COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *« voir README et le site de votre école doctorale »*

Par

« Prénom NOM »

« Titre de la thèse »

« Sous-titre de la thèse »

Thèse présentée et soutenue à « Lieu », le « date »
Unité de recherche : « voir README et le site de de votre école doctorale »
Thèse N° : « si pertinent »

Rapporteurs avant soutenance :

Prénom NOM Fonction et établissement d'exercice
Prénom NOM Fonction et établissement d'exercice
Prénom NOM Fonction et établissement d'exercice

Composition du Jury :

Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition du jury doit être revue pour s'assurer qu'elle est conforme et devra être répercutée sur la couverture de thèse

Président :	Prénom NOM	Fonction et établissement d'exercice *(à préciser après la soutenance)*
Examinateurs :	Prénom NOM	Fonction et établissement d'exercice
	Prénom NOM	Fonction et établissement d'exercice
	Prénom NOM	Fonction et établissement d'exercice
	Prénom NOM	Fonction et établissement d'exercice
Dir. de thèse :	Prénom NOM	Fonction et établissement d'exercice
Co-dir. de thèse :	Prénom NOM	Fonction et établissement d'exercice *(si pertinent)*

Invité(s) :

Prénom NOM Fonction et établissement d'exercice

Acknowledgement

Je tiens à remercier
I would like to thank. my parents..
J'adresse également toute ma reconnaissance à ....
....

Table of Contents

INTRODUCTION

Context and Motivation

Software systems are increasingly growing in complexity, which leads to a substantial burden in terms of maintenance, often resulting in a high cost that may surpass the cost of software development itself [1]. Since Object Management Group (OMG) has introduced Model driven Engineering in 2001 [2], MDE has been prominent in developing and maintaining large-scale and embedded systems while increasing the developers' productivity. By adopting MDE, industry can reduce time (development time, time-to-market), costs (development, integration, reconfiguration), and improve sustainability and international competitiveness Metamodel is a central artifact for building software languages [3]. It specifies the domain concepts, their properties, and the relationship between them. A metamodel is the cornerstone to generate model instances, constraints, transformations, and code when building the necessary language tooling, e.g. editor, checker, compiler, data access layers, etc. In particular, metamodels are used as inputs for complex code generators that leverage on the abstract concepts defined in metamodels. The generated code API for creating, loading and manipulating the model instances, adapters, serialization facilities, and an editor, all from the metamodel elements. This generated code is further enriched by developers to offer additional functionalities and tooling, such as validation, transformation, simulation, or debugging. For instance, UML [1] and BPMN [2] Eclipse implementations rely on the UML and BPMN metamodels to generate their corresponding code API before building around it all their tooling and services in the additional code.

1. https://www.eclipse.org/modeling/mdt/downloads/?project=uml2
2. https://www.eclipse.org/bpmn2-modeler/

Challenges

C1: Resolve the impact of the metamodel evolution on the code automatically

One of the foremost challenges to deal with in MDE is the impact of the evolution of metamodels on its dependent artifacts. We focus on the impact of metamodels' evolution on the code. Indeed, when a metamodel evolves and the core API is regenerated again, the additional code implemented by developers can be impacted. As a consequence, this additional code must be co-evolved accordingly by executing a resolution for each impacted part of the code.

However, manual co-evolution can be tedious, error-prone, and time-consuming. Therefore, it is essential to support an automatic co-evolution of code when metamodels evolve. The co-evolution challenge has been extensively addressed in *MDE*. In particular, [4]–[9] focused on consistency checking between models and code, but not its co-evolution. Other works [10], [11] proposed to co-evolve the code. However, the former handles only the generated code API, it does not handle additional code and aims to maintain bidirectional traceability between the model and the code API. The latter supports a semi-automatic co-evolution requiring developers' intervention. Moreover, it does not use any validation process to check the correctness of the co-evolution and with no comparison to a baseline.

Considering that metamodel and co-evolution is one of many other MDE tasks like for example Model generation, code generation. Since their appearance, LLMs have been applied in different domains of scientific research, such as Software Engineering and Model-Driven Engineering (MDE), however, the challenge of exploring LLMs in the task of metamodel and code co-evolution is never addressed.

C2: Behavioral correctness of the metamodel and code co-evolution

In literature, when the problem of metamodel and code co-evolution is addressed, the challenge of checking that the co-evolution impacted or not the behavioral correctness of the code is not handled. In any Model-driven Engineered system, the elements of the metamodel are used in the code. The evolution of the metamodel will be propagated in the code that is co-evolved and its behavior may be altered. Hence, the importance of checking the correctness of the co-evolution.

Contributions

To tackle these challenges, we propose three contributions:

— First, we propose a fully automatic code co-evolution approach du to metamodel evolution based on pattern matching. Our approach handles both atomic and complex changes (ref) of the metamodel.

— Second, we investigate the ability of LLMs in giving correct co-evolutions in the context of metamodel and code co-evolution

— Third, w propose an approach that assist developers to check the behavioral correctness of the co-evolution. This approach leverages unit tests before and after the co-evolution and gives visual report about passing, failing, and erroneous tests before and after the co-evolution.

Chapter 1

State Of The Art

To develop a software, a list of specifications is given to the developers, to code the final product. This approach can work in the case of small projects.

When the complexity of the software increases, more efficient approaches must be adopted. Model driven engineering has proven its efficiency comparing to other engineering disciplines [12].

Model-Driven Engineering(MDE) is the systematic use of models as primary artifacts during a software engineering process. MDE includes various model-driven approaches to software development, including model-driven architecture, domain-specific modeling and model-integrated computing [13]. The first appearance of MDE like approaches started in the 80's [14]. MDE still adopted and a lot of work is being done in academia and industry, refs.

The goal of MDE is to improve productivity, quality, and maintainability by leveraging high level abstractions throughout the development process. The metamodeling phase implied the experts of the domain who focus on the major key aspects of the problem rather than being concerned about the underlying programming language and the implementation.

There are many definitions of the concept "metamodel" that can be found in literature:

A metamodel describes concepts that can be used for modeling the model (i.e. in the instances of the metamodel).

Metamodels are models that make statements about modeling. More precisely, a metamodel describes the possible structure of models in an abstract way, it defines the constructs of a modeling language and their relationships, as well as constraints and modeling rules – but not the concrete syntax of the language.

A metamodel defines the abstract syntax and the static semantics of a modeling language ,Vice versa, each formal language, such as Java or UML, possesses a metamodel.

Seidewitz [15] gives another commonly used definition of *metamodels* in MDE. A metamodel is a specification model for a class of systems under study where each system

under study in the class is itself a valid model expressed in a certain modeling language.

Metamodeling:

Metamodeling is the process of metamodel creation. Metamodeling is done thanks to metamodeling languages (that is in turn described by a meta metamdmodel).

Metamodeling must gathers the whole knowledge that is required to define, precise, and deal with MDE challenges in its different tasks:

— The construction of metamodel describes the abstract syntax of target (software languages, solution system).
— Model validation: models are validated against the constraints defined in the metamodel.
— Model-to-model transformations: such transformations are defined as mapping rules between two metamodels.
— Code generation: the generation templates refer to the metamodel of the "system".
— Tool integration: based on the metamodel, modeling tools can be adapted to the respective domain.

In the context Domain-Specific Language, DSMLs can be tailored via metamodeling to precisely match the domain's semantics and syntax. The concrete syntax that is, the concrete form of the textual or graphical constructs with which the modeling is done must represent the metamodel in an unambiguous way. Having graphic elements that linked directly to a familiar domain makes it easier to learn and allows domain experts to contribute, such as system engineers and experienced software architects, ensure that software systems meet user needs [16]. The metamodel (and not the concrete syntax) is the basis for the automated, tool-supported processing of Metamodeling models. On the other hand, a suitable concrete syntax is the interface to the modeler and its quality decides what degree of readability the models have **empty citation**

Metamodeling tools/languages/techniques examples :

// Add figure?

Metamodeling languages are classfied into two categories languistic and ontological[17]. Linguistic metamodeling represents way for defining modeling languages and their primitives (e.g., Object, Class, MetaClass) on the layer of a metamodel. Ontological metamodeling aims to represent domain knowledge accurately, it is s concerned with semantics and meaning, ex: OWL **empty citation** Linguistic metamodeling aims to define a language for creating models. it is concerned with syntax and structure. We can use a different classfication by purpose: General Purpose Modeling Langugaes and Domain-Specific

Modeling Languages [18]. General purpose modeling languages as for example : UML and its variants, generic metamodeling frameworks, such as MOF **empty citation** and Ecore **empty citation** As examples of DSLs, we cite sysML and EXPRESS DSL.

In MDE, there are language workbenches that are used for language creation, such as Xtext, MetaEdit+.

Generated artifacts, artifacts linked to the metamodel

Metamodel is the backbone in model driven engineering. In the language modeling ecosystem, other artifacts are created by the mean of the metamodel. By definition, the model is an instance of a metamodel, which mean that the metamodel defines the concepts with which a model can be created. Constraints are written in Object Constraint Language. The created models can be validated through a set of constraints to check the models' correctness. They precise specifications on the model that cannot be expressed by diagrammatic notation. In order to safe effort and avoid errors, models transformation is one of the common automated tasks in Model-driven engineering. Model transformation are expressed in Transformation Languages for example, ATL). A transformation consists of a set of rules that map the source metamodel elements to the metamodel target's elements.

// Add example of Metamodel+ model+constraint+transformation ?

Automation in the MDE ecosystem

Brief description about approaches that allow automated tasks in MDE.

Code Generation

One of the most important advantages of MDE, is the automation in many of its activities. The code generation activity is recurrent, and its automation enhances the productivity and the cost. For example, Eclipse Modeling Framework built-in code generator allows to generated a java API from an Ecore metamodel. The generated code API structure and technical choices are done to fit Java programming language and model driven engineering abstraction standards/principles (ex: each metaclass is used to generate an interface and concrete implementation class that extends the generated interface, pattern observer), to have an efficient as possible. *@generated* annotation is used to mark generated interfaces, classes, methods, and fields.

In Eclipse Modeling Framework, two model resources (files) are manipulated: the .ecore file that contains xmli serialization of the Ecore model and the .genmodel for the serialized generator model.

What is an evolution in the MDE context?

During the software development process, software artifacts are meant to be changed, due to many reasons: client requirements and domain specification, software maintenance or bug correction. Like any other software system, modeling languages are the subject of an inevitable evolution, during their process of building, multiple versions are developed, tested, and adapted until a stable version is reached.

Different types of evolution are categorized depending on the impact and purpose of the applied modifications [19], [20]:

— Corrective: aims to correct discovered problems and inconsistencies such as processing failures, performance failures, or implementation failures by applying a set of reactive modifications of a software product.

— Adaptive: in case of changing environment, such as changes in data environment or processing environment, this evolution aims to keep a software product usable.

— Perfective: this evolution aims to improve functionalities, enhance the performance, reliability, or to increase the maintainability of a software.

The term *Evolution* can be refined as the literature presents various related terms like : Maintenance, Refactoring, and **Co-evolution**, which are different types of modifications that could be applied on a software.

//Why we need to perform evolution ?

Evolution: when users and developers get by learning new requirements that lead to adapt the software to new changes **empty citation**

Maintenance:It is modifying a software product after delivery to correct faults, to improve performance, or to adapt the product to a changing environment **empty citation**

Refactoring: It is an oriented object term, that means modifications of software to make it easier to understand and to change or to make it less susceptible to errors when future changes are introduced **empty citation**

co-evolution: It consists of the process of adapting and correcting a set of artifacts A_1, A_2, ...A_N in response to the evolution of an artifact B on which A_1, A_2, ...A_N strongly depend **empty citation** for example the co-evolution of models with the evolving metamodel.

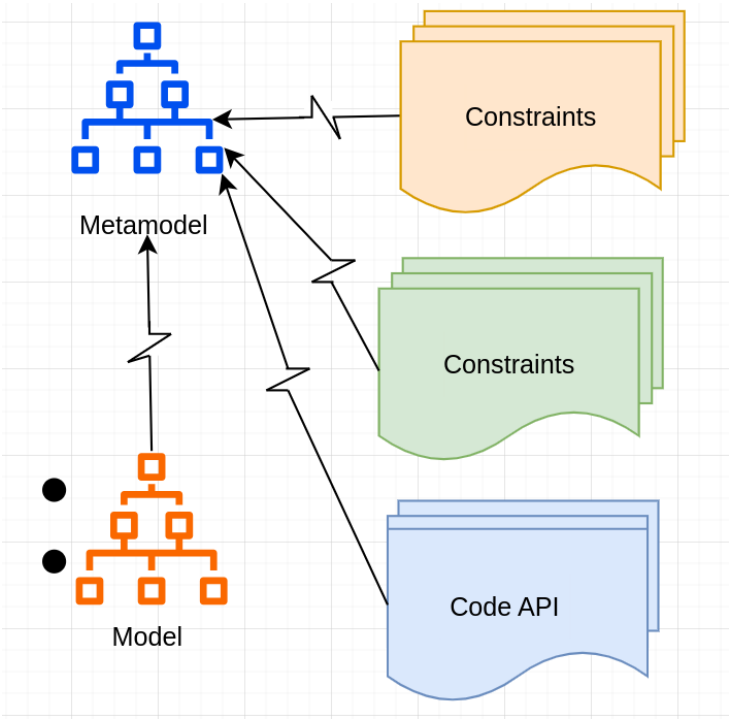

Figure 1.1 – Software Life Cycle

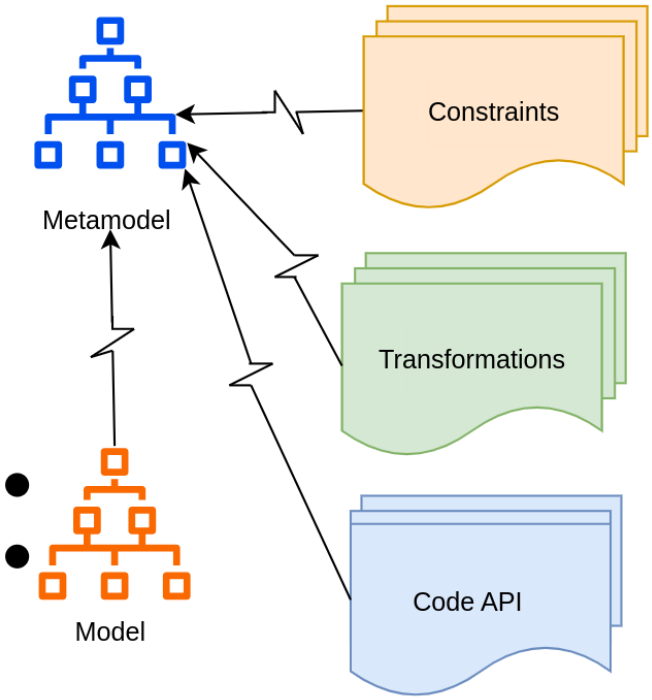

Figure 1.2 – MDE Ecosystem

Chapter 2

Titre du premier chapitre

2.1 Première section du chapitre

2.1.1 Première sous-section

2.2 Approach

This section presents the overall approach of our automated co-evolution of code with evolving metamodels, instantiating on the Ecore technological space. First, we give an overview of the approach and specify the metamodel evolution changes we consider. Then, we present how we retrieve the resulting errors due to metamodel evolution, followed by the regeneration of the code API. After that, we present the pattern matching process, which is an important part of our fully automatic co-evolution approach, before discussing the resolutions of the code errors.

2.2.1 Overview

Figure 2.1 depicts the overall steps for the automatic co-evolution of the metamodel and code, with horizontally separated parts defining chronological order from the top to the bottom. After the generation step (the upper part of Figure 2.1), the evolution of the Ecore metamodel will cause errors in the additional Java code that depends on the API of the newly generated code (the middle part of Figure 2.1). We take as input the evolution changes of the metamodel between the two versions of this metamodel [1]. Then, we parse the additional code [2] to retrieve the list of errors. After that, we get to the bottom part of Figure 2.1, both the list of metamodel changes and the list of errors are used as inputs for the pattern matching step [3]. It analyzes the structure of the error to match it with its impacting metamodel change and decides which resolution to apply for the error co-evolution [4]. The metamodel changes provide the ingredients and necessary

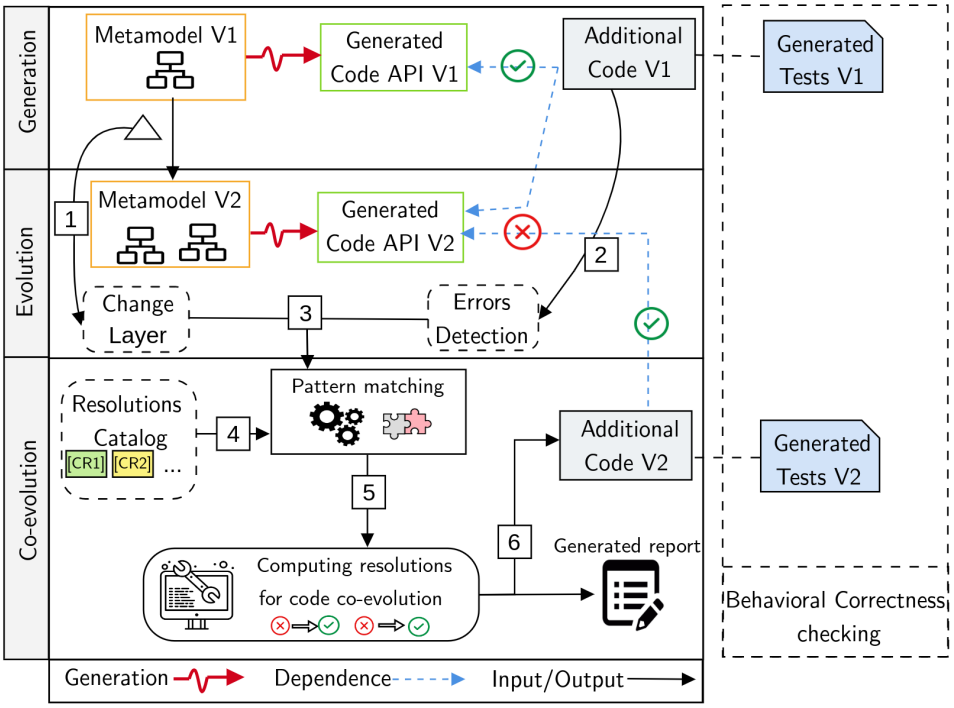

Figure 2.1 – Overall approach for metamodel and code co-evolution

information that are used for the co-evolution. At the end of the automatic co-evolution, we obtain a new co-evolved additional code 5. In addition to the automatic co-evolution, we generate test cases before and after co-evolution to highlight its possible effect. In fact, many research papers rely on the use of tests to check the behavior of the code during its evolution. For example, Godefroid et al. [21] uses tests to find regressions in different versions of REST APIs. In particular, Lamothe et al. [22],test [23] use tests to validate the evolution of the client code after Android API migration. We apply a similar method to check the effect of the co-evolution. [24] Finally, during the co-evolution process, we generate a report linking the applied resolutions for each code error with its impacting metamodel change. If needed, this can help developers in understanding the performed co-evolution, since we fully automate it.

BIBLIOGRAPHY

[1] Y.-T. Chen, C.-Y. Huang, and T.-H. Yang, « Using multi-pattern clustering methods to improve software maintenance quality », *IET Software*, vol. 17, *1*, pp. 1–22, 2023. DOI: https://doi.org/10.1049/sfw2.12075. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/sfw2.12075. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/sfw2.12075.

[2] M. Brambilla, J. Cabot, and M. Wimmer, *Model-driven software engineering in practice*. Morgan & Claypool Publishers, 2017.

[3] J. Cabot and M. Gogolla, « Object constraint language (ocl): a definitive guide », *in Formal methods for model-driven engineering*, Springer, 2012, pp. 58–90.

[4] M. Riedl-Ehrenleitner, A. Demuth, and A. Egyed, « Towards model-and-code consistency checking », *in 2014 IEEE 38th Annual Computer Software and Applications Conference*, IEEE, 2014, pp. 85–90.

[5] G. Kanakis, D. E. Khelladi, S. Fischer, M. Tröls, and A. Egyed, « An empirical study on the impact of inconsistency feedback during model and code co-changing. », *The Journal of Object Technology*, vol. 18, *2*, pp. 10–1, 2019.

[6] V. C. Pham, A. Radermacher, S. Gerard, and S. Li, « Bidirectional mapping between architecture model and code for synchronization », *in 2017 IEEE International Conference on Software Architecture (ICSA)*, IEEE, 2017, pp. 239–242.

[7] R. Jongeling, J. Fredriksson, F. Ciccozzi, A. Cicchetti, and J. Carlson, « Towards consistency checking between a system model and its implementation », *in International Conference on Systems Modelling and Management*, Springer, 2020, pp. 30–39.

[8] R. Jongeling, J. Fredriksson, F. Ciccozzi, J. Carlson, and A. Cicchetti, « Structural consistency between a system model and its implementation: a design science study in industry », *in The European Conference on Modelling Foundations and Applications (ECMFA)*, 2022.

[9] M. Zaheri, M. Famelis, and E. Syriani, « Towards checking consistency-breaking updates between models and generated artifacts », *in 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, IEEE, 2021, pp. 400–409.

[10] Y. Yu, Y. Lin, Z. Hu, S. Hidaka, H. Kato, and L. Montrieux, « Maintaining invariant traceability through bidirectional transformations », *in 2012 34th International Conference on Software Engineering (ICSE)*, IEEE, 2012, pp. 540–550.

[11] D. E. Khelladi, B. Combemale, M. Acher, O. Barais, and J.-M. Jézéquel, « Co-evolving code with evolving metamodels », *in Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20, Seoul, South Korea: Association for Computing Machinery, 2020, pp. 1496–1508, ISBN: 9781450371216. DOI: 10.1145/3377811.3380324. [Online]. Available: https://doi.org/10.1145/3377811.3380324.

[12] B. Selic, « The pragmatics of model-driven development », *IEEE Software*, vol. 20, *5*, pp. 19–25, 2003. DOI: 10.1109/MS.2003.1231146.

[13] J. Hutchinson, M. Rouncefield, and J. Whittle, « Model-driven engineering practices in industry », *in Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11, Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 633–642, ISBN: 9781450304450. DOI: 10.1145/1985793.1985882. [Online]. Available: https://doi.org/10.1145/1985793.1985882.

[14] J. Bézivin, « On the unification power of models », *Softw. Syst. Model.*, vol. 4, *2*, pp. 171–188, May 2005, ISSN: 1619-1366. DOI: 10.1007/s10270-005-0079-0. [Online]. Available: https://doi.org/10.1007/s10270-005-0079-0.

[15] E. Seidewitz, « What models mean », *IEEE software*, vol. 20, *5*, p. 26, 2003.

[16] M. Volter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[17] D. Gašević, N. Kaviani, and M. Hatala, « On metamodeling in megamodels », *in Model Driven Engineering Languages and Systems: 10th International Conference, MoDELS 2007, Nashville, USA, September 30-October 5, 2007. Proceedings 10*, Springer, 2007, pp. 91–105.

[18] J. de Lara and E. Guerra, « Domain-specific textual meta-modelling languages for model driven engineering », *in Modelling Foundations and Applications: 8th European Conference, ECMFA 2012, Kgs. Lyngby, Denmark, July 2-5, 2012. Proceedings 8*, Springer, 2012, pp. 259–274.

[19] B. Lientz and E. Swanson, *Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*. Addison-Wesley, 1980, ISBN: 9780201042054. [Online]. Available: https://books.google.fr/books?id=8a6gAAAAMAAJ.

[20] E. B. Swanson, « The dimensions of maintenance », *Proceedings - International Conference on Software Engineering*, pp. 492–497, 1976, ISSN: 02705257.

[21] P. Godefroid, D. Lehmann, and M. Polishchuk, « Differential regression testing for rest apis », *in Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2020, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 312–323, ISBN: 9781450380089. DOI: 10.1145/3395363.3397374. [Online]. Available: https://doi.org/10.1145/3395363.3397374.

[22] M. Lamothe, W. Shang, and T.-H. P. Chen, « A3: assisting android api migrations using code examples », *IEEE Transactions on Software Engineering*, vol. 48, *2*, pp. 417–431, 2022. DOI: 10.1109/TSE.2020.2988396.

[23] M. Fazzini, Q. Xin, and A. Orso, « Apimigrator: an api-usage migration tool for android apps », *in Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, ser. MOBILESoft '20, Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 77–80, ISBN: 9781450379595. DOI: 10.1145/3387905.3388608. [Online]. Available: https://doi.org/10.1145/3387905.3388608.

[24] N. Meng, M. Kim, and K. S. McKinley, « Lase: locating and applying systematic edits by learning from examples », *in Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, San Francisco, CA, USA: IEEE Press, 2013, pp. 502–511, ISBN: 9781467330763.

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : titre (en français).....

Mot clés : de 3 à 6 mots clefs

Résumé : Eius populus ab incunabulis primis ad usque pueritiae tempus extremum, quod annis circumcluditur fere trecentis, circummurana pertulit bella, deinde aetatem ingressus adultam post multiplices bellorum aerumnas Alpes transcendit et fretum, in iuvenem erectus et virum ex omni plaga quam orbis ambit inmensus, reportavit laureas et triumphos, iamque vergens in senium et nomine solo aliquotiens vincens ad tranquilliora vitae discessit. Hoc inmaturo interitu ipse quoque sui pertaesus excessit e vita aetatis nono anno atque vicensimo cum quadriennio imperasset. natus apud Tuscos in Massa Veternensi, patre Constantio Constantini fratre imperatoris, matreque Galla. Thalassius vero ea tempestate praefectus praetorio praesens ipse quoque adrogantis ingenii, considerans incitationem eius ad multorum augeri discrimina, non maturitate vel consiliis mitigabat, ut aliquotiens celsae potestates iras principum molliverunt, sed adversando iurgandoque cum parum congrueret, eum ad rabiem potius evibrabat, Augustum actus eius exaggerando creberrime docens, idque, incertum qua mente, ne lateret adfectans. quibus mox Caesar acrius efferatus, velut contumaciae quoddam vexillum altius erigens, sine respectu salutis alienae vel suae ad vertenda opposita instar rapidi fluminis irrevocabili impetu ferebatur. Hae duae provinciae bello quondam piratico catervis mixtae praedonum.

Title: titre (en anglais).....

Keywords: de 3 à 6 mots clefs

Abstract: Eius populus ab incunabulis primis ad usque pueritiae tempus extremum, quod annis circumcluditur fere trecentis, circummurana pertulit bella, deinde aetatem ingressus adultam post multiplices bellorum aerumnas Alpes transcendit et fretum, in iuvenem erectus et virum ex omni plaga quam orbis ambit inmensus, reportavit laureas et triumphos, iamque vergens in senium et nomine solo aliquotiens vincens ad tranquilliora vitae discessit. Hoc inmaturo interitu ipse quoque sui pertaesus excessit e vita aetatis nono anno atque vicensimo cum quadriennio imperasset. natus apud Tuscos in Massa Veternensi, patre Constantio Constantini fratre imperatoris, matreque Galla. Thalassius vero ea tempestate praefectus praetorio praesens ipse quoque adrogantis ingenii, considerans incitationem eius ad multorum augeri discrimina, non maturitate vel consiliis mitigabat, ut aliquotiens celsae potestates iras principum molliverunt, sed adversando iurgandoque cum parum congrueret, eum ad rabiem potius evibrabat, Augustum actus eius exaggerando creberrime docens, idque, incertum qua mente, ne lateret adfectans. quibus mox Caesar acrius efferatus, velut contumaciae quoddam vexillum altius erigens, sine respectu salutis alienae vel suae ad vertenda opposita instar rapidi fluminis irrevocabili impetu ferebatur. Hae duae provinciae bello quondam piratico catervis mixtae praedonum.