Programme de la journée commune des GT VL & IDM

Paris, LIP6, 10 Mars 2023

How to come to LIP6 : https://www.lip6.fr/informations/comment.php

~~The meeting will take place in room 101 "Laurière" at level 1 between towers 25 & 26.~~

~~There is a door in the corridor to access the room. If it is closed, please call the phone number you have received by e-mail.~~

Zoom meeting information :

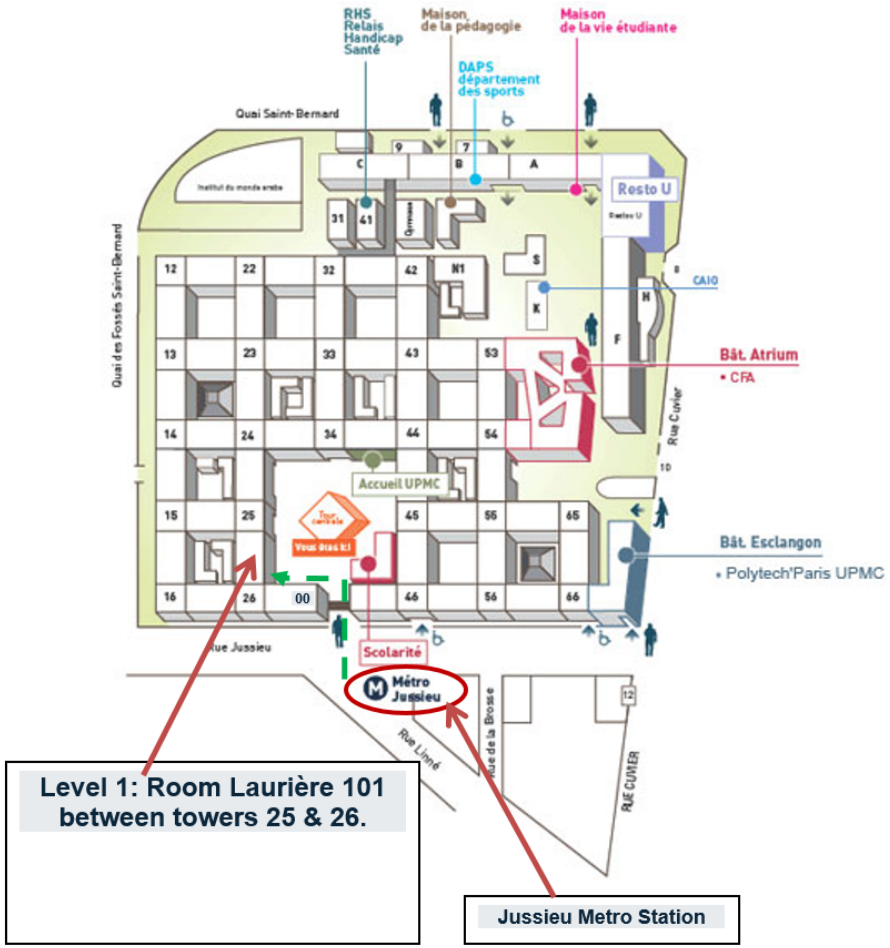
Topic: GT Vélocité Logicielle
Time: Mar 10, 2023 09:30 AM Paris

Join Zoom Meeting
https://u-bordeaux-fr.zoom.us/j/85792741747?pwd=TmtBalBVbDVpWURvOU40a3JKS29uZz09

Meeting ID: 857 9274 1747
Passcode: 978808

Join by SIP
85792741747@zoomcrc.com

Join by H.323
162.255.37.11 (US West)
162.255.36.11 (US East)
213.19.144.110 (Amsterdam Netherlands)
213.244.140.110 (Germany)
Meeting ID: 857 9274 1747
Passcode: 978808



Programme du matin

Horaire	Orateur	Email	Titre	Résumé
9h30-10h00	**Accueil & Café**			
10h00-11h00	Jean-Rémy Falleri, LABRI	falleri@labri.fr	Diff de logs de builds logiciels	
11h00-11h15	**Pause café**			
11h15-11h45	Aless Hosry, INRIA Lille	aless.hosry@inria.fr	Classification and detection of hidden dependencies in software engineering.	The use of multi-tier and multi-language applications has become increasingly common in modern software practice. In such applications, developers bind numerous artifacts written in different languages or tiers, leading to hidden dependencies between these artifacts. The identification of those dependencies is necessary to provide advanced tools going from smart refactoring to software reengineering. The identification of those dependencies is necessary to provide advanced tools going from smart refactoring to software reengineering. Some tools identify such dependencies in a very specific way: the rules to detect them are usually enforced in the tool and there is no generic framework to help defining them. In this presentation, we focus on identifying multiple types of hidden dependencies and discuss a new tool XLLDetector that we developed, how patterns are developed and rules are codified, and some early results.
11h45-12h15	Guillaume HABEN, Université du Luxembourg	guillaume.haben@uni.lu	What Made This Test Flake? Pinpointing Classes Responsible for Test Flakiness	Flaky tests are defined as tests that manifest non-deterministic behaviour by passing and failing intermittently for the same version of the code. These tests cripple continuous integration with false alerts that waste developers' time and break their trust in regression testing. To mitigate the effects of flakiness, both researchers and industrial experts proposed strategies and tools to detect and isolate flaky tests. However, flaky tests are rarely fixed as developers struggle to localise and understand their causes. Additionally, developers working with large codebases often need to know the sources of non-determinism to preserve code quality, i.e., avoid introducing technical debt linked with non-deterministic behaviour, and to avoid introducing new flaky tests. To aid with these tasks, we propose re-targeting Fault Localisation techniques to the flaky component localisation problem, i.e., pinpointing program classes that cause the non-deterministic behaviour of flaky tests. In particular, we employ Spectrum-Based Fault Localisation (SBFL), a coverage-based fault localisation technique commonly adopted for its simplicity and effectiveness. We also utilise other data sources, such as change history and static code metrics, to further improve the localisation. Our results show that augmenting SBFL with change and code metrics ranks flaky classes in the top-1 and top-5 suggestions, in 26% and 47% of the cases. Overall, we successfully reduced the average number of classes inspected to locate the first flaky class to 19% of the total number of classes covered by flaky tests. Our results also show that localisation methods are effective in major flakiness categories, such as concurrency and asynchronous waits, indicating their general ability to identify flaky components.
12h15-12h45	Nicolas Méric, LRI	nicolas.meric@lri.fr	Using Ontologies in Formal Developments Targeting Certification	A common problem in the certification of highly safety or security critical systems is the consistency of the certification documen- tation in general and, in particular, the linking between semi-formal and formal content of the certification documentation. We address this problem by using an existing framework, Isabelle/DOF, that allows writing certification documents with consistency guarantees, in both, the semi-formal and formal parts. Isabelle/DOF supports the modeling of document ontologies using a strongly typed ontology defini- tion language. An ontology is then enforced inside documents including formal parts, e. g., system models, verification proofs, code, tests and validations of corner-cases. The entire set of documents is checked within Isabelle/HOL, which includes the definition of ontologies and the editing of integrated documents based on them. This process is supported by an IDE that provides continuous checking of the document consistency. In this paper, we present how a specific software-engineering certification standard, namely CENELEC 50128, can be modeled inside Is- abelle/DOF. Based on an ontology covering a substantial part of this standard, we present how Isabelle/DOF can be applied to a certification case-study in the railway domain.
12h45-12h55	Andrey Sadovykh, Softeam	andrey.sadovykh@softeam.fr	Requirements as Code	(exposé court)
12h55-13h05	Asbath Biyalou-Sama, CRIStAL	asbathou.biyalousama@univ-lille.fr	More simple interactions in MDE	(exposé court)
13h05-14h15	**Déjeuner**			

Programme de l'après-midi

14h15-14h45	Arnaud Blouin, IRISA	arnaud.blouin@irisa.fr	Interacto: A Modern User Interaction Processing Model	Since most software systems provide their users with interactive features, building user interfaces (UI) is one of the core software engineering tasks. It consists in designing, implementing and testing ever more sophisticated and versatile ways for users to interact with software systems, and safely connecting these interactions with commands querying or modifying their state. However, most UI frameworks still rely on a low level model, the bare bone UI event processing model. This model was suitable for the rather simple UIs of the early 80's (menus, buttons, keyboards, mouse clicks), but now exhibits major software engineering flaws for modern, highly interactive UIs. These flaws include lack of separation of concerns, weak modularity and thus low reusability of code for advanced interactions, as well as low testability. To mitigate these flaws, we propose Interacto as a high level user interaction processing model. By reifying the concept of user interaction, Interacto makes it easy to design, implement and test modular and reusable advanced user interactions, and to connect them to commands with built-in undo/redo support. To demonstrate its applicability and generality, we briefly present two open source implementations of Interacto for Java/JavaFX and TypeScript/Angular. We evaluate Interacto interest (1) on a real world case study, where it has been used since 2013, and with (2) a controlled experiment with 44 master students, comparing it with traditionnal UI frameworks.
14h45-15h15	Pooya Rostami Mazrae, Université de Mons	Pooya.ROSTAMIMAZRAE@umons.ac.be	On the Use of GitHub Actions in Software Development Repositories	GitHub Actions was introduced in 2019 and constitutes an integrated alternative to CI/CD services for GitHub repositories. The deep integration with GitHub allows repositories to easily automate software development workflows. This paper empirically studies the use of GitHub Actions on a dataset comprising 68K repositories on GitHub, of which 43.9% are using GitHub Actions workflows. We analyse which workflows are automated and identify the most frequent automation practices. We show that reuse of actions is a common practice, even if this reuse is concentrated in a limited number of actions. We study which actions are most frequently used and how workflows refer to them. Furthermore, we discuss the related security and versioning aspects. As such, we provide an overview of the use of GitHub Actions, constituting a necessary first step towards a better understanding of this emerging ecosystem and its implications on collaborative software development in the GitHub social coding platform.
15h15-15h45	Faezeh Khorram, IMT Atlantique	faezeh.khorram@inria.fr	Automatic test amplification for executable models	Behavioral models are important assets that must be thoroughly verified early in the design process. This can be achieved with manually-written test cases that embed carefully hand-picked domain-specific input data. However, such test cases may not always reach the desired level of quality, such as high coverage or being able to localize faults efficiently. Test amplification is an interesting emergent approach to improve a test suite by automatically generating new test cases out of existing manually-written ones. Yet, while ad-hoc test amplification solutions have been proposed for a few programming languages, no solution currently exists for amplifying the test cases of behavioral models. In this paper, we fill this gap with an automated and generic approach. Given an executable DSL, a conforming behavioral model, and an existing test suite, our approach generates new regression test cases in three steps: (i) generating new test inputs by applying a set of generic modifiers on the existing test inputs; (ii) running the model under test with new inputs and generating assertions from the execution traces; and (iii) selecting the new test cases that increase the mutation score. We provide tool support for the approach atop the Eclipse GEMOC Studio and show its applicability in an empirical study. In the experiment, we applied the approach to 71 test suites written for models conforming to two different DSLs, and for 67 of the 71 cases, it successfully improved the mutation score between 3.17% and 54.11% depending on the initial setup.
15h45-16h15	Corentin Latappy, LABRI	corentin.latappy@labri.fr	MLinter: Learning Coding Practices from Examples-Dream or Reality?	Coding practices are increasingly used by software companies. Their use promotes consistency, readability, and maintainability, which contribute to software quality. Coding practices were initially enforced by general-purpose linters, but companies now tend to design and adopt their own company-specific practices. However, these company-specific practices are often not automated, making it challenging to ensure they are shared and used by developers. Converting these practices into linter rules is a complex task that requires extensive static analysis and language engineering expertise. In this paper, we seek to answer the following question: can coding practices be learned automatically from examples manually tagged by developers? We conduct a feasibility study using CodeBERT, a state-of-the-art machine learning approach, to learn linter rules. Our results show that, although the resulting classifiers reach high precision and recall scores when evaluated on balanced synthetic datasets, their application on real-world, unbalanced codebases, while maintaining excellent recall, suffers from a severe drop in precision that hinders their usability.
16h15-16h30	**Pause café**			
16h30-17h00	**Bilan / Discussion**			