary, version control tool for PRA models.

The MAR-D provides the means to store and track complete PRA models using text-based files. Combined with Git, the proposed methodology allows PRA model builders to store, update, modify, and track their models. Moreover, by keeping consistent nomenclature, the proposed methodology allows reviewers to track all changes throughout the history of the PRA model. This allows for easier peer review and quicker assessment of the impact of different changes on the overall model.

Moreover, by having the capabilities to export/read MAR-D files, any PRA tool, including SAPHIRE and CAFTA, allows users to collaborate in building their model. This reduces the time needed to build a PRA model while providing high confidence in any changes committed since all the model history is stored and can be restored if needed. This capability, however, requires care while merging different branches to incorporate only intentional changes while avoiding any discrepancies due to different branches.

Finally, this proposed methodology provides a straightforward solution to the issue of version control for PRA models. Better implementation of the proposed approach can be incorporated directly into the PRA tools which will provide the user with the capability to directly pull, modify, push, and merge any changes directly from the PRA tool. Such a capability is made easier by most PRA tools being able to read/export text-based files. However, until implementing such a feature into the PRA tools, the proposed methodology provides a user-driven solution to both version control and collaboration capabilities.

References

[1] Rasmussen, "Reactor safety study. An assessment of accident risks in U. S. commercial nuclear power plants. Executive summary: main report. [PWR and BWR]," Nuclear Regulatory Commission, Washington, D.C. (USA), WASH-1400-MR; NUREG-75/014-MR, Oct. 1975. doi: 10.2172/7134131.

[2] Office of Technology Assessment. (1984). Nuclear power in an Age of Uncertainty. Chapter 8 Public Attitudes Toward Nuclear Power, pp. 218–219.

[3] "Analysis of Three Mile Island-Unit 2 accident," Electric Power Research Institute, Palo Alto, Nuclear Safety Analysis Center, USA, EPRI-NSAC--80-1, Mar. 1980. [Online]. Available: https://inis.iaea.org/collection/NCLCollectionStore/_Public/13/677/13677904.pdf.

[4] NUREG/CR-2300, Vol.1, "PRA Procedures Guide: A Guide to the Performance of Probabilistic Risk Assessments for Nuclear Power Plants," U.S. Nuclear Regulatory Committee, USA.

[5] Regulatory Guide 1.200, Revision 2, "An Approach for Determining the Technical Adequacy of Probabilistic Risk Assessment Results for Risk-Informed Activities," U.S. Nuclear Regulatory Committee, USA.

[6] Delaney, M. J., Apostolakis, G. E., & Driscoll, M. J. (2005). Risk-informed design guidance for future reactor systems. Nuclear Engineering and Design, 235(14), 1537–1556. https://doi.org/10.1016/j.nucengdes.2005.01.004.

[7] S. Kaplan and B. J. Garrick, "On The Quantitative Definition of Risk," Risk Analysis, vol. 1, no. 1, pp. 11–27, 1981, doi: 10.1111/j.1539-6924.1981.tb01350.x.

[8] Probabilistic Risk Assessment Standard for Advanced Non-LWR Nuclear Power Plants (ASME/ANS RA-S-1.4-2021). (2021).

[9] SAPHIRE | SPAR Models. (n.d.). Retrieved April 10, 2022, from https://saphire.inl.gov/#/models

[10] NUREG/CR-7039, Vol. 1, "Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 8, Volume 1: Overview and Summary." (2011). https://doi.org/10.2172/130641
[11] NUREG/CR-7039, Vol. 3, "Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 8, Volume 3: User's Guide." (p. 206). (2011).
[12] Loeliger, J., & McCullough, M. (2012). Version control with Git (Second edition). O'Reilly.
[13] Chacon, S. (2014). Pro Git (Second edition). Apress.
[14] Branham-Haar, K. A., Dinneen, R. A., Russell, K. D., & Skinner, N. L. (1992). Models and Results Database (MAR-D), Version 4. 0 (NUREG/CR-5301, EGG--2627, 5299087; p. NUREG/CR-5301, EGG--2627, 5299087). https://doi.org/10.2172/5299087
[15] Vedros, K., Boring PhD, R., Knudsen, J., Lawrence, S., Mandelli, D., Park, J., Prescott, S., & Smith, C. (2021). Enhancement of Industry Legacy Probabilistic Risk Assessment Methods and Tools (INL/EXT-21-64448-Rev000, 1822882; p. INL/EXT-21-64448-Rev000, 1822882). https://doi.org/10.2172/1822882
[16] National Academy of Engineering, Kunreuther, H. C., Bier, V. M., Phimister, J. R., & National Academy of Engineering Staff. (2004). Accident Precursor Analysis and Management: Reducing Technological Risk Through Diligence. National Academies Press. http://ebookcentral.proquest.com/lib/ncsu/detail.action?docID=3377259