Next-Gen automation strategies in PSA model engineering and analysis

Stratégies d'automatisation de nouvelle génération dans l'ingénierie et l'analyse de modèles EPS

Friedlhuber Thomas
Edgemind
Paris, France
thomas.friedlhuber@edgemind.net

Sébastien Vermuse
EDF Edvance
Montrouge, France
sebastien.vermuse@edvance.fr

Gurvan Le Moigno
EdgeMind
Vannes, France
gurvan.lemoigno@edgemind.net

Abstract: Current risk assessment tools often lack concepts and functionality to assist users in automating tasks. A task in the wider sens is any activity related to model engineering or risk assessment: modeling, model validation (model consistency), quantification, report generation etc. Users perform tasks mainly manually which may be time consuming and error-prone.

This article presents an approach of automating certain recurrent tasks related to model engineering, qualification and quantification. While the focus is on (Internal Event) PSA models, the approach is equally applicable for other model types (such as functional sequence diagrams or specific hazard models).

Principal goals are to disburden safety analysts from tasks that can be automatically executed, to enable frequent and early feedback on model modifications, model consistency and quantification results and to ensure ongoing consistency between models and results.

Keywords: Automation, Probabilistic Safety Analysis, Model Engineering, Risk Analysis, Quality Assurance

Résumé: Les outils actuels d'évaluation de risques manquent souvent de concepts et de fonctionnalités pour aider les utilisateurs à automatiser les tâches. Une tâche au sens large est toute activité liée à l'ingénierie des modèles ou à l'évaluation des risques : modélisation, validation des modèles (cohérence des modèles), quantification, génération de rapports, etc. Les utilisateurs effectuent principalement les tâches manuellement, ce qui peut être chronophage et source d'erreurs.

Cet article présente une approche pour automatiser certaines tâches récurrentes liées à l'ingénierie des modèles, à la qualification et à la quantification. Bien que l'accent soit mis sur les modèles d'études probabilistes de sûreté (EPS) pour les événements internes, l'approche est également applicable à d'autres types de modèles (tels que les diagrammes de séquence fonctionnelle ou les modèles de risques spécifiques).

Les objectifs principaux sont de décharger les analystes de sécurité des tâches pouvant être exécutées automatiquement, de permettre un retour d'information fréquent et précoce sur les modifications des modèles, la cohérence des modèles et les résultats de quantification, et d'assurer une cohérence continue entre les modèles et les résultats.

Mots-clés: Automatisation, Analyse Probabiliste de Sûreté, Ingénierie des Modèles, Analyse des Risques, Assurance Qualité

I. INTRODUCTION

The creation or modification of the content of a PSA model is a recurring task in nuclear engineering, whether in the design phase (Conceptual Design, Basic Design, Detailed Design) of a new reactor or in the phase of periodic safety reassessment (for example, decennial reevaluation in France) of a reactor in operation. Moreover, the scope of PSA has greatly increased with the rise of level 2 PSA and the development of internal or external hazards PSA. It should be noted that these PSA models are usually interdependent: hazards PSA are developed from the Internal Events model using overlay techniques, a level 2 model is an extension of the level 1 model, etc.

The size of the team in charge can reach up to 10-20 people to create or modify the PSA of nuclear projects in order to take into account knowledge evolutions, material modifications and/or intellectual modifications related to changes in operating procedures. All these engineers do not have the same profile and skills: the number of PSA expert engineers is much less than that of PSA confirmed or PSA beginner engineers.

In fact, simultaneous, sometimes iterative work with inter-activity dependencies on the different bricks of a PSA model is not easy. It must also meet constraints imposed by project managers to deliver new PSA on time and within the allocated development budget. Therefore, ways of working must evolve.

One solution to help analysts to cope with the increasing challenges is to introduce software tools that allow to **automate** certain tasks.

Unfortunately today's risk assessment tools fail to provide concepts and functionality for task automation. Because of the high amount of time necessary for the manual execution of certain tasks, for example those related to model validation and quantification, users tend to perform those less frequently and consistently. However **frequent** and **early** feedback is important to ensure quality and efficiency in model engineering.

Automation is not exempt of drawbacks but has nevertheless several benefits if it is well-managed: Firstly, it reduces manual work of low interest of analysts which can focus on other tasks (ideally those that require PSA skills or impossible to be automated). Secondly, it contributes to quality of model engineering and risk assessment in general because automated tasks are *consistently* and *repeatable* executed what is generally less error-prone than manual (human) executions. Thirdly, it ensures that assets related to a same study are consistently updated, for example PSA models and their quantification results.

The objective of this article is to present an approach and use cases of automating tasks in the field of model based safety engineering.

Our approach applies principles from software engineering, in particular those from CI/CD [3][10][11] and DevOps practices [9][8][2], to model engineering. Indeed, from a technical point of view, the challenges faced during (PSA) model development are quite related to those from source code engineering:

1. Models may be partly generated in today's tools like Andromeda[4][5], or Python/Java/... scripts, likewise source code is often partly generated.
2. Executing model consistency tests (model validation) is similar to running software tests.
3. Quantifying models can be seen similar to source code compilation.
4. Deploying risk assessment together with documentation and reports is similar to deploying software packages.
5. Providing specific IT infrastructure (for example high performance clusters for intensive calculations) is a well known issue from DevOps.

It is thus an interesting idea to figure out if and how experiences from software engineering could enable automation capacities for model engineering.

After the first chapter dedicated to the introduction, this article is organised as follows.

Chapter 2 provides current enhancements of automation in model and software engineering, applied to the PSA model engineering field. While automation has gained significant importance in software engineering, it is quite unknown and currently not used in model engineering in today's risk analysis tools.

In chapter 3, we introduce a methodology for automating tasks. Our methodology allows to specify, trigger, schedule and execute tasks in form of so-called *pipelines* and to obtain execution feedback.

We implemented our approach in *EdgeMind PSA*, a risk assessment platform developed at EdgeMind. We provide some additional information about this tool in chapter 4.

In chapter 5 we explain how we tested our approach. We define some significant real-life use cases that are representative for today's lack of automation capacities and explain how we automated them with our approach.

In chapter 6 we state whether our expectations have been reached with satisfaction and provide some additional observations regarding best practices.

Finally, in chapter 7 we summarize our work and elaborate on what it could mean for future risk assessment activities in case the importance of automation gets recognized within the risk assessment community.

II. STATE OF THE ART

This chapter precises insights about current automation possibilities in the field of software and model engineering.

A. Automation in software engineering

Software development practices have evolved significantly with the adoption of DevOps principles, particularly in the realm of Continuous Integration/Continuous Deployment (CI/CD) [3][10] [11].

This chapter provides an overview of the current state of automation in software engineering, with a specific focus on CI/CD practices within the DevOps framework.

The integration of CI/CD practices into DevOps methodologies has transformed the software development lifecycle, enabling teams to deliver high-quality software at a rapid pace. Frequent software releases reveal software problems and the need for user changes early and often during developments. Initially introduced as separate processes, CI and CD have become integral components of DevOps pipelines, facilitating automation, collaboration, and efficiency across development, testing, and deployment stages.

Key Components of CI/CD Automation [11]:

1. Version Control Systems (VCS): provide robust branching and merging capabilities essential for CI/CD workflows and additionally features such as issue tracking, code reviews, and automated testing integration.
2. Automated testing: Automated testing provides rapid feedback on the quality of code changes. By automating tests, developers can quickly identify and address issues introduced during the development process. Automated testing helps maintain software quality by systematically validating functionality, performance, and reliability. Risks of regression after introduction of new functionalities are then reduced.
3. Continuous Integration (CI) [3]: automate the build and testing process, ensuring that code changes are regularly integrated and verified. These tools support automated testing suites, static code analysis, and artifact generation, enabling rapid feedback loops for developers.
4. Continuous Deployment (CD) [12]: automate the deployment of applications to various environments, including development, staging, and production. Containerization technologies have revolutionized CD practices, enabling seamless deployment across diverse infrastructures.
5. Provision of Infrastructure: enable organizations to scale infrastructure resources up or down dynamically in response to changing demand. By leveraging cloud computing platforms and elastic scaling capabilities, teams can provision additional resources automatically to handle increased workloads and release them when no longer needed, optimizing resource utilization and cost efficiency.

B. Automation in model engineering in Andromeda

Automation in the field of PSA model engineering started in some engineering teams due to the lack of human PSA skills in regards to the increase of the scope extension and applications of PSA insights in the safety probabilistic demonstration. However, it is relatively unknown and currently not a focus in model engineering or in today's risk analysis tools.

Andromeda, one of the most modern PSA management tools [7][4], can however provide basic functionality that could ease automation :

- Version Control System Support [4]: Andromeda manages model versions by using a version control system (VCS). This is a crucial functionality for automation because it allows to trigger automation on model changes on clear model states under version control.
- Scripting : Andromeda provides scripting interfaces for model engineering and a dedicated API to facilitate massive model modifications and data research [6]. While a script itself is not yet an automation, it could be called from an automation workflow in order to execute PSA functionality.
- Andromeda CLI: Andromeda CLI (Command Line Interface) allows to access core modeling and quantification core functionality of Andromeda without the need of installing the full application. Further, the CLI has only few dependencies and can easily be executed in other environments as often required by automation systems.

Further, some interesting functionality exists in the form of batch mode (without GUI), which may be an advantage (if not mandatory) for automation:

- generation of PSA event trees by the means of Sequence Analysis Diagrams [5]
- execution of model consistency tests
- execution of qualification tests
- execution of non regression of quantitative results

- local and remote quantification of PSA models

III. METHODOLOGY

This chapter presents our approach to enable task automation. First we introduce the notion of pipelines and provide information that concerns their execution. Then we share some architectural considerations of how to apply DevOps principles to risk assessment tools. Finally we discuss similarities and differences with Gitlab's concept for automation.

A. Pipelines

A pipeline precises when, how and what operations (tasks) to execute. It introduces the fundamental elements of specification that is required and processed by the Automation System. This notion has been adopted from Gitlab [13] where pipelines are the fundamental element for managing CI/CD.

A pipeline consists of the following items:

- Events (*when*): defines which events can trigger the pipeline. Note that a pipeline can be triggered in various ways, see Section 3.2.1.
- Stages (*how*): groups a set of operations. Operations that belong to a same stage can execute in parallel, operations of different stages must execute in sequence.
- Task (*what*): defines the operation to execute.
- Notification: defines how and whom to notify about execution results (successes and errors).

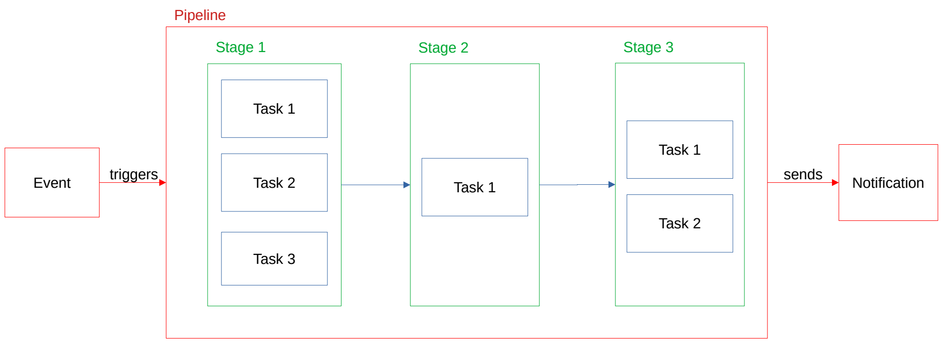


Figure 1: Pipelines: A pipeline executes operations in form of tasks that are organized in stages. While tasks of a same stage execute in parallel, tasks between different stages execute in sequence.

In our approach, to simplify execution, stages are strictly ordered: There is a first stage, subsequent intermediate stages and a last stage.

1. *Example Pipeline*

Figure 2 shows a small pipeline that executes two sequential tasks. The first one verifies model consistency, the second one quantifies a PSA model.

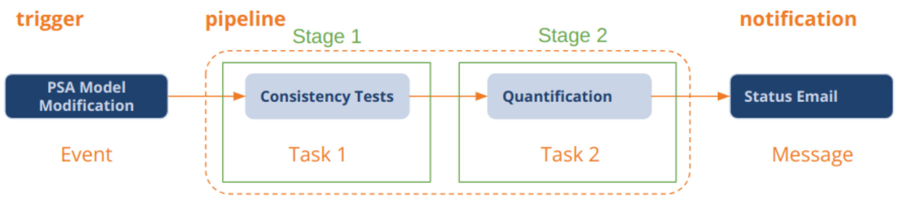


Figure 2: Example Pipeline to verify model consistency and to run quantification on PSA model changes. The final task status is sent via email.

The pipeline is triggered by an event that indicates a PSA model change. In practice, this event comes from VCS (Version Control System) and is only sent when a user creates a new model version.

Finally, a message is sent to user(s), indicating the pipeline state (successful or failing) and additional information (execution time etc.)

2. *Pipeline Scripts*

Pipelines may be specified in form of scripts, i.e. in a textual language that conforms to a certain syntax. Pipeline scripts are thus a way of encoding pipelines that is rather simple and technical. This is analog to Gitlab CI scripts.

For our approach, we introduced a grammar for pipeline scripts, that allows to encode stages, tasks, triggers, notifications in different sections within a script.

Pipeline scripts are suggested to be under version control, likewise risk models and related assets, as it is a part of the quality's demonstration.

Two different items are exposed to the notion of pipeline scripts:

- Risk Analysis Tool: In order to specify pipelines, to launch them manually (see section 3.2.1) and to show execution results.
- Automation System: In order to execute pipelines.

Pipelines variables: Value holders that can be used within pipeline scripts in order to customize pipelines. Values for pipelines variables are typically set outside pipeline definitions (they are 'injected'). However pipelines may precise default values for their variables.

B. Execution

In our approach, the stages of a pipeline are processed one by one (in sequence). The first stage is always executed. Stage N (except the first stage) is only executed in case stage N-1 has successfully terminated. If one stage fails, the whole pipeline fails.

A stage is successful, if all of its tasks have been successfully executed. The tasks of a same stage are executed in parallel. If one task fails, the stage fails as well.

1. *Trigger*

A pipeline can be triggered in several ways:

1. Manually by user: The user configures and launches the automation.
2. Automatically by an event: An event (such as a 'commit' event from version control) launches the automation.
3. Externally via API: Executed by third party software.

2. *State*

If not terminated, the *state* of a pipeline is either:

- Pending: planned for execution, but not yet executing
- Executing: the pipeline is currently executing. Depending on the structure of the pipeline, some elementary tasks can be finished, other running, and the last waiting for termination of other upstream tasks.

After execution, the *state* is either:

- Success: the execution of all jobs has been successfully
- Failed: a problem occurred while execution, for example
  - the execution of a job failed
  - the automation script is inconsistent
- Cancelled: the user cancelled the execution before pipeline termination. In complex pipelines, intermediate notifications can provide useful information and save engineering time if it is obvious that the results are wrong.

3. *Log*

Log: Each pipeline execution is associated with (regardless of its state):

- a main execution log
- a detailed log per task

C. Architectural Considerations

As mentioned in the introduction, our goal is to apply several DevOps principles (known from software engineering) to model engineering and risk assessment, in particular those from CI/CD. Figure 3 illustrates how this can be achieved.

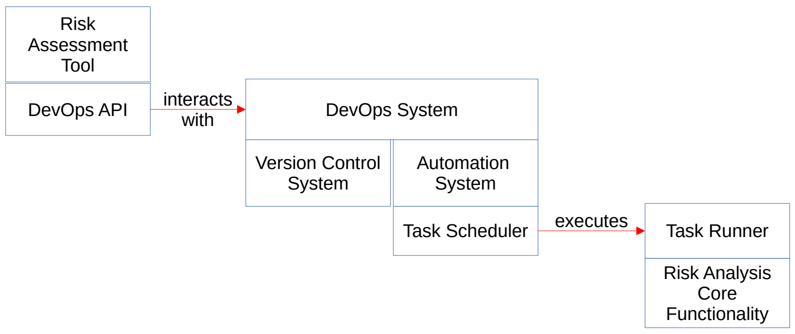


Figure 3: Concept: The risk assessment software interacts with the DevOps tool. An Automation System is the responsible part to schedule and launch tasks.

We propose to introduce a *DevOps Tool* that provides DevOps related functionality. Risk assessment tools use a *DevOps API* to interact with the *DevOps Tool*.

The *DevOps Tool* should provide a *Version Control System (VCS)* and an *Automation System*. While the VCS is responsible to manage version of PSA models, results and any other related assets, the *Automation System* allows to automate task executions.

A strong interaction between the VCS and *Automation System* is essential in order to deliver the full potential of the approach. The *Automation System* operates on consistent model versions and results managed by VCS, may be triggered by VCS events (for example when a user creates a new model version) and may write results back to VCS.

Another important point is the *asynchronous* nature of many executed tasks, i.e. the deferred execution of tasks. For this purpose, we propose the introduction of a *Task Scheduler* as part of the *DevOps Tool*, dedicated to plan and schedule the execution of tasks. Tasks can have dependencies towards other tasks. It is the *Task Scheduler* that should take these dependencies into account, for example to ensure a task is not launched before termination of its dependent tasks (this decides finally which tasks may execute in parallel or in sequence).

The execution itself should be performed within a dedicated *TaskRunner*. *TaskRunners* should have access to *Risk Analysis Core Functionality*, in form of an API or library that provide core functionality related to risk model engineering and quantification. *TaskRunner* instances are supposed to be created and launched by the *Task Scheduler*. Further they should run in an *isolated* environment that is not conflicting with model and results files a user is currently working with (while task execution).

The asynchronous task execution proposed in our approach has two advantages:

- users are not blocked waiting until task termination.
- the right moment for task launching can be optimized due to multiple criteria (task dependencies, CPU and memory constraints, load balancing etc.).

Note that there are three different environments to consider. While the risk assessment tool runs typically locally at the operator's host (likewise certain scripts used to automate PSA transformations), the *DevOps Tool* and the *Task Executors* run typically (but not necessarily) remotely (at a remote server). This permits to centralize DevOps activities between operators and to disburden single operators regarding IT infrastructure and operations. Finally, often in dependence with the kind of task to execute, the *DevOps Tool* and the *Task Runner* may run on different remote environments. For example, the task related to the quantification of PSA models may execute on a High Performance Cluster (HPC) for performance reasons.

D. Differences with Gitlab

Our approach aligns with concepts from Gitlab CI [1], the "continuous integration" feature provided by version control system (VCS) Git [14], and CI/CD concepts from software engineering in general. From Gitlab CI, we took over the way to base automation on script files and the introduction of so-called pipelines, jobs (tasks in our approach) and events. The purpose of pipelines is to schedule a set of possibly dependent individual tasks in a consistent way and for sequential tasks to ensure subsequent tasks are only executed in case previous ones have succeeded. Events are used to trigger pipelines.

While notions such as **Job** (though called **Task** in our approach), **Stage** and **Pipeline** have been adopted from Gitlab, there are two general differences to mention:

1. Simplicity: All Jobs of a same Stage are executed in parallel (except their number exceeds the maximum number of available threads or processes). All Stages are executed in sequence (one by one). There are no possibilities to have specific job dependencies such as with Gitlab. Keep in mind that Gitlab is designed for developers (which are supposed to have a rather technical background, while our approach is designed for risk analysis with possibly less technical IT background compared to developers).

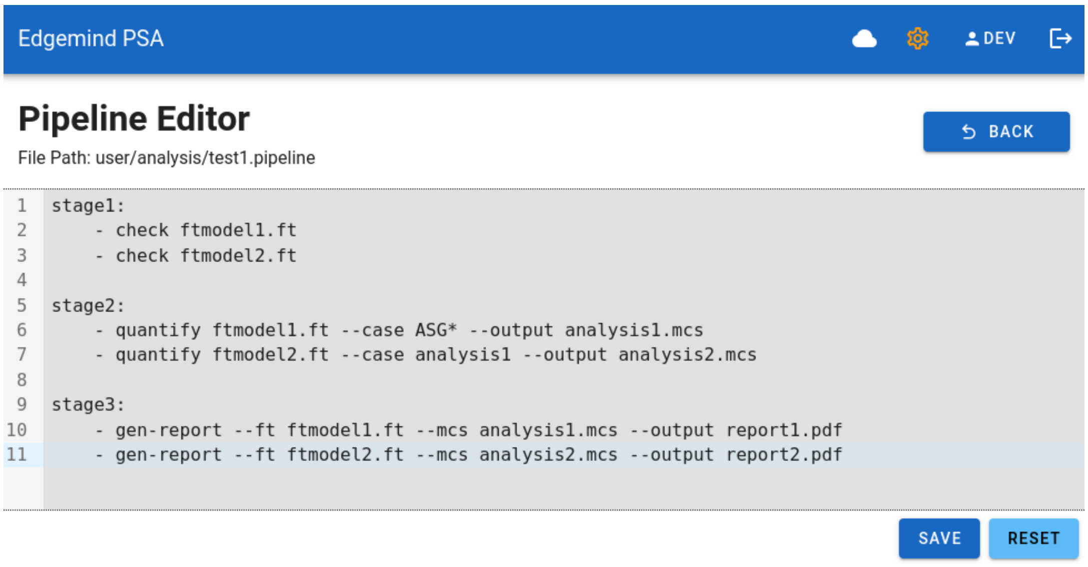
2. Events in Gitlab are mainly those related to version control (though there are webhooks and APIs for external events). Events in our approach are more general.

IV. IMPLEMENTATION

In this chapter, we inform about our test implementation in EdgeMind PSA. EdgeMind PSA is a research risk assessment platform developed at EdgeMind in order to explore, test and validate new concepts related to model engineering and risk assessment.

We have introduced and implemented the following items:

- Automation System that interprets and schedules Pipelines
- Task Runner concept in form of remote task executions
- Script language to specify Pipelines
- Event based notification system required to *trigger* Pipelines
- Integration with Git version control system
- View to create and edit pipelines, see Figure 4.



File Path: user/analysis/test1.pipeline

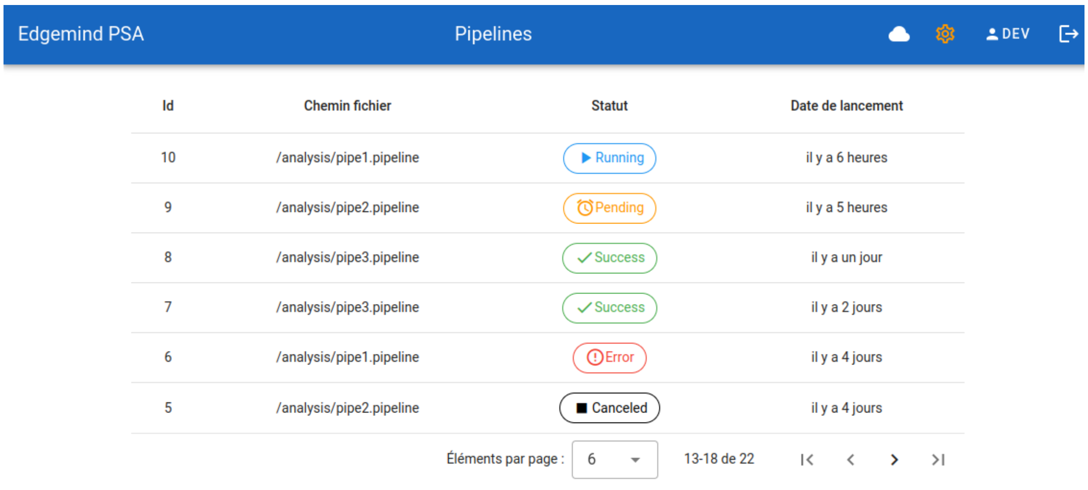
```
stage1:
    - check ftmodel1.ft
    - check ftmodel2.ft

stage2:
    - quantify ftmodel1.ft --case ASG* --output analysis1.mcs
    - quantify ftmodel2.ft --case analysis1 --output analysis2.mcs

stage3:
    - gen-report --ft ftmodel1.ft --mcs analysis1.mcs --output report1.pdf
    - gen-report --ft ftmodel2.ft --mcs analysis2.mcs --output report2.pdf
```

Figure 4: Pipeline Editor in EdgeMind PSA permits to create and edit pipelines.

- View to list ongoing and finished Pipelines, see Figure 5.



Id	Chemin fichier	Statut	Date de lancement
10	/analysis/pipe1.pipeline	Running	il y a 6 heures
9	/analysis/pipe2.pipeline	Pending	il y a 5 heures
8	/analysis/pipe3.pipeline	Success	il y a un jour
7	/analysis/pipe3.pipeline	Success	il y a 2 jours
6	/analysis/pipe1.pipeline	Error	il y a 4 jours
5	/analysis/pipe2.pipeline	Canceled	il y a 4 jours

Éléments par page : 6 13-18 de 22

Figure 5: Pipeline View in EdgeMind PSA lists ongoing and terminated pipeline executions.

\

V. TESTS

In this chapter, we setup a pipeline in order to automate some interesting tasks.

A. Goals

We want to test and evaluate the automation of the following tasks:

- Automate Event Tree Generation
- Automate PSA Model Consistency Testing
- Automate PSA Model Quantification
- Automate Report Generation
- Automate Result Deployment

B. Setup

This section precises the test setup used to validate our approach. We consider a real-life scenario as it may find application at EDF, see Figure 6.

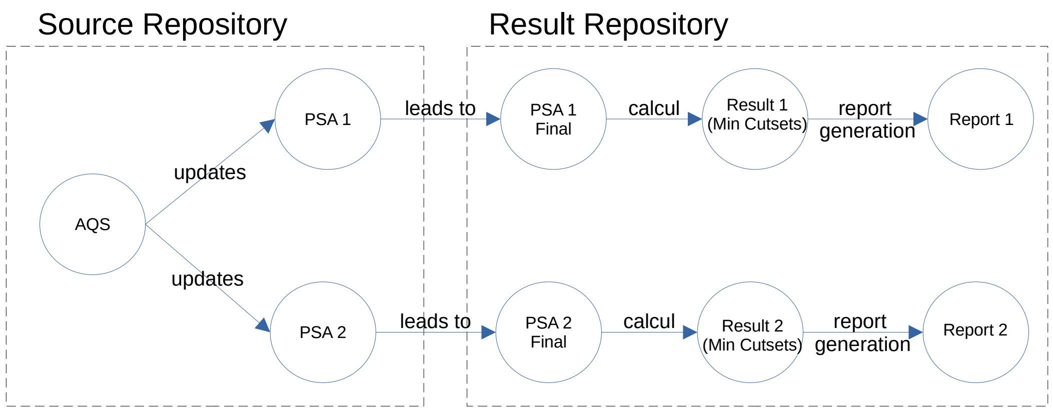


Figure 6: Test Setup: Two PSA Models are party updated from an AQS model. Then cutsets and reports are produced. While the initial models reside in the source repository, the updated models, results and reports are put into a result repository.

Explications:

- We consider two PSA Models but without Event Trees.
- The Event Trees are supposed to be generated from a common AQS (Sequence Analysis Diagrams).
- After generating the Event Trees, the final PSA models get quantified by an risk assessment tool. The results are typically in form of minimal cutsets.
- Once quantification results are available, additional reports are generated from those.
- The source repository is under version control and stores the initial source model, i.e one AQS model and two PSA models.
- The result repository is also under version control and stores the results and the final PSA model.

The separation between source and result repositories has the following advantages: Firstly, risk of corruption is impossible as source and output data are separated. Secondly, ensuring reproducibility as often as necessary without interfering with the source repository.

Note that the final PSA Model is uploaded together with results and reports to the result repository. This is to ensure consistency between the actual model (the one that got finally quantified) and corresponding results.

C. Pipeline

Figure 7 shows a pipeline with 6 stages.

The pipeline is triggered if any of the source models has been modified. First, consistency tests are executed for the Sequence Analysis Diagram (AQS) model (stage 1). Then, in stage 2, event trees are generated into the PSA models. Note that it is not

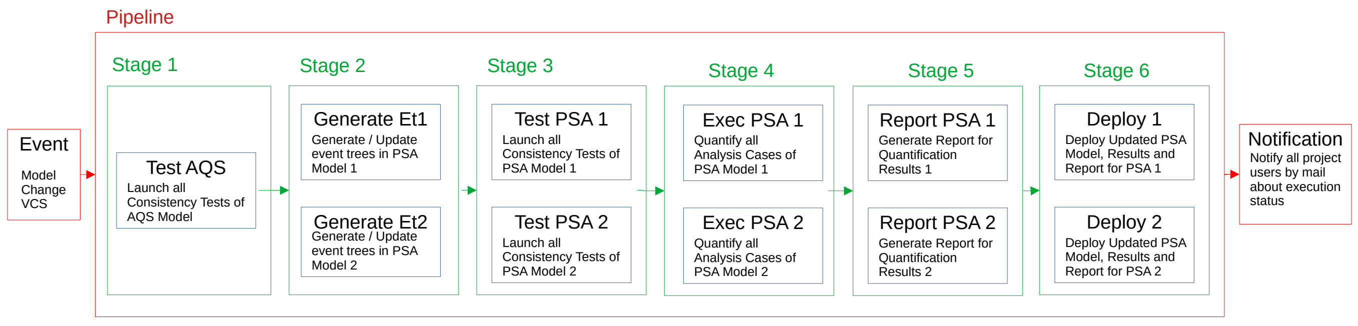

Pipeline
Event
Model Change VCS
Stage 1
Test AQS
Launch all Consistency Tests of AQS Model
Stage 2
Generate Et1
Generate / Update event trees in PSA Model 1
Generate Et2
Generate / Update event trees in PSA Model 2
Stage 3
Test PSA 1
Launch all Consistency Tests of PSA Model 1
Test PSA 2
Launch all Consistency Tests of PSA Model 2
Stage 4
Exec PSA 1
Quantify all Analysis Cases of PSA Model 1
Exec PSA 2
Quantify all Analysis Cases of PSA Model 2
Stage 5
Report PSA 1
Generate Report for Quantification Results 1
Report PSA 2
Generate Report for Quantification Results 2
Stage 6
Deploy 1
Deploy Updated PSA Model, Results and Report for PSA 1
Deploy 2
Deploy Updated PSA Model, Results and Report for PSA 2
Notification
Notify all project users by mail about execution status

Figure 7: Pipeline with 6 stages that automates consistency validations, event tree generation, psa quantification, report generation and final result deployment.

necessary to save the outcome in new models because the automation works in an isolated environment and original source models are not altered.

In stage 3, consistency tests are executed on the PSA models. If everything is ok, the PSA models are quantified (stage 4). Once results are obtained, reports are produced from them (stage 5).

Finally, in stage 6 results and the original model are uploaded to the result repository (deployment).

VI. RESULTS

The tests demonstrated the advantages of automation:

- Rapid feedback on the consistency of models: In case of a problems, the pipeline execution stops immediately and we received a mail that informed us about the problem.
- Needless to perform time intensive tasks (such as PSA quantifications) locally: All tasks are executed remotely and automatically (a major benefit).
- Consistency between deployed PSA models and results: The deployment stage copied the results and final PSA models into a same directory that indicated project version and timestamp.

However we noticed some points of attention that lead to the following best practices as lessons learned:

- In case a source file (for example a PSA model) gets modified by automation, the modified file should not overwrite the original source file in the source repository. This could lead to inconsistencies when someone modifies the source file manually while an automation is ongoing at same time. In our case, the modified files (the final PSA models) are deployed into a distinct result repository.
- It may become necessary to define a human organization when multiple Users work on the same PSA project.
- In case of multiple pipelines are triggered equally, one has to ensure that same tasks are not executed twice to optimize the production of new results.
- In case of interdependencies between pipelines, one has to ensure that there are no loops. However such complex pipelines' modeling should in practice be reserved to PSA experts that have a better comprehension of the full process.

Some of the mentioned points should be considered to be tested and avoided by the Automation System of our approach.

VII. CONCLUSION

In conclusion, the presented approach for automating recurrent tasks could mean a significant innovation in the realm of risk management. The combination of advanced automation, precise task management, and seamless integration with version control systems can support safety analyst significantly in their daily work.

By embracing automation, organizations can accelerate model engineering activities, improve quality, and enhance collaboration across development and operations teams.

Such automation should nevertheless not be developed without IT formation of the users, particularly for new PSA analysts. Indeed, they may forge a wrong opinion of the complexity of the tasks necessary to develop a PSA model and become unable to develop a critic spirit of the final results of the pipelines production.

To conclude, as for many other IT subjects, automation has a very high potential in this time of research of optimization of the PSA processes but shall be used wisely. A comprehension of the induced risks of introducing too much automation guides currently the research program around EdgeMind PSA by mixing IT and PSA engineer visions. The goal is to find a good and healthy equilibrium where PSA human skills are well-managed.

VIII. BIBLIOGRAPHY

References

[1] Mohammed Shamsul Arefeen and Michael Schiller. Continuous integration using gitlab. *Undergraduate Research in Natural and Clinical Science and Technology Journal*, 3:1–6, 2019.

[2] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *IEEE software*, 33(3):94–100, 2016.

[3] Martin Fowler and Matthew Foemmel. Continuous integration, 2006.

[4] T. Friedlhuber, M. Hibti, and A. Rauzy. Overview of the open psa platform. In R. Virolainen, editor, *Proceedings of International Joint Conference PSAM'11/ESREL'12*, June 2012.

[5] T. Friedlhuber, M. Hibti, and A. Rauzy. Automated generation of event trees from event sequence/functional block diagrams and optimisation issues. In A. Yamaguchi, editor, *Proceedings of PSAM Topical Conference in Tokyo*, April 2013.

[6] T. Friedlhuber, M. Hibti, and A. Rauzy. Andromeda scripting interface to efficiently treat psa models. In *Proceedings of International Topical Meeting on Probabilistic Safety Assessment and Analysis, PSA 2015*, USA, April 2015. ANS.

[7] Thomas Friedlhuber. *Model Engineering in a Modular PSA*. Theses, LIX, Ecole polytechnique ; EDF R&D, October 2014.

[8] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. What is devops? a systematic mapping study on definitions and practices. In *Proceedings of the scientific workshop proceedings of XP2016*, pages 1–11, 2016.

[9] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of devops concepts and challenges. *ACM Computing Surveys (CSUR)*, 52(6):1–35, 2019.

[10] MEYER, Mathias. Continuous integration and its tools. *IEEE software*, 31(3):14–16, 2014.

[11] Sikender Mohsienuddin Mohammad. Continuous integration and automation. *International Journal of Creative Research Thoughts (IJCRT), ISSN*, pages 2320–2882, 2016.

[12] Pilar Rodríguez, Alireza Haghighatkhah, Lucy Ellen Lwakatare, Susanna Teppola, Tanja Suomalainen, Juho Eskeli, Teemu Karvonen, Pasi Kuvaja, June M Verner, and Markku Oivo. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of systems and software*, 123:263–291, 2017.

[13] Vikas Singh. Developing a ci/cd pipeline with gitlab. 2022.

[14] Diomidis Spinellis. Git. *IEEE software*, 29(3):100–101, 2012.