FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Model-Driven Engineering Approach for Reengineering CI/CD Pipelines

André Flores

WORKING VERSION

Mestrado em Engenharia Informática e Computação

Supervisor: Jácome Cunha

Co-supervisor: Hugo Gião

February 5, 2024

A Model-Driven Engineering Approach for Reengineering CI/CD Pipelines

André Flores

Mestrado em Engenharia Informática e Computação

February 5, 2024

Resumo

Continuous integration, delivery, and deployment (CI/CD) facilita o desenvolvimento colaborativo de software e melhora a qualidade do produto. A prática ganhou popularidade com a metodologia de *eXtreme Programming* (XP) e tem visto uma adoção crescente nos últimos anos.

Existem muitas tecnologias de CI/CD disponíveis para automatizar *workflows* de desenvolvimento, e cada uma delas possui vantagens e desvantagens, dependendo dos projetos de software individuais. Muitos projetos de software utilizam várias tecnologias CI/CD simultaneamente ou migram entre elas, à procura de funcionalidade, escalabilidade ou melhores preços. Essas tecnologias frequentemente têm uma curva de aprendizado elevada, principalmente devido à sua sintaxe, e os engenheiros de software relatam dificuldades ao migrar entre tecnologias. A migração de tecnologias também pode ser um processo demorado.

Este trabalho tem como objetivo aproveitar *model-driven engineering* (MDE) para facilitar a migração de pipelines CI/CD entre tecnologias. Isso será feito usando um meta-modelo detalhado de pipelines CI/CD para mapear funcionalidades comuns entre tecnologias.

Para começar, serão analisados repositórios de software no GitHub com mais de dez estrelas e uso de CI/CD para descobrir a evolução das tecnologias mais populares e as migrações mais comuns. Essa análise envolverá a recuperação de *snapshots* de cada repositório em intervalos de aproximadamente três meses desde 1 de janeiro de 2012 ou o primeiro *commit* do repositório, dependendo do que for mais recente. As tecnologias serão identificadas pelo uso ou conteúdo de arquivos específicos no repositório.

Os resultados desta análise serão usados para determinar que tecnologias e migrações de CI/CD são mais importantes suportar. Isto garantirá que este trabalho é relevante à prática de desenvolvimento de software, e fornecerá casos de avaliação.

O resultado final deverá ser software de migração que analisa *pipelines* CI/CD para um meta-modelo e pode gerar *pipelines* corretos para outras tecnologias a partir desse meta-modelo. Este software deve suportar as migrações de tecnologias mais populares.

Os engenheiros de software podem utilizar o software resultante para reduzir as dificuldades ao executar migrações e manter a produtividade.

Abstract

Continuous integration, delivery, and deployment (CI/CD) facilitates collaborative software development and improves product quality. The practice gained popularity with the eXtreme Programming (XP) methodology and has seen increased adoption in recent years.

There are many CI/CD technologies available to automate development workflows, and each of them has pros and cons depending on the individual software projects. Many software projects use multiple CI/CD technologies simultaneously or migrate between technologies, chasing functionality, scalability, or better pricing. These technologies often have a high learning curve, primarily due to syntax, and developers report difficulties when migrating between technologies. Migration can also be a lengthy process.

This work aims to leverage model-driven engineering (MDE) to facilitate the migration of CI/CD pipelines between technologies. This will be done using a detailed meta-model of CI/CD pipelines to map common functionality between technologies.

To start, software repositories on GitHub with more than ten stars and CI/CD use will be mined to discover the evolution of the most popular technologies and the most common technology migrations. This mining will consist of retrieving snapshots of each repository at intervals of roughly three months since either January 1st, 2012 or the first commit of the repository, depending on which is more recent. Technologies will be detected using the presence or the content of specific files in the repository.

The mining results will be used to ascertain which CI/CD technologies and migration are the most important to support. This will ensure the results of this work are relevant to the software development practice and provide cases for evaluation.

The result should be migration software that parses CI/CD pipelines to a meta-model and can generate correct pipelines for other technologies from that meta-model.

Software developers could use this software to lessen difficulties when executing migrations and maintain productivity.

"If I had more time, I would have written a shorter letter."

Cicero

Contents

List of Figures

List of Tables

List of Listings

Abbreviations and Symbols

AADL	Architecture Analysis and Design Language
API	Application Programming Interface
ATL	Atlas Transformation Language
BPM	Business Process Model
CAMEL	Cloud Application Modelling Execution Language
CD	Continuous Delivery/Deployment
CI	Continuous Integration
CLI	Command Line Interface
CMT	Configuration Management Tool
CPS	Cyber-Physical System
DevOps	Development & Operations
DSL	Domain-Specific Language
DSML	Domain-Specific Modeling Language
EDMM	Essential Deployment Meta-Model
EMF	Eclipse Modeling Framework
EMOF	Essencial MOF
FMI	Functional Mock-up Interface
FuSaFoMo	Functional Safety Formal Model
GAI	GitHub Actions Importer
GHA	GitHub Actions
GMF	Eclipse Graphical Modeling Framework
GPML	General-Purpose Modeling Language
GUI	Graphical User Interface
IDE	Integrated Development Environment
IoT	Internet-of-Things
JSON	JavaScript Object Notation
LCEP	Low-Code Engineering Platform
MBE	Model-Based Engineering
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Model-Driven Engineering
MDSE	Model-Driven Software Engineering
MLS	Modeling Language Suite
MOF	Meta Object Facility
MTT	Model-to-Text
NPM	Node Package Manager
OCL	Object Constraint Language
OMG	Object Management Group

OSS	Open-Source Software
PIM	Platform-Independent Model
PSM	Platform-Specific Model
QA	Quality Assurance
QVT	Query/View/Transformations
SaaS	Software-as-a-Service
SCS	Safety-Critical System
SDLC	Software Development Life Cycle
SLR	Systematic Literature Review
RQ	Research Question
TCS	Textual Concrete Syntax
TIM	Technology-Independent Model
TOSCA	Topology and Orchestration Specification for Cloud Application
TSM	Technology-Specific Model
UML	Universal Modeling Language
VM	Virtual Machine
XP	eXtreme Programming

Chapter 1

Introduction

1.1 Context

Continuous Integration, Delivery, and Deployment (CI/CD) means that changes to a program's code are consistently integrated into the current system and deployed to a production environment with little delay. These changes should only be integrated if, after adding the code, the system can be built from scratch and pass all required tests [11, 63].

Practicing CI/CD implies a set of synergistic activities like frequent code commits and builds, automated building and testing, immediately fixing a broken build, etc. [43, 36, 118, 114, 5]. These activities are organized into CI/CD pipelines [108, 126, 59, 12, 135]. Listing 1.1 is an example of a high-level definition of such a pipeline.

```
on pull request creation:
    - run static analysis on code
    - build project
    - run unit tests
    - run acceptance tests
    - output static analysis, unit and acceptance test results, and unit test
     ↪  coverage report
```

Listing 1.1: High-level example of a CI/CD pipeline.

CI/CD brings several benefits. These include reduction of cost and risk of work integration in distributed teams, increased software reliability, reduced time to market, improved customer satisfaction, and enhanced productivity [43, 118, 36, 24, 5].

CI/CD pipelines are implemented using CI/CD technologies. These CI/CD technologies support the automated building, testing, and deployment of software. They also offer other features like integration with the code-hosting platform and package marketplaces where users can search for extensions that improve functionality or usability (differing technologies have different terminology for packages, e.g. Plugins in Jenkins, Actions in GitHub Actions (GHA), Orbs in CircleCI) [28, 41, 82].

```yaml
name: 'Link Checker: All English'

# The `on` key lets you define the events that trigger the workflow.
on:
  push:
    branches:
      - main

# The `jobs` key groups together all the jobs that run in the workflow file.
jobs:
  check-links:
    runs-on: ubuntu-latest
    # The `steps` key groups together all the steps that will run as part of the
    ↪ job.
    steps:
      # The `uses` key tells the job to retrieve the action named
      ↪ `actions/checkout`.
      - name: Checkout
        uses: actions/checkout@v4

      - name: Gather files changed
        uses: trilom/file-changes-action
        with:
          fileOutput: 'json'

      # The `run` key tells the job to execute a command on the runner.
      - name: Link check (warnings, changed files)
        run: ./script/rendered-content-link-checker.mjs --language en --max 100
        ↪ --check-anchors --check-images --verbose --list $HOME/files.json
```

Listing 1.2: Example of a CI/CD pipeline in GHA. Adapted from GitHub [125].

There are many CI/CD technologies available on the market. The Cloud Native Computing Foundation's curated list of CI/CD technologies numbers sixty-one, and it is not complete [120]. Recently, more CI/CD technologies have been emerging [108, 114].

Often, CI/CD technologies are configured as code in a configuration file written in their domain-specific language (DSL), as seen in listing 1.2. These configuration files are stored with the rest of the project's code.

1.2 Problem Definition

A project is not limited to using just one CI/CD technology at a time (co-usage) or always sticking to the same technology. Over its lifetime, a project can change the CI/CD technologies it uses. This process, referred to as migration, involves taking the CI/CD pipeline modeled in the current CI/CD technology/technologies and translating it into the new one(s).

There are diverse motives for migrating CI/CD technologies. In their study of the reasons for the changes in the CI/CD landscape [108], Mazrae et al. interview twenty-two respondents with experience setting up, managing, or maintaining the CI/CD process of projects. They query their interviewees on what motivated CI/CD technology migrations in their projects. They found that the interviewees were motivated by the following:

- Having fewer restrictions on the free tier of the CI/CD technology
- Using a more reliable CI/CD technology
- Better integration with the hosting platform
- Better support for multiple platforms
- Decreasing the amount of CI/CD technology co-usage
- Having better features
- Moving to a successor CI/CD technology
- Making the project open source
- Moving to a new ecosystem
- Reducing the maintenance burden

The study of GitHub repositories detailed in chapter 3 reveals a significant change rate in the CI/CD technology stacks of projects over time, meaning migrations are commonplace. Furthermore, it also finds a significant amount of co-usage of CI/CD technologies. Considering that one of the main motivators for migration is decreasing the amount of co-usage, this could mean many projects are looking to migrate.

Migrations are often hard to execute. CI/CD technologies have a high learning curve, there are fundamental differences between technologies, configuring CI/CD is trial-and-error by nature, and some features may be missing for CD. The syntax of the new tool is highlighted as a problem [108]. CI/CD implementations for complex projects can take around five weeks to migrate. If the implementation must support multiple projects, the timeline can shift to months [64].

While migration is taking place, projects can experience reduced productivity due to not only the effort being expended in the migration but also because CI/CD processes can break down during the process. This hardship in migrating CI/CD can also lead projects to stay with a given technology after it stops being optimal for their use case. There is a need to support developers in migrating CI/CD.

1.3 Objectives

This work's objective is to support developers in migrating CI/CD technologies. The result should be to make migration easier and faster, keeping up productivity and reducing lock-in to a technology. This is part of an ongoing effort to improve developers' experience with CI/CD by leveraging model-driven engineering to interact with CI/CD in a technology-independent manner.

In doing so, this work should answer the following research questions (RQs).

RQ1. Is it possible to model CI/CD pipelines in a technology-independent manner with low-level detail?

RQ2. Can such a model allow accurate translation of CI/CD pipelines between technologies?

RQ3. What pipeline conversion rate between technologies can be achieved?

1.4 Document Structure

The rest of this document is structured as follows.

Chapter 2 gives further background information on CI/CD and presents current migration support given by CI/CD technology providers. Furthermore, it also details model-driven engineering concepts relevant to this work. Lastly, it includes related work.

Chapter 3 details a study of around 600,000 GitHub repositories that was done to discover trends in CI/CD usage over time. It focuses on the most popular technologies, co-usage of technologies, and migrations between technologies.

Chapter 4 details the solution proposal and the work plan to implement it.

Chapter 5 concludes with a reflection on the state of CI/CD and CI/CD technology migrations and how model-driven engineering can and has been used in the CI/CD landscape.

Chapter 2

State of the Art

This chapter details the current state of CI/CD and of Model-Driven Engineering.

Section 2.1 goes into the emergence of CI/CD and its relevance in software engineering.

Section 2.2 lays out support given by CI/CD technology providers on migrating CI/CD technologies. It includes both automated and manual migration support.

Section 2.3 gives background information on Model-Driven Engineering. It includes the motivation for using modeling in software engineering, details on models and model transformations, and how models and model transformations can be used in software reengineering.

Section 2.4 includes related work regarding Model-Driven Engineering and CI/CD.

2.1 Background on CI/CD

In the late 90s, Kent Beck proposed the eXtreme Programming (XP) methodology for software development to address shortcomings of the Waterfall Model [11].

The Waterfall Model, the more traditional approach to software development that was first described in 1970 [109, 101], is static and approaches software development linearly and sequentially, completing one activity before the other [4]. It can be said to involve four phases: requirement analysis, design, implementation, testing, and operation and maintenance [4].

Due to its rigid nature, the use of Waterfall generates well-known problems. These include but are not limited to reduced ability to deal with change, increased rework, and unpredictable software quality due to late testing [101, 115].

According to Beck, Waterfall arose from the measurement that the cost of changing software increased dramatically over time. However, the software community made strides to reduce this change cost by introducing relational databases, modular programming, and information hiding. With this in mind, there was no need for the software engineering community to be beholden to Waterfall [11].

As an Agile process [92], XP embraces change throughout the software development lifecycle (SDLC). To achieve this, XP uses shorter development cycles. XP also “turns the conventional

software process side-ways", executing the four constituent phases of Waterfall a little at a time during the development cycle instead of sequentially [11].

XP aims to increase communication, improve software quality, improve customer feedback, and create smaller and more frequent software releases, shortening time to market [11].

When Beck presented XP, he outlined thirteen constituent practices: planning game, small releases, metaphor, simple design, tests, refactoring, pair programming, collective ownership, on-site customer, 40-hour work weeks, open workspace, just rules, and continuous integration (CI) [11].

Later, Humble and Farley extended the philosophy behind CI into software deployment. Continuous Delivery/Deployment (CD) expresses the steps to deploy software as a deployment pipeline. This pipeline automates the steps that take successfully integrated code and put a new software version into production, increasing release frequency [5, 63]. Together, CI and CD form CI/CD [5].

Circa 2007, DevOps was introduced [97]. A portmanteau of Development and Operations, DevOps is a development methodology that bridges the gap between these two areas by emphasizing communication and collaboration, CI/CD, and quality assurance (QA) [65]. CI/CD pipelines are at the core of DevOps [10].

Agile practices like XP have seen a significant adoption rate since they were introduced [80, 95] and DevOps's importance to organizations has been increasing [97]. In its annual reports on the state of DevOps, Puppet has found that high-performing organizations enabled by DevOps deploy code thirty times faster than their lower-performing peers [98]. They also have fewer failures on deployments and recover from failure much faster [99, 100]. Moreover, CI/CD has also become an essential part of cloud-computing [5].

With XP's and DevOps's increasing popularity, CI/CD has seen greater relevance in software development for companies or open-source software (OSS) communities since it ensures integrity and control over changes made to the software project [114, 108, 60].

2.2 Migration Support from CI/CD technologies' providers

This section details support for CI/CD migrations that is currently offered to practitioners, whether this support is in the form of an automated migration tool (section 2.2.1) or manual migration guides (section 2.2.2).

Of the CI/CD technologies listed by the Cloud Foundation [120], only the ones that had ever achieved over 1% market share (fig. 3.5) in the study detailed in chapter 3 were analyzed. These number eleven: AppVeyor, CircleCI, Codefresh, Concourse, Drone, GHA, Gitlab CI/CD, GoCD, Jenkins, Kubernetes, and Travis CI.

2.2.1 Automated Migrations

Only one of the aforementioned CI/CD technologies has an available tool for automated migration.

GitHub Actions Importer

GitHub provides a tool called GitHub Actions Importer (GAI) to plan and automate migrations to GHA [8]. Its goal is to achieve an 80% conversion rate for every workflow, but this depends on the makeup of each pipeline. The tool is an extension to the GitHub Command Line Interface (CLI) and runs on a Docker container.

GAI supports migration from seven CI/CD technologies: Azure DevOps, Bamboo, Bitbucket Pipelines, CircleCI, GitLab, Jenkins, and Travis CI. The user connects GAI to the existing CI/CD technology platform by supplying access credentials.

GAI can audit existing pipelines (to determine how complete and complex an automated migration can be), forecast the usage time of GHA by the transformed pipeline, dry-run a migration, and create a pull request with the migrated pipeline in the GitHub repository.

GAI's functionality can be extended with the use of custom transformers. The transformers can convert items that GAI does not automatically convert or modify how items are converted. They can also convert references to runners, virtual machines (VMs) where a pipeline is executed, and environment variables. Transformers are defined in a DSL built on top of Ruby.

There are limitations to GAI's functionality. These vary with the CI/CD being migrated from but are usually related to functionality that cannot be mapped one-to-one in GHA, unknown packages being used in the original pipeline, unsupported functionality in GHA, and secret environment variables.

2.2.2 Manual Migrations

Of the eleven providers analyzed, only four provide guidance on migrating to their CI/CD technology. These are: CircleCI, Codefresh, GHA, and Gitlab CI/CD.

The guides mostly center on comparing syntaxes between migrating technologies. These comparisons focus on basic common functionality or key differences between technologies and rarely provide help for more complex CI/CD pipelines [64, 52, 67, 83, 102]. Fig. 2.1 is an example of such guidance.

Cases where the technology providers offer more detailed guides are laid out below. GHA does not provide any support more detailed than what was already specified.

CircleCI

CircleCI provides a detailed methodology on migrating CI/CD technologies [64]. The method includes various phases: assessment, planning, preparation, testing, and migration. They provide a rough time frame for migration: less than one week for a simple project and around five weeks for a complex one.

Figure 2.1: Example of syntax comparison guide from a CI/CD technology company [89].

Codefresh

Codefresh offers a superset of Jenkins capabilities [67]. Codefresh has a detailed guide on migrating from Jenkins pipelines. This guide includes feature, architecture, and installation comparisons between the two technologies and general advice on creating Codefresh pipelines. There are detailed instructions on migrating Jenkins freestyle jobs, pipelines, credentials, pipelines that create Docker images, pipelines that deploy to Kubernetes, shared libraries, checking out source code, and step conditions. There is also a guide on co-usage with Jenkins while the migration is happening. This guide includes instructions on how to run Codefresh pipelines from Jenkins jobs and how to run Jenkins jobs from Codefresh pipelines.

Codefresh can also run Actions available in the GitHub Actions Marketplace [52]. This could ease the transition process to GHA.

Gitlab CI/CD

Gitlab gives pointers to manage organizational change and technical questions to consider before a migration [102].

Advice on organizational changes includes setting and communicating clear migration goals, ensuring alignment from the relevant leadership teams, educating users on changes, finding ways to sequence or delay parts of the migration, not leaving the CI/CD pipeline in a partially-migrated state for too long, and not moving the CI/CD pipeline as-is, instead taking advantage of new functionality and updating the implementation.

The suggested technical questions center on the number of projects using the pipeline, the git branching strategy, the tools used to build and test code, security scanners, and deployment.

2.3 Model-Driven Engineering

This section lays the background for Model-Driven Engineering (MDE). It details what modeling is (section 2.3.1), its presence in software engineering (section 2.3.2), Model-Driven Software Engineering (section 2.3.3), modeling languages (section 2.3.4), meta-modeling (section 2.3.5), model transformations (section 2.3.6), Model-Driven Software Reengineering (section 2.3.7), and technologies used in MDE (section 2.3.8).

2.3.1 From Abstraction to Modeling

Abstraction is a natural behavior of the human mind. It can be defined as the capability of finding the commonality in many different observations and thus generating a mental representation [19].

To be able to abstract is to simultaneously [19]:

- generalize specific features of real objects (generalization)
- classify objects into coherent clusters (classification)
- aggregate objects into more complex ones (aggregation)

In science and technology, abstraction is often referred to as modeling. A model is "a simplified or partial representation of reality, defined in order to accomplish a task or to reach an agreement on a topic". This also means a model will never describe reality in its entirety [19]. Notable models in science include the Bohr model of the atom [16, 19].

Models perform at least two roles regarding abstraction [19]:

- **Reduction** - models only reflect a portion of the original object's properties
- **Mapping** - models are based on an original object, which is taken as a prototype of a category of individuals and is abstracted and generalized to a model

Models can also be classified regarding their purpose [19]. They can be:

- **Descriptive** - for describing the reality of a system or context
- **Prescriptive** - for determining the scope and detail at which to study a problem
- **Defining** - for defining how a system shall be implemented

Models are meant to describe two main dimensions of a system: the static (or structural) and the dynamic (or behavioral). *Static models* focus on the static parts of the system, its managed data, and architecture. *Dynamic models* describe the behavior of the system by showing the execution

sequence of actions and algorithms, collaborations among system components, and changes to the internal state of components and applications [19].

Since the human mind can process nothing without abstraction, it can be said that "everything is a model" [19].

2.3.2 Modeling in Software Engineering

Modeling can be used to develop software artifacts, and it has been shown to increase efficiency and effectiveness in software development [3, 19].

The need for using models in software engineering is driven by [19]:

- the complexity of software artifacts and the need to discuss them at different levels of abstraction
- the pervasiveness of software in people's lives and the expected need for more software that comes with it
- the continuous shortage of software developers
- the need for software developers to interact with non-developers, which requires some mediation of the technical aspects of development

From this, Fowler derives three use cases for models [15, 19]. They can be:

- **Sketches** - when used for communication purposes, only specifying partial views of the system
- **Blueprints** - when used to provide a complete and detailed specification for the system
- **Programs** - when used to develop the system, instead of code

2.3.3 Model-Driven Software Engineering

Model-Driven Software Engineering (MDSE), or simply Model-Driven Engineering (MDE), is a "methodology for applying the advantages of modeling to software engineering activities" [19]. According to Sendall and Kozaczynski, MDE's objective is "to increase productivity and reduce time-to-market by enabling development and using concepts closer to the problem domain at hand" [112].

MDE's core concepts are models and transformations (manipulation operations on models). According to Brambilla et al., if one were to adapt Niklaus Wirth's equation [136], $Algorithms + DataStructures = Programs$, to the MDE context, it would read: $Models + Transformations = Software$ [19].

The modeling language is MDE's notation, and the kinds of models defined vary with the software produced. To practice MDE, one must use an integrated development environment (IDE)

that supports defining models and transformations and compilers or interpreters that can make the final software artifacts [19].

There can be some ambiguity in the terminology associated with MDE. This work uses the following terminology, defined by Brambilla et al. [19]:

- **Model-Driven Development (MDD)** - a development paradigm that uses modeling as the primary artifact of the development process. In MDD, the implementation is (semi-)automatically generated from the models.
- **Model-Driven Architecture (MDA)** - a vision of MDD proposed by the Object Management Group (OMG), can be seen as a subset of MDD.
- **Model-Driven Engineering (MDE)** - a superset of MDD, goes beyond pure development activities and encompasses other model-based tasks of a complete software engineering process. Strictly adheres to "everything is a model".
- **Model-Based Engineering (MBE)** - a "softer" version of MDE where models play an important role, although they are not necessarily the key artifacts of the development process

2.3.4 Modeling Languages

As was previously stated, modeling languages are one of the principal components of MDE. A modeling language lets designers specify the models for their systems in graphical or textual representations. Modeling languages are formally defined, and designers must comply with their syntax when modelling [19].

Modeling languages can be classified as [19]:

- **Domain-Specific Modeling Languages (DSMLs)** - designed specifically for a certain domain, context, or company to ease the description of things in that domain
- **General-Purpose Modeling Languages (GPMLs)** - represent tools that can be applied to any sector or domain for modeling purposes

It is also possible to distinguish modeling languages according to their level of abstraction.

The distinction between static and dynamic models also highlights the importance of having different viewpoints on the same system. Multi-viewpoint modeling is a cornerstone of MDE, leading to the building of various models to describe the same system. These multiple models may use different modeling languages. Although it is possible to define a design composed of models in several independent languages, it is more convenient to explore a suite of languages that have a common foundation and are aware of each other. Thus, GPMLs typically include several coordinated notations that complement each other. These GPMLs are known as Modeling Language Suites (MLSs). The most known example of an MLS is the Universal Modeling Language (UML).

2.3.5 Meta-modeling

A natural extension of the definition of objects as instances of a model is to define the models themselves as instances of more abstract models. The more abstract models, called meta-models, highlight the properties of models themselves and, in a practical sense, constitute the definition of a modeling language, as they provide the capability of describing the whole class of models that can be represented by that language [19].

Following this chain of reasoning, one can create models that describe objects, meta-models that describe models, and meta-meta-models that describe meta-models. While it is theoretically possible to define infinite levels of meta-modeling, it has been shown that meta-meta-models can be defined based on themselves, providing little benefit to going beyond this level of abstraction [19].

While, when referring to an object in MDE, we say that an object is an instance of a model, when referring to a model, we say it conforms to a meta-model [19].

Meta-models can be used for [19]:

- defining new languages for modeling or programming
- defining new modeling languages for exchanging and storing information
- defining new properties or features to be associated with existing information (metadata)

This work will use a meta-model to represent CI/CD pipelines in a platform-independent manner, providing an abstraction from the CI/CD pipeline implementations of specific technologies' platforms.

2.3.6 Model Transformations

Besides models, transformations are the other crucial ingredient of MDE. They allow for the definition of mappings between different models [19]. According to Sendall and Kozaczynski, model transformations are "the heart and soul of model-driven software development" [112].

While transformations are applied at the model level, they are defined at the meta-model level.

Transformations can be further classified as model-to-model (MTM) transformations, model-to-text (MTT) transformations, and text-to-model (TTM) transformations.

Model-to-Model Transformations

Generally, MTM transformations take one or more models as input and return one or more models as output. In most cases, one-to-one transformations are sufficient, with one input and one output model. However, there are also situations where many-to-one, one-to-many, or many-to-many transformations are required, like merging models [19].

MTM transformations can also be classified as endogenous or exogenous. Endogenous transformations are transformations between models conforming to the same meta-model, while exogenous transformations are between models conforming to different meta-models [85, 19]. Endogenous transformations are also referred to as *rephrasing*, and exogenous transformations as *translating* [85, 127].

Examples of endogenous transformations include optimization, refactoring, simplification/normalization, and component adaptation. Exogenous transformations can be synthesis, reverse engineering, or migration [85].

Endogenous MTM transformations can be made *in-place*, where the input and output model are the same, or *out-place*, where the output model is created from scratch. By definition, exogenous transformations can only be *out-place* [85].

Model-to-Text Transformations

Model-to-Text (MTT) transformations automate text derivation from models [19].

The primary purpose of MTT transformations in MDE is code generation. MDE's objective is to obtain a working system out of models. Since current execution platforms are mostly code-based, this implies transitioning the model to code level. MTT transformations can also generate code other than the one that describes the system, like test cases or deployment scripts [19].

When implementing MTT transformations, there are several points to consider. MTT transformations vary in how much code is generated since it may not be possible to generate the code entirely from the model, what code is generated, generally code in high-level languages is preferable for better readability, and how the code is generated since there are multiple technologies to accomplish this [19].

Code generation can be described as a vertical transition from models with a higher level of abstraction to lower-level artifacts. These different levels of abstraction imply a gap that must be filled since not all specifics of the underlying technologies may be representable in the models [19].

This missing information has to be filled by the modeler using model augmentations, applying the convention-over-configuration principle for code generation, or leaving the specification open on the model level and entering the details at the code level [19].

These three approaches each have their pros and cons. Model augmentations allow detailed tweaking of the derived implementations by spending more effort preparing the models for code generation. The convention-over-configuration approach forgoes this effort, but the derived implementations can only be optimized at the code level. There is space for a hybrid of the model augmentation and convention-over-configuration approaches. The last approach implies only partial implementation generation, meaning the developer must complete the code themselves, which could lead to inconsistencies [19].

Text-to-Model Transformations

Text-to-Model (TTM) transformations have text as an input and a model as an output [19].

TTM transformations are used in reverse engineering, obtaining a higher-level system representation. Using them, it is possible to parse text, like code, into a model [19].

2.3.7 Model-Driven Software Reengineering

Chikofsky and Cross define software reengineering as "the examination and alteration of software systems to reconstitute it in new form and the subsequent implementation of the new form" [26, 81]. The process starts with the source code of the current system and ends with the source code of the new system. This process can involve just translating code from one language into another or also redesigning and determining the requirements in legacy systems, comparing them to new system requirements, and removing unneeded elements [81, 107].

Software reengineering involves a set of subpractices, namely, forward and reverse engineering, re-documentation, restructuring, and translation [131, 81]. Reverse engineering is "the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction" [107]. Forward engineering is the traditional approach to software development, starting with the conceptual design of a system and moving down through the abstraction levels until we have an implementation [81, 107].

In essence, software reengineering implies abstracting a system's implementation into a higher-level representation of that system (reverse engineering), applying transformations to that representation, and refining that representation into a new implementation (forward engineering). These steps map well to previously discussed MDE concepts: the higher-level system representation is a model, reverse engineering is a TTM transformation, changes to the model are MTM transformations, and forward engineering is an MTT transformation. MDE approaches to software reengineering have been gaining traction, as they can automate a significant part of the process [42, 14, 104, 96, 124].

This work will involve reengineering CI/CD pipelines using as a basis the CI/CD pipeline meta-model mentioned above. TTM transformations will populate instances of the meta-model with a CI/CD pipeline, and MTT transformations will allow code generation in a different CI/CD technology.

2.3.8 Model-Driven Engineering Technologies

MDE relies on technologies that support the development of models, their transformation, and their integration in the SDLC [19]. In [66], Jácome-Guerrero et al. list various MDE technologies and categorize them according to the elements of MDE they address. Those are listed in the following sections.

Meta-modeling languages

- **Meta Object Facility (MOF)** - a language intended to model classes provided by the OMG. Due to its complexity, a subset of MOF is understood to be enough for most use cases [87].

- **Essencial MOF (EMOF)** - subset of MOF with less complexity also provided by the OMG [87].
- **Ecore** - a part of the meta-modeling architecture of the Eclipse Modeling Framework (EMF), which provides the Eclipse Integrated Development Environment (IDE) with meta-modeling capabilities. Ecore is a meta-model language that provides object-oriented concepts for creating meta-models and a subset of MOF [37].
- **Object Constraint Language (OCL)** - used to define rules to determine if a model is well-formed. It can be used with any MOF meta-model [94, 128].
- **MetaDepth** - a modeling language that supports an arbitrary number of meta-levels. This makes it useful for defining multi-level languages [86].

Development Environments

- **OMG Meta-modeling Architecture** - includes both MOF and UML since UML is defined with MOF [87].
- **Eclipse/Eclipse Modeling Framework** - the modeling project intended to support MDE in Eclipse. The EMF provides the basic mechanisms to handle meta-models [55].

Graphical Editors

- **Eclipse Graphical Modeling Framework (GMF)** - an Eclipse plugin that supports the development of graphical model editors from meta-models [17].
- **Graphiti** - used to create highly sophisticated editors with support for EMF [132].
- **Sirius** - an Eclipse project that allows the development of graphical modeling editors, leveraging Eclipse technologies like EMF and GMF. It is highly adaptable and supports blended modeling. Relies on Acceleo and other projects to facilitate establishing relationships between model data and its graphical representation [113].

Textual Editors

- **Xtext** - an EMF-based framework to create textual modeling languages. It associates a textual representation with the meta-model. It provides mechanisms that allow the editing and manipulation of textual models. Includes a language for defining grammar and an application programming interface (API) for defining different aspects of a DSL. Automatically generates a parser, static analyzer, code formatter, code generator, etc., for the defined DSL [140].
- **EMFText** - an Eclipse plugin that allows defining a textual syntax for an Ecore meta-model. It can generate code without any EMFText dependencies. Allows automatic generation of default syntax and complete analysis of syntax to warn of potential problems [39].

- **Textual Concrete Syntax (TCS)** - a component of Eclipse Generative Modeling Technologies that allows the specification of a textual syntax for DSL, attaching syntactic information to meta-models. Offers an Eclipse editor that supports syntax highlighting, a schema, and hyperlinks per each DSL syntax represented in the DSL [122].

Model Transformations

- **Atlas Transformation Language (ATL)** - a hybrid model transformation language that implements imperative and declarative paradigms. An ATL transformation is composed of rules defining how the target model elements are created and initialized from the elements of the source model [122].
- **Query/View/Transformations (QVT)** - a family of languages the OMG provides that allow for the definition of transformations. There are two end-user languages, QVT Operational Mappings and QVT Relations, and a low-level language, QVT Core [1].

Code Generation Languages

- **Acceleo** - an implementation of OMG's MOF-to-Text standard, part of the Eclipse M2T project. Allows easy code generation, high personalization capacity, interoperability, and traceability management. Uses a template mechanism [2].
- **Java Emitter Templates** - a tool to generate output files from an input model using templates [130].
- **Xpand** - can generate code based on DSL models defined with Xtext. It is a statically typed template language [129].

2.4 Related Work

Gião et al.'s systematic literature review (SLR) of model-driven approaches to DevOps [45] finds relevant papers and categorizes them according to the phases of DevOps they cover. These phases, proposed by Zhu et al. [143], are Development, Integration, Testing, Monitoring, Feedback, Deployment, and Operations. More detail about the phases can be found in tab. 2.1.

From the relevant papers identified by Gião et al., only the ones about the Integration, Testing, Deployment, and Operation phases were analyzed. This was because attempts to model the other DevOps phases, Development, Monitoring, and Feedback, were irrelevant to this study's focus, CI/CD pipelines. The selected studies are the following, [110, 38, 72, 116, 31, 62, 58, 84, 13, 71, 22, 18, 30, 29].

Most of the selected studies pertain to a specific domain [45]: data science [84, 71], cloud [110, 74], Internet-of-Things (IoT) [38, 116, 72], blockchain [58], big data [22], machine learning [13], safety-critical systems (SCSs) [88], and cyber-physical systems (CPSs) [31, 62]. Three studies are domain-agnostic [29, 30, 18, 106].

Table 2.1: DevOps phases proposed by Zhu et al. [143].

Phase	Description
Development	Involves planning and developing software.
Integration	Core of the DevOps lifecycle. Committing new changes to the source code.
Testing	Automatic testing tools.
Monitoring	Performance monitoring and recording of the application.
Feedback	Gathering, analyzing, and using clients' software usage feedback.
Deployment	Deployment of the code to the production environment.
Operations	Automating all release operation processes.

A snowballing effort from the selected studies led to the discovery of more related works. These are [139, 133, 25, 34, 7, 20, 79, 32, 61, 40, 9, 70, 105]. The inclusion criteria were works related to applying a model-driven approach to DevOps or CI/CD.

The following sections summarize these studies and explain why they are or are not relevant to this work. Section 2.4.1 details attempts to model CI/CD in a more general manner, section 2.4.2 details attempts to model CI/CD elements more specifically, and section 2.4.3 details papers that, while about both CI/CD and MDE, are not relevant to this work.

2.4.1 General Approaches

El Khalyly et al. [38] propose a DevOps meta-model and an IoT meta-model. The DevOps meta-model has a high degree of abstraction, while the IoT meta-model is more detailed. The authors claim a dependency from the IoT meta-model to the DevOps meta-model, as DevOps tools are "in service of Internet of Things ecosystem to guarantee the continuous integration, delivery and deployment of programs". The DevOps meta-model proposed by El Khalyly et al. has too much abstraction to be helpful to this work.

Bordeleau et al. [18] identify requirements of a modeling framework for DevOps through a case study. This framework would be composed of processes, methods, and tools. They identify general, description, analysis, and simulation (to support continuous framework improvement) requirements. This work is relevant as several requirements can be directly applied to CI/CD pipelines.

Colantoni et al. [29] propose DevOpsML, a modeling language for DevOps platforms and processes to support documentation implemented with EMF. In their work, they recognize the increasing interest in the integration of DevOps and MDE practices in low-code engineering platforms (LCEPs) and the tension between the often non-technical LCEP users and the current DevOps processes that are considered on a more technological level.

DevOpsML uses three meta-models:

- **Platform Meta-model** - a static model used to define platforms by their tools and interfaces (ex: graphical user interface (GUI), CLI), capabilities and concerns, and common concepts (ex: status, DevOps phase).

- **Process Meta-model** - a dynamic model that describes DevOps processes like CI/CD pipelines. The prototype of this meta-model is implemented with the Software and Systems Process Engineering Meta-model (SPEM) since it specifies the minimal required set of modeling language capabilities. SPEM has a higher descriptive capability than other solutions, which are better for executability.
- **Linking Meta-model** - capable of linking process to process, platform to platform, and process to platform.

This work is related, as Colantoni et al. create a meta-model that can describe the DevOps process. However, DevOpsML differs from the meta-model used in this work because it tries to model DevOps itself instead of CI/CD pipelines. This increased abstraction and scope would make it harder to directly execute transformations from platform-specific models (PSM) to the platform-independent model (PIM). This is not an issue for Colantoni et al., as the initial version of DevOpsML is intended to support documentation.

Kumar and Goyal [74] propose ADOC, a conceptual model for automated DevSecOps, DevOps embedded with security controls providing continuous security assurance, for OSS over the cloud. ADOC is based on a continuous security conceptual framework described in the article. There are three components to ADOC: the ADOC Engine, an end-to-end automated workflow with a set of practices and embedded security assurance controls; the OSS suite, the propellant for this ADOC Engine; and the cloud infrastructure and technologies to power ADOC. This work is related as Kumar and Goyal propose a model describing a specific branch of DevOps, DevSecOps.

Ferry et al. [40] present ENACT, a model-driven DevOps framework for trustworthy smart IoT systems. ENACT includes a continuous delivery toolkit with two enablers: an orchestration and continuous deployment enabler and a test emulation and simulation enabler. The first enabler has a DSML that can support the automatic deployment of software components over IoT, edge, and cloud resources. This work is related because of the attempt to generate deployment code from a model automatically.

TOSCA [93, 133], short for Topology and Orchestration Specification for Cloud Applications, is an emerging standard. Its main goal is to enhance the portability and management of cloud applications. It can be used to model and automate DevOps for cloud applications, as Wettinger et al. did in [133]. TOSCA is related because of this and because it is used or referenced in other related works.

MODAClouds [34] is a European project undertaken to simplify cloud services using MDE. One of its goals is to support developers in building and deploying applications to multi-clouds across the full cloud stack. MODAClouds includes MODACloudML, a set of tool-supported DSLs to support the design of multi-cloud applications with guaranteed quality of service. MODAClouds is related as it is a model-driven approach to CD in cloud applications. It is also used in other related works.

MELODIC [61] allows modeling, deploying, and optimizing multi-cloud applications. The application is modeled in the Cloud Application Modelling Execution Language (CAMEL) and

Business Process Models (BPMs). The CAMEL application model is transformed into a Constraint Programming Model for mathematical optimization of the deployment [30]. MELODIC is related as it can model the deployment of cloud applications.

MORE [25] is a model-driven approach to automate a system's initial deployment and dynamic configuration. MORE includes a model to specify the high-level view of a system in the form of a desired deployment topology. This topology is then transformed into executable code for Puppet, an infrastructure deployment and management platform [103], to get virtual machines, physical machines, and containers [110]. MORE is related as it is a model-driven approach to CD in cloud-based IT systems.

Babar et al. [9] model DevOps deployment choices to enable enterprises to devise a DevOps approach suitable to their requirements while considering possible process reconfigurations. Their approach enables the modeling of trade-offs of alternative deployment options [18]. Babar et al.'s work is related as it allows modeling CI/CD, but only in a very high-level manner.

Wurster et al. [139] propose the Essential Deployment Meta-Model (EDMM) to enable a common understanding of declarative deployment models by facilitating the comparison, selection, and migration of technologies. Wurster et al.'s approach is a meta-model of software deployment, but it is intended to help users select the best deployment technology for their scenario.

2.4.2 Specific Approaches

Melchor et al. [84] present a model-driven framework for defining data science pipelines independent of a particular execution platform and tools implemented with EMF. This framework separates the pipeline definition into two different modeling layers: a conceptual layer, where a data scientist specifies all the data and operations to be carried out by the pipeline, and an organizational layer, where a data engineer can specify the execution environment where the operations will be implemented. This approach allows the usage of different tools, improving replicability, the automation of process execution, improving reproducibility, and the definition of model verification rules, providing intentionality restrictions. This paper is relevant due to separating the conceptual and organizational layers and the automatic pipeline execution.

Sandobalin [110] develops a DSL, called ARGON, that can model cloud infrastructure; the model then allows the generation of scripts of different configuration management tools (CMTs) for CD through model-to-text (MTT) transformations. ARGON is implemented using EMF. Sandobalin's infrastructure meta-model abstracts the capabilities of cloud computing. Sandobalin's work is relevant as it uses MDE to reengineer CD scripts, similar to this work's goal. The works differ in the subject of the meta-model being used and the kind of text that will be parsed to the meta-model, as this one will use scripts from various CI/CD technologies.

Van den Heuvel et al. [58] introduce ChainOps, a model-driven DevOps approach for the blockchain. They focus on modeling smart contracts, computations in the form of executable code that promise to simplify trade ecosystems where parties may remain anonymous. They claim their model addresses concerns with trustworthiness, enabling non-technical end-users, and blockchain environment lock-in. They can apply MTT transformations to generate code for

various blockchain technologies. ChainOps is based on the AstraKode Blockchain Modeler, a software-as-a-service (SaaS) modeling platform for blockchain technologies. Their work is relevant to this one due to similar concerns regarding explainability, enabling non-technical users, and technology lock-in in CI/CD, and due to the MTT transformations.

Casale et al. [22] propose using MDE to support QA in data-intensive software systems. Their tool, DICE QA, would be capable of modeling big data applications. DICE QA covers simulation, verification, and architectural optimization. DICE could also generate code and performance, reliability, and safety models. Despite the different modeling subjects, Casale et al.'s work is relevant due to MTT transformations.

Kirchhof et al. [72] present MontiThings, a modeling infrastructure for systematic engineering of IoT applications. MontiThings is an extension of MontiArc, an architecture description language, and it features:

- integrated modeling languages for architectures, deployment, and error-handling of IoT applications that lift the level of abstraction in IoT system engineering and ensure separation of concerns
- a model-driven toolchain for the automated synthesis of executable IoT containers
- automated deployment planning, featuring deployment suggestions, for the generated containers
- support to suggest changes to deployment goals based on deployment planning results

Song et al. [116] create a model-based tool, GeneSIS, to generate deployment plans for IoT devices without human interaction. The tool takes a PSM as an input and transforms it into a PIM. Using the PIM and constraint solving, the tool then assigns deployment plans to devices in the PIM. Lastly, the PIM is transformed back into the PSMs necessary for device deployment. Song et al.'s work is relevant as it involves the entire reengineering process.

Hugues et al. [62] propose TwinOps, a process that unifies MBE, digital twins, and DevOps in one workflow that can be used to improve the engineering of CPSs. DevOps practices are combined with model-based code generation to facilitate deployments. TwinOps is based on a combination of SysML, a GPML for systems engineering [121], Architecture Analysis and Design Language (AADL), an architecture-centric DSML [6], Modelica, an object-oriented language to model CPSs [90], and Functional Mock-up Interface (FMI), standard that defines a container and an interface to exchange dynamic simulation models [44]. Hugues et al. bring MBE to DevOps by automatically generating code for monitoring, simulation, and testing based on their models. This differs from this work as there is no attempt to model CI/CD pipelines.

Colantoni et al. [30] present a work in progress for modeling CD pipelines based on JavaScript Object Notation (JSON). This work is separate from their aforementioned DevOpsML [29].

They use a previously developed approach, JSONSchemaDSL, to semi-automatically generate JSON-based DSLs specified through a JSONSchema. JSONSchemaDSL generates an EMF metamodel, an Xtext grammar, and a Sirius graphical representation. With this, Colantoni et al. allow

blended modeling, "the activity of interacting seamlessly with a single model through multiple notations, allowing a certain degree of temporary inconsistencies" [27], of JSON-based DSLs.

This approach is applied to Keptn, a CD technology, through its JSON-based DSL, Shipyard, leading to the modeling of CD pipelines.

Colantoni et al.'s approach is more limited than the one used in this paper as their meta-model will only describe one CI/CD technology.

Meyers et al. [88] present an MDE framework that supports continuous testing and fast development iterations in SCSs. Their framework is based on two DSLs implemented in Xtext. The first is a formal modeling language, the Functional Safety Formal Model (FuSaFoMo), that allows engineers to build a formal architectural description of a system similar to AADL and SysMl. The second is a contract-based requirements language that can link to FuSaFoMo and specify system behavior. Signal temporal logic can be generated from these contracts to verify system behavior. While Meyers et al.'s work does not attempt to model CI/CD pipelines, it is related as it uses MDE and MTT transformations to automate the generation of test cases from requirements.

Rivera et al. [106] propose Urano, a tool for automating the deployment process that uses UML to specify software architecture and the deployment process. From graphical UML specifications, Urano generates textual specifications in Amelia, a DSL specifically conceived for specifying and executing deployment workflows for distributed software systems. The Amelia specification can then be compiled into a Java-based application that automatically performs deployment operations [68]. Urano uses EMF. Rivera et al.'s work is related as they use MDE to automate the generation of deployment software from a model.

Ribeiro et al. [105] present a model-based solution to deploy software in the cloud automatically. Their solution uses UML models.

After all required deployment information has been input into the model, the solution generates an automatic deployment in four steps: it starts by interpreting a specific model, with information related to the cloud provider as well as the repository, then it interprets a general model, that has elements that specify VMs, OS, services, etc., then it creates the software stack, and lastly, it automatically generates code. The deployment code is written in Ruby and for Chef, an infrastructure deployment platform [23].

Ribeiro et al.'s work is relevant as it models the deployment of software in the cloud, a part of CI/CD pipelines.

Bucchiarone et al. [20] exploit multi-level modeling, meta-modeling with an arbitrary number of meta-levels [77], for designing and deploying gameful systems. Their approach is implemented in JetBrains MPS, a text-based meta-modeling framework [91]. While large parts of their work are unrelated, as it deals mainly with modeling gamification, their approach permits deployment from the model, so it is related.

Artač et al. [7] propose DICER, a framework to support the model-driven continuous design and deployment of data-intensive applications. As part of the DICE EU H2020 project [35], DICER's architecture comprises two main components: the Modeling Environment and the Deployment Service.

In the Modeling Environment, the user specifies the elements in the MODAClouds4DICE meta-model, an extension of MODACloudML. This model is transformed into a TOSCA-compliant model using ATL. The TOSCA model is transformed into a deployable TOSCA YAML blueprint using an Xtext MTT transformation.

The Deployment Service receives the TOSCA YAML blueprint and handles it using Chef. The Deployment Service is meant to be integrated with other CI/CD technologies.

Artač et al.'s work is related as their model can represent continuous software deployment steps.

Ketfi and Belkhatir [70] propose DYVA, a unified framework for dynamic deployment and reconfiguration of component-based software systems. The framework is based on their hierarchical meta-model, a PIM, that provides an abstract view of a component model. Then, this meta-model can be personalized into a specific component model. Changes in this personalized model, a PSM, trigger the deployment or the reconfiguration process. Although this framework was conceived using MDA principles, there is no evidence of the use of OMG modeling standards, which compromises interoperability [106]. Ketfi and Belkhatir's work is related as it aids CD in component-based software systems.

Silva et al. [32] propose OpenTOSCA for IoT, a TOSCA-based system to deploy IoT applications fully automatically. This approach requires low-level technical details to function. Silva et al.'s work is an approach to CD in IoT systems.

2.4.3 Other Studies

Bergerlin and Strandberg [13] report industrial requirements for bridging the gap between artificial intelligence (AI), MDE, and DevOps in CPSs. This paper is irrelevant to this work as it only approaches MDE and DevOps by how they relate to CPSs and not to each other.

Cobemale and Wimmer [31] introduce a vision for supporting model-based DevOps practices for CPSs. However, their focus is on improving the integration of DevOps practices in model-based engineering, not on creating a model-based approach to DevOps, so their work is irrelevant to this one.

Kirchhof et al. [71] compare two model-driven tools used in ML, MontiAnna and ML-Quadrat. They weigh the pros and cons of each tool when it comes to simplifying the ML process with MDE. They mention MLOps, CI/CD for ML-enabled projects [21], but only in passing.

Luz et al. [79] develop a model of DevOps adoption in organizations. This model represents the organizational entities related to DevOps adoption and only minimally describes DevOps activities to link them to these entities.

Chapter 3

Understanding the trends in CI/CD usage over time

When considering the number of CI/CD technologies available and the relevance of CI/CD in the contemporary SDLC, several questions arise about its usage over time: Do the most popular CI/CD technologies change? How often do the CI/CD technologies used by a project change? How often is there co-usage?

To answer these questions, we designed a study that involved mining public software repositories on GitHub using the GitHub API.

GitHub is the most widely used version control and software hosting service [137]. As of 2023, over 100 million developers used GitHub, and the platform hosted over 284 million public repositories [75]. GitHub also provides an API to collect information from its software repositories. With this API, it is possible to search repositories inside GitHub using parameters such as keywords in their name and README, size, number of stars, followers, and forks [111]. The insights collected using the API help understand various developer behaviors and have been used in several studies [56, 57, 53, 78, 21].

This mining, its results, and conclusions derived from them are also the basis for "Chronicles of CI/CD: A Deep Dive into its Usage Over Time" by Hugo Gião, André Flores, Jácome Cunha, and Rui Pereira [48].

This chapter centers on the study and its results.

Section 3.1 presents the methodology used for this work. This methodology includes how we found the CI technologies for this study, what information we have collected from the repositories and how it is related to our research questions, how we used the GitHub API to collect information from the repositories, and how we organized the data collected from the said repositories.

Section 3.2 showcases and analyzes the mining results.

Section 3.3 presents several works related to this one. Those works focus on mining information about several DevOps aspects in various software repositories.

Section 3.4 discusses the threats to the validity of our work.

Finally, section 3.5 presents the study's conclusions.

3.1 Methodology

This section details the process followed to collect the data for the analysis. Fig. 3.1 presents an overall view detailed in the following paragraphs.

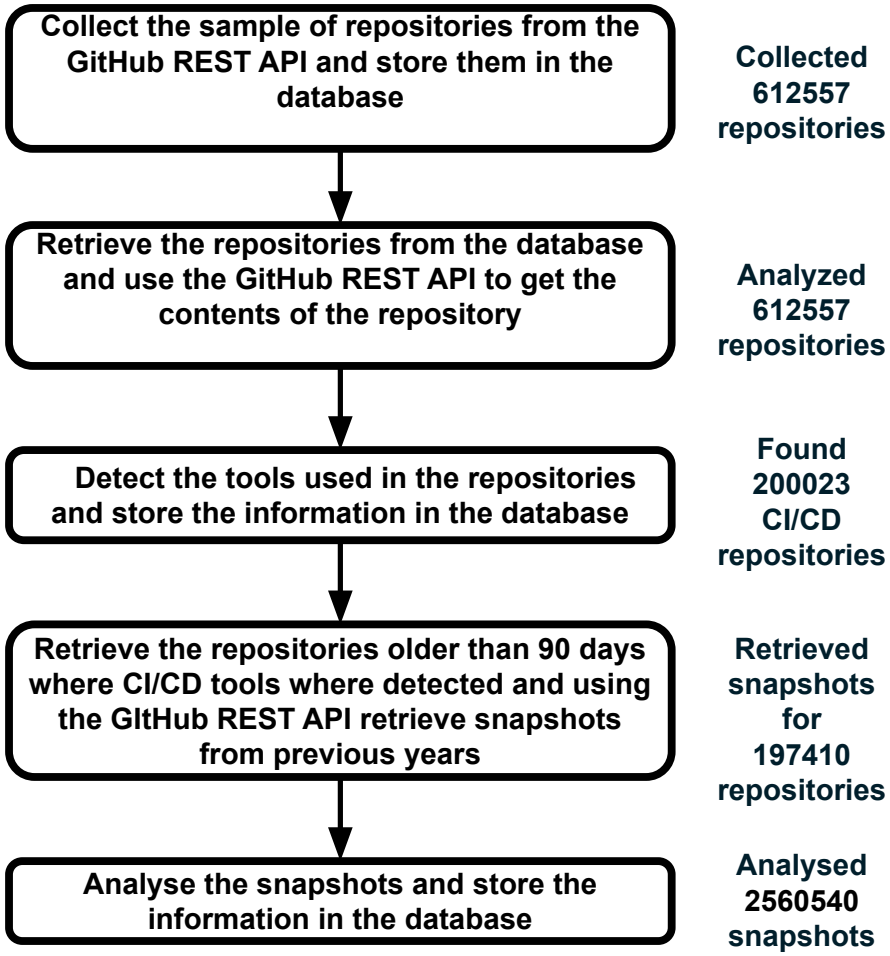


Figure 3.1: Data collection process.

3.1.1 Repository Sampling

The initial data collection phase involved assembling a representative sample of GitHub repositories reflective of real-world projects. To achieve this, school projects and smaller repositories were excluded, employing a methodology inspired by prior research projects. That is, the focus was on repositories with a certain level of popularity. Ultimately, only those with more than ten stars were collected, accounting for a mere 5% of all GitHub repositories.

The sampling process encompassed repositories from the beginning of 2012 to October 18, 2023. This timeframe was selected because the concept of DevOps gained prominence around

that period. This assertion is supported by the emergence of the first comprehensive survey on the state of DevOps by Puppet Labs in the same year [97].

Leveraging the GitHub REST API, a methodology was devised to identify the top one thousand repositories with the highest star count each week. GitHub was queried, retrieving one thousand repositories every week in the time range. From these, the ones with fewer than ten stars were excluded. This approach yielded a comprehensive dataset totaling 612,557 repositories, spanning the specified date range. The entire dataset of repositories is provided for reference [50]. The code used to create this dataset is also available [46].

3.1.2 CI/CD Technologies

Following the collection phase, the objective was to discern the technologies employed in each repository. The list of technologies to detect came from a dataset of 61 CI/CD technologies curated by the Cloud Native Computing Foundation [120], established by the Linux Foundation in 2015. Pertinent artifacts and patterns for each technology in the dataset were identified, enabling automatic recognition of repositories utilizing those technologies. This process involved analyzing file trees accessed through the GitHub REST API and scrutinizing file contents with extensions compatible with the respective technologies. Each technology was identified using one of two heuristics: *i)* some technologies use files with a particular extension; *ii)* for the others, specific types of files (e.g., YAML files) had to be inspected for content specific to the underlying technology. From this process, the initial 61 technologies went into different categories: *i)* 39 technologies that could be identified (section 3.1.2); *ii)* 10 which could not be identified since there was not a clear artifact to use (section 3.1.2); *iii)* 4 technologies that required the use of specific code to be identified as they are libraries embedded in the code (section 3.1.2); and *iv)* 1 deprecated technology with no current documentation that made it impossible to recognize (section 3.1.2).

3.1.3 The Repositories with CI/CD Technology

The analysis focused on the sample acquired earlier, specifically the contents of the latest commit to the main branch of each repository. To ensure reproducibility, the analysis result includes the SHA of each file tree (available in the database of the dataset). A total of 200,023 repositories were identified as utilizing one or more of the CI/CD technologies identified in the previous section. The comprehensive dataset containing the repositories and the corresponding technologies is also available for further examination [49]. The code used to create this dataset and the figures generated with the data are also available [46].

To examine the past state of CI/CD, snapshots of repositories were retrieved over time, and the same CI/CD technologies analysis was run on each snapshot. The comprehensive dataset containing the repositories and the corresponding snapshots and technologies is also made available for further examination [51]. The code used to create this dataset and the figures generated with the data are also available [47].

Agola	AppVeyor
ArgoCD	Bytebase
Cartographer	CircleCI
Cloud 66 Skycap	Cloudbees Codeship
Devtron	Flipt
GitLab	Google Cloud Build
Helmwave	Travis
Jenkins	JenkinsX
Keptn	Liquibase
Mergify	OctopusDeploy
OpenKruise	OpsMx
Ortelius	Screwdriver
Semaphore	TeamCity
werf	Woodpecker CI
GitHubActions	Codefresh
XL Deploy	Drone
Flagger	Harness.io
Flux	GoCD
Concourse	Kubernetes
AWS CodePipeline	

Table 3.1: CI/CD technologies that can be identified and analyzed.

Akuity	Bamboo
Buildkite	Bunnyshell
CAEPE	Keploy
Northflank	OpenGitOps
Ozone	Spacelift

Table 3.2: CI/CD technologies that cannot be identified due to the lack of clearly identifiable artifacts.

Brigade	k6
OpenFeature	Unleash

Table 3.3: Libraries introducing unnecessary complexity.

D2iQ Dispatch

Table 3.4: Deprecated technologies lacking documentation.

A test was run on a random sample of 85 repositories to choose the time interval for the analysis. Snapshots were retrieved at 90-day, 180-day, and 365-day intervals for each repository. Each snapshot was then searched for CI/CD technologies. The goal was to determine the number of changes in the CI/CD technology stack that would be lost by increasing the retrieval interval, a change being any difference in the CI/CD stack compared to the previous snapshot. For the sample, there were 270 stack changes at a 90-day sampling interval, 217 changes at a 180-day

sampling interval, and 179 changes at a 365-day sampling interval. From a 90-day to a 180-day rate, there was a 19.6% decrease in the detected changes, and from a 180-day to a 365-day rate, there was a 17.5% decrease in detected changes. A lower sampling interval, or retrieving all commits for each repository, was not feasible due to GitHub API rate limits and the time for the study. Based on these results, the analysis used a 90-day sampling rate.

A subset of the previously selected repositories was analyzed. The inclusion criteria were repositories where CI/CD technologies were detected in the latest commit and created before July 16, 2023 (so they were at least 90 days old). This resulted in a subset containing 197504 repositories.

For each selected repository, the first commit was retrieved. A sequential iterative process was employed, where the latest commit (if one existed) was retrieved for each 90-day interval starting from either January 1, 2012, or the repository's first commit date, whichever was later. This process continued until the date of the repository's last update at the time of retrieval. All commits were retrieved from the default branch of the repository. The git trees of the snapshot commits were then examined for CI/CD technology presence using the previously described methodology.

After all repositories were processed, the retrieved data was cleaned. Any snapshots from before January 1, 2012, were discarded, and the last snapshot of each repository was set to the one used in the previous analysis. For some snapshots, the GitHub API could not return a git tree. Another attempt was made to process these snapshots to eliminate any momentary API malfunction. The original detection method for GHA could lead to false detection when other technologies were present in the snapshot, so all snapshots that had more than one technology detected and GHA present were reprocessed. Lastly, each snapshot's date and detected technologies were checked against the detected technologies' launch dates, and snapshots where a technology was detected before it was launched were removed. If a repository was left without snapshots at the end of these cleaning steps, it was discarded.

From an initial 197,504 repositories selected for temporal analysis, the CI/CD technology use history of 197,410 was retrieved. For the 94 repositories whose CI/CD technology history could not be retrieved, the reasons are as follows: the 19 repositories could not be processed in the initial snapshot retrieval because they had either been deleted or gone private; in the data cleaning step, another 39 repositories could not be processed because they had either been deleted or gone private, and 36 were discarded because they had no snapshots at the end of the cleaning steps.

3.2 Results

Starting with analyzing the usage of CI/CD from 2012 until 2023. Fig. 3.2, shows the number of repositories created each year where, in their last commit, there is the presence of CI/CD technologies. As can be seen, even in repositories from 2012, almost one quarter included at least one technology; the percentage then increased until 2015 to more than 37% and has been in a gentle descending/stable curve since then. The drop in the last year is justified because many projects start without CI/CD, and only later is it introduced.

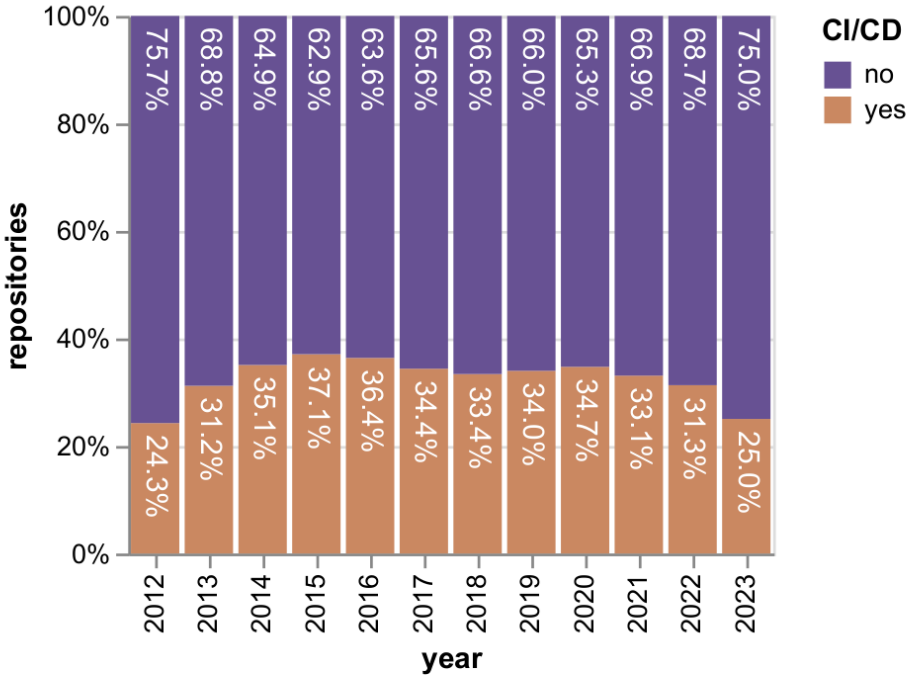


Figure 3.2: The usage of CI/CD per year.

As seen in fig. 3.3, CI/CD technologies are being integrated into the development workflow sooner as time goes on. CI/CD's increasing popularity is due to older, possibly more complex, projects' adoption and new, perhaps simpler, projects that see value in continuous practices. The sharp decreases in 2022 and 2023 come from all analyzed repositories having at least one CI/CD technology in 2023.

3.2.1 Usage of different technologies

A repository may use more than one technology at a time, as seen in fig. 3.4. The technologies used in a repository in a given year are the union of the technologies of the snapshots retrieved in that year. In cases where a repository does not have any snapshots in a given year but has activity following that year, the technologies used for that year are the ones in the most recent snapshot up to that point in time, i.e., if a repository has snapshots in 2016 and 2018, but not in 2017, its used technologies in 2017 are the ones from the last snapshot from 2016. If a repository has no activity following a given year, its technology usage stops being considered. This methodology applies to all charts showing technology usage over time.

Fig. 3.5 shows two significant trends in CI/CD, Travis, and GHA. Travis usage steadily increased from 2012 until it peaked in 2019 with 73284 repositories (36.6%). Since 2019, Travis's usage has been declining. This coincides with the rapid adoption of GHA; from 2019 to 2020, there was a 502.8% growth in the number of repositories using GHA, and from 2020 to 2021,

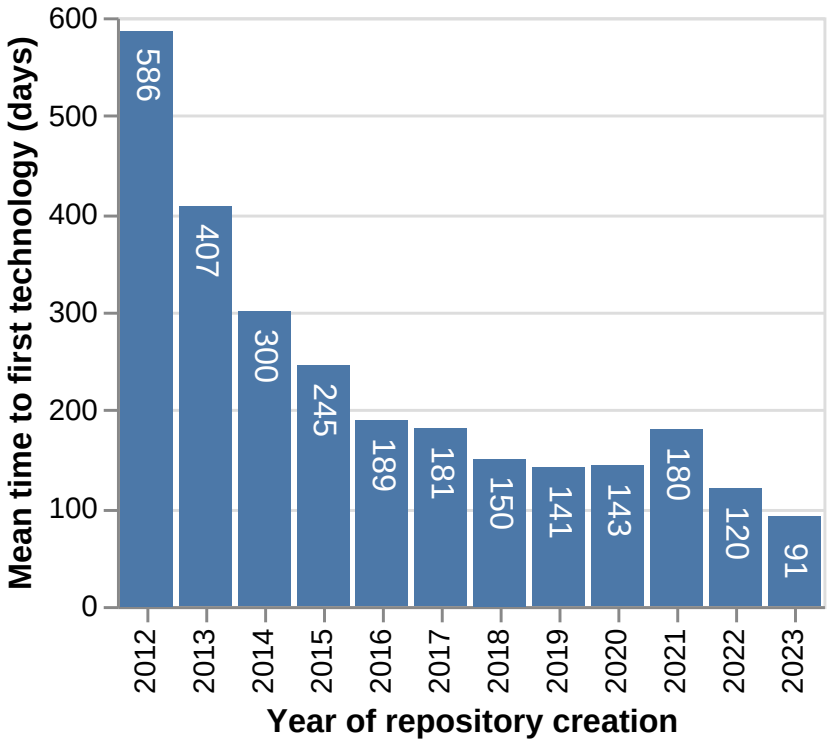


Figure 3.3: Mean time in days to first CI/CD technology detection by repository creation year.

there was an 86.6% growth. Of the 36587 repositories that used Travis in 2019 and were still active in 2023, 59.7% were using GHA and not Travis in 2023, 21.1% used Travis and not GHA, and 16.9% used both. Of the 87582 repositories using GHA in 2023, 45.4% had no snapshots from before 2020. The exodus from Travis and the influx from newer repositories have been the main drivers for GHA's growth.

From this data, there is an observable shift in the CI/CD technologies used over the years.

Fig. 3.6 shows the top 10 CI/CD stack transitions from repositories that solely used Travis in 2019. While many stopped being active, 22.9% moved from Travis to GHA. If we consider only the ones active, this 53.7% of all active Travis projects moved to GHA while.

3.2.2 How often do projects change CI/CD technologies?

As fig. 3.7a shows, the percentage of snapshots with CI/CD changes compared to the previous snapshot grows steadily from 2013 (2.3%) to 2019 (6.9%) and peaks in 2020 (12.2%) and 2021 (12.6%), coinciding with GHA's explosive growth phase. Since 2021, this number has remained stable at almost 8%.

This is a very significant result since it shows that every year, between 2.3% (in 2013) and 12.6% (in 2021) of all snapshots include some kind of change in the CI/CD technologies used. This represents a very significant amount of technology shift, with all the known issues with that. Moreover, it is essential to notice that this is constant over time, and there is no reason to think this

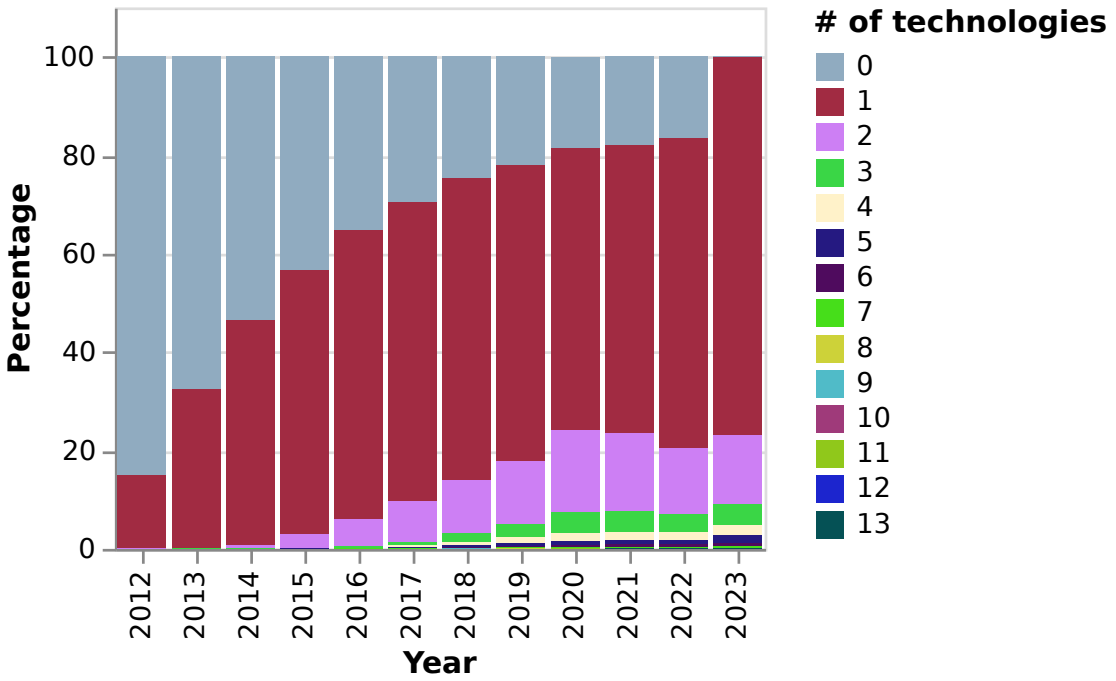


Figure 3.4: Percentage of repositories using at most a given number of technologies in a given year by year.

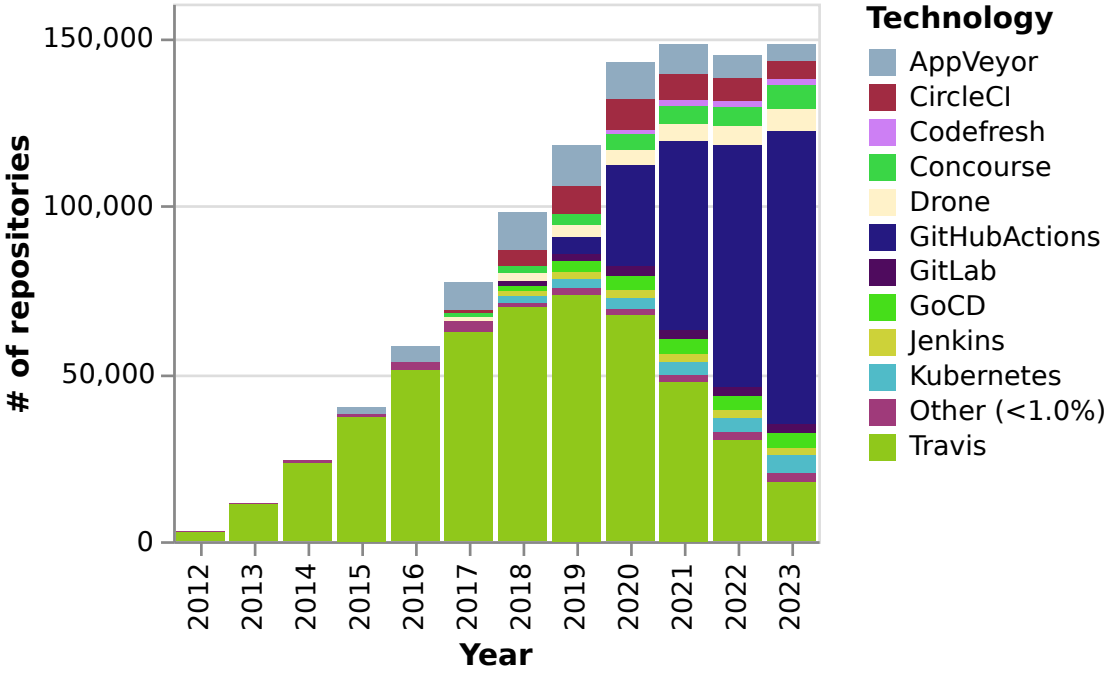


Figure 3.5: Number of repositories where each technology was detected by year.

may change in the near future. This means the research community can significantly contribute to aiding all these teams when they migrate and evolve their systems.

If we limit this analysis to a subset of active repositories from 2012 to 2023, that is, repositories

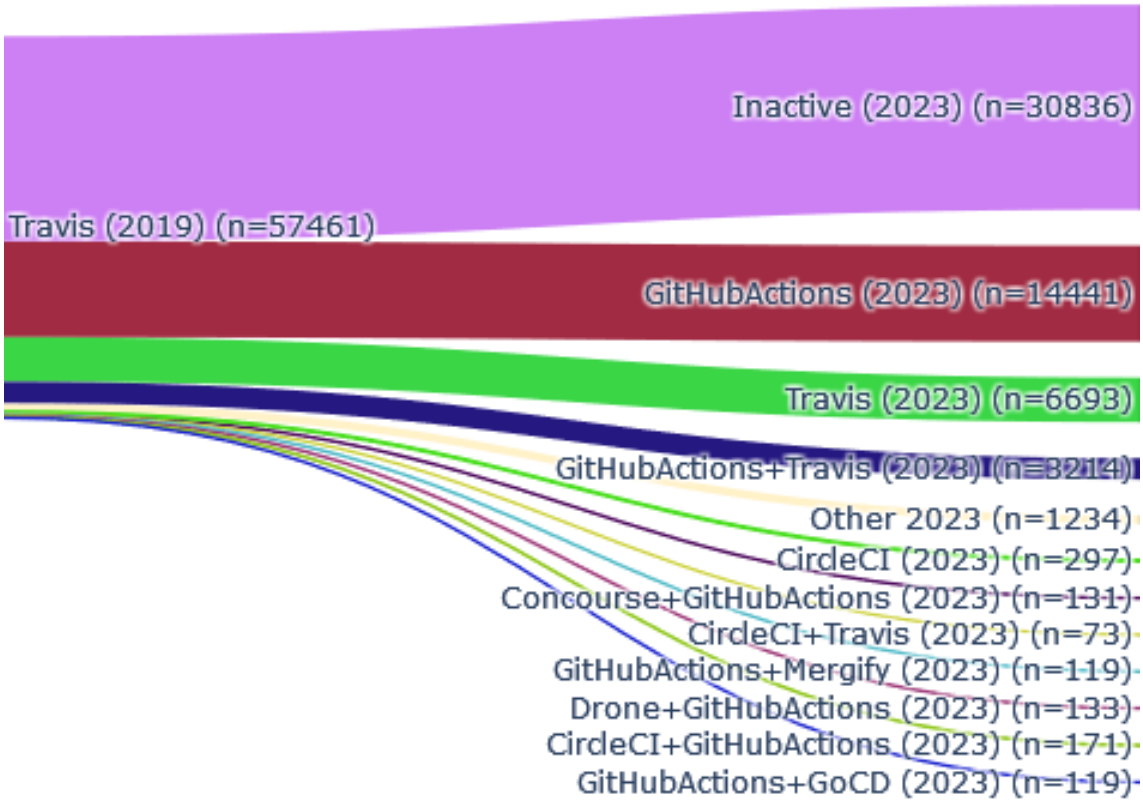


Figure 3.6: CI/CD technology stack transitions from repositories solely using Travis CI in 2019.

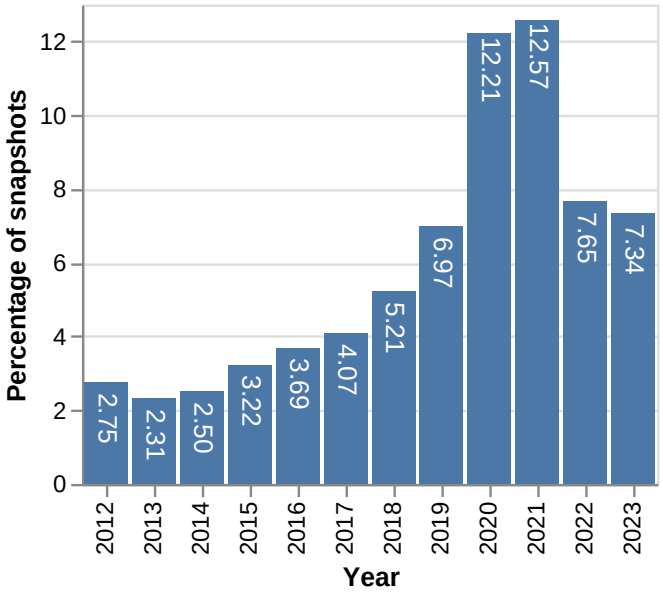
for which we would retrieve CI/CD technology information for every year from 2021 to 2023 following the previously described methodology, we find interesting results. As fig. 3.7b shows, similar trends can be observed but with higher change percentages over time. For this analysis, all snapshots for each repository before the first where we detected CI/CD technologies were discarded. This was done to eliminate a project's first choice of technologies from the analysis.

The percentage of CI/CD technology stack changes is significant and is consistently higher in long-running projects, meaning projects continuously look for technologies that better fit their workflow. Although one could expect some stability over the years for mature projects, this is not the case, as seen in the data.

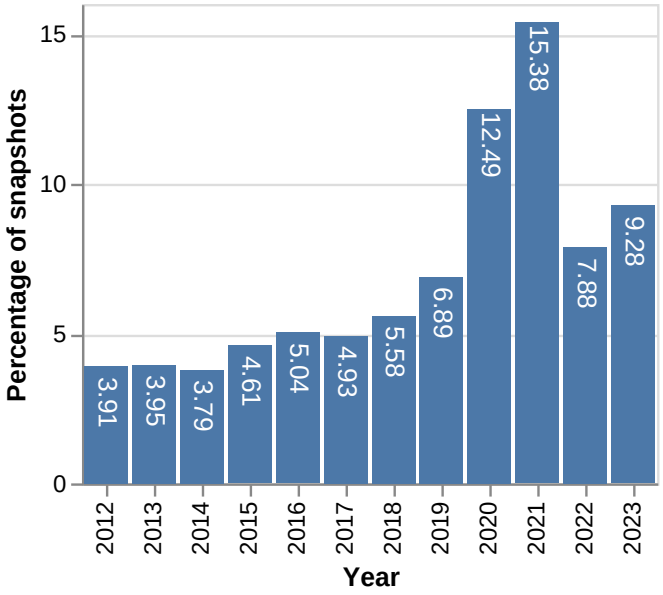
3.3 Related Work

In this section, we introduce our related work. This paper's related work focuses on several articles that mine software repositories to understand and improve aspects of software development. Similarly to these works, we mine repositories related to the CI/CD.

Xu et al. [141] introduce the idea of mining container image repositories for configuration and other deployment information of software systems. The authors also showcase the opportunities based on concrete software engineering tasks that can benefit from mining image repositories. They also summarize the challenges of analyzing image repositories and the approaches to address



(a) Percentage of snapshots with changes in the CI/CD technology stack from the previous snapshot by year (all repositories).



(b) Percentage of snapshots with changes in the CI/CD technology stack from the previous snapshot by year for the set of repositories active from 2012 to 2023 (n=8296).

Figure 3.7: Percentage of snapshots with changes in the CI/CD technology stack from the previous snapshot by year, considering all repositories (fig. 3.7a) and the set of repositories active from 2012 to 2023 (fig. 3.7b).

them. These authors focus their work on technologies for deployment, while with this work, we give a broader overview of the usage of CI/CD technologies.

Wu et al. [138] present a preliminary study on 857,086 Docker builds from 3,828 open-source projects hosted on GitHub. Using the Docker build data, the authors measure the frequency of broken builds and report their fix times. They also explore the evolution of Docker build failures across time. This work is focused on a particular technology, while ours covers many.

Zahedi et al. [142] present an empirical study exploring continuous software engineering from the practitioners' perspective by mining discussions from Q&A websites. The authors analyzed 12,989 questions and answers posted on Stack Overflow. The authors then used topic modeling to derive the dominant topics in this domain. They then identify and discuss critical challenges. Although the studied topic is related to CI/CD, we analyzed concrete software projects.

Mazrae et al. [108] present a qualitative study of CI/CD technologies usage, co-usage, and migration based on in-depth interviews. They identify reasons for using specific technologies, reasons for co-using CI/CD technologies in the same project, and migrations executed by the interviewees. Their study reveals a clear trend in migration from Travis to GHA. We have used a very different data source, but some of our study's conclusions align with Mazrae et al.

Goldazeh et al. [53] conduct a qualitative analysis of the usage of seven popular CI technologies in the GitHub repositories of 91m810 active npm packages having used at least one CI service over a period of nine years. Their findings include the fall of Travis, the rapid rise of GHA, and the co-usage of multiple CI technologies. These results align with ours, but we have investigated many more projects and CI/CD technologies.

Liu et al. [78] mine 84,475 open-source Android applications from GitHub, Bitbucket, and GitLab to search for CI/CD adoption. They find only around 10% applications leverage CI/CD technologies, a small number of applications (291) adopt multiple CI/CD technologies, nearly half of the applications that adopt CI/CD technologies do not really use them, and CI/CD technologies are useful to improve project popularity. Their approach is similar to ours. However, our analysis is done with a more significant sample of 612,557 repositories, and we don't limit ourselves to Android applications.

Calefato et al. [21] study MLOps (DevOps but focused on machine learning projects) practices in GitHub repositories, focusing on GHA and CML. Their preliminary results suggest that the adoption of MLOps workflows is somewhat limited. On the other hand, we have found that many projects rely on CI/CD pipelines.

Kumar et al. [73] assess the maturity of DevOps practices in the software industry. To this end, they analyze the HELENA2 dataset (an international survey aiming to collect data regarding the common use of software and systems in practice). They rank organizations by DevOps maturity. The authors conducted a user survey while we analyzed software repositories.

Hilton et al. [59] studied CI by mining 34,544 OSS projects on GitHub and surveying 442 developers. They found many OSS teams that don't use CI. Of the ones that use CI, 90% used Travis. They find popular projects are more likely to use CI and that the median time for CI

adoption is one year. Hilton et al.'s study was published in 2016, so its results are not representative of the current state of CI/CD.

Beller et al. [12], through an analysis of GitHub, found that Travis had seen a sharp increase in usage up to 2017, being used by one-third of popular projects. Just as Hilton et al.'s, Beller et al.'s study is not representative of the current state of CI/CD.

Widder et al. [135] conducted a qualitative study of 7,276 GitHub projects that had migrated away from Travis. They found that a project's dominant language is an important predictor for Travis abandonment. They also found more complex projects were less likely to migrate from Travis. In a follow-up [134], they investigate the pain points of Travis.

Decan et al. [33] study GHA use in nearly 70,000 GitHub repositories. They find 43.9% of repositories use GHA workflows. They also characterize these repositories according to which jobs, steps, and reusable Actions are used and how.

Vasilescu et al. [126] studied 1,884 GitHub projects in 2014. They found Travis usage in 918 repositories (48.7%).

Lamba et al. [76] study the spread of CI/CD in Node Package Manager (NPM) package repositories. Their analysis is done through repository badges, a recent innovation on code hosting platforms. They search for 12 CI/CD technologies in 168,510 NPM package repositories hosted on GitHub. Their study focuses on how CI/CD technologies gain market share.

JetBrains [123] provides results from yearly surveys of developers about the developer ecosystem from 2017 to 2023. Their results show Jenkins as the most popular CI/CD technology until 2022 when GHA takes over. They also reveal the increasing relevance of CI/CD.

The Continuous Delivery Foundation, a project of the Linux Foundation, [119] provides a report on the state of CD. They find that 84% of developers participated in DevOps-related activities as of Q1 2023 and that the average number of DevOps-related technologies used by developers concurrently remained stable from 2019 to 2023 at 4.5.

Stack Overflow's annual developer surveys [117] show an increase in CI/CD and DevOps usage year-over-year.

3.4 Threats to validity

There are multiple threats to the validity of the study, which are addressed in the following paragraphs.

The study focuses on open-source software and, in particular, on projects hosted on GitHub. Thus, the sampling does not include other kinds of software (e.g., propriety). Conclusions cannot be generalized to these other kinds of software projects. Nevertheless, many companies also have their software on GitHub, and one may expect workers from these companies to use similar technologies in other projects. Moreover, others have reached similar conclusions by interviewing developers [53].

Since only GitHub was used, it cannot be said that these results apply to projects in other code repository services. However, there is no reason to consider projects hosted on GitHub to be significantly different from other projects in other repository services.

Still regarding the use of GitHub as the source of the software projects analyzed, there is the predominance of GHA. One of the main reasons for this may in fact be related to the use of GitHub as the source of projects. However, GitLab CI/CD was also detected, the CI/CD technology used by another repository service (GitLab).

The sample repositories were collected by getting the 1,000 results sent by the GitHub API, doing it every week in the time frame. This resulted in more than 600,000 repositories, from which more than 200,000 have CI/CD. Although more repositories could have been collected, this would increase the time to retrieve them in a way that would make the work unfeasible. Moreover, the query did not impose any restriction on the results, except for the 10 stars used to have some kind of "quality" metric for the projects. Thus, the repositories retrieved should not be biased in any other way.

Only technologies that could be identified through files in the repository were considered. Indeed, from the 61 technologies identified by the Cloud Native Computing Foundation, 14 technologies could not be identified (plus 1 deprecated). Nevertheless, 64% of all technologies could be identified.

Some technologies are detected through file contents and there is no guarantee a random file would not have a certain string inside that matches. However, the defined content would only make sense in the technology context, this probably did not happen. In any case, if it happened, it was for a very small number of files that should not change the overall conclusions of the work.

There is an assumption that the presence of CI/CD artifacts (e.g., configuration files) means the underlying project is using such a technology. However, this may not be the case as some artifacts may be left forgotten from older usages.

3.5 Conclusions

From the more than 600,000 repositories collected, almost one-third, 32.7%, include CI/CD technologies. This number has been relatively stable over the years. We can also see one of the current technologies dominates the market, with 57.8% of the projects using GHA. This number may be biased as we used GitHub as our source of projects, but, if this is the case, it means the source code manager has a significant impact on the choice of CI/CD technologies. Furthermore, we have shown the time between the creation of the repository and the adoption of CI/CD has been decreasing substantially.

Regarding the usage over the years, we discovered developers tend to change quite often CI/CD technologies. Indeed, there has been a massive change from Travis to GHA, but changes in technology are quite common. Indeed, in the last two years, more than 7% of all snapshots include some kind of change in the CI/CD technological stack. If we consider long-running projects

(projects running from 2012 up until 2023), this number is even greater. CI/CD's steep learning curve means developers need support to be successful in these endeavors, which currently is mostly non-existent.

Many projects use more than one technology for CI/CD. In the last 4 years, each year, more than 23,000 projects include more than one technology, which accounts for about 20% of all projects. From this, we can see one technology is often not sufficient to cope with all the requirements of some projects. Taking into account Mazrae et al.'s reporting that one of the main reasons for migrating CI/CD technologies is to diminish co-usage [108], this could also represent a large number of projects looking to migrate.

In future work, we plan to extend this work to other repository services such as GitLab, or Bitbucket. Another interesting research direction is to replicate this work in an industrial setting.

For this work, the information retrieved from this study gives insight into what the most popular CI/CD technologies and migrations are. This will mean the CI/CD pipeline reengineering software that will be developed will support relevant real-world cases.

Chapter 4

Solution Proposal

4.1 Automatic Migration Tool

The best way to support developers in migrating CI/CD would be to automate translating their existing pipeline into the new syntax as much as possible. This would speed up the migration, helping keep up productivity and reducing lock-in to any technology.

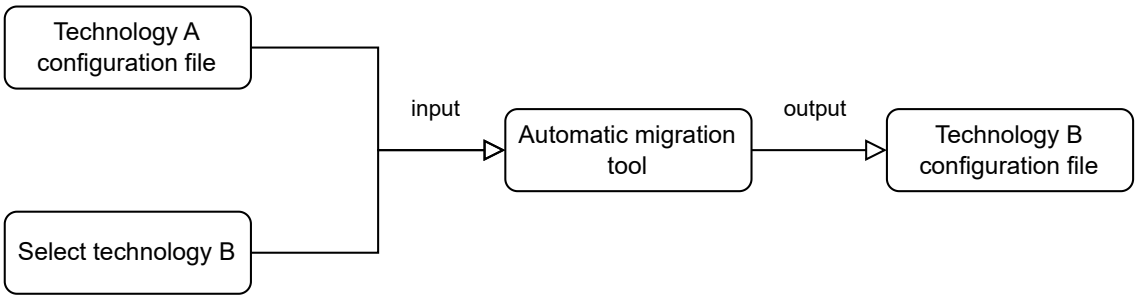


Figure 4.1: Automatic CI/CD migration tool.

This work proposes an automatic CI/CD pipeline migration tool that would function as described in fig. 4.1. A user would input a CI/CD pipeline written in the DSL of a given technology A and choose the technology they want to migrate that pipeline into (technology B). The program would output a pipeline that is in the correct syntax for technology B's DSL and that is semantically equivalent to the original pipeline that was provided in technology A.

To this end, this work intends to leverage model-driven engineering by creating a meta-model that defines a modeling language for CI/CD pipelines in a technology-independent manner. This technology-independent meta-model would serve as the basis for a complete reengineering process in automatic CI/CD migration. Such a meta-model would still need to allow modeling pipelines with a high level of detail to make translation between technologies possible.

The tool's logic will be implemented using model transformations following the method base for migration methods specified by Grieger et al. [54]. TTM transformations will generate a technology-specific model (TSM) of an input pipeline configuration file. That TSM will be transformed into a technology-independent model (TIM) that conforms to the technology-independent

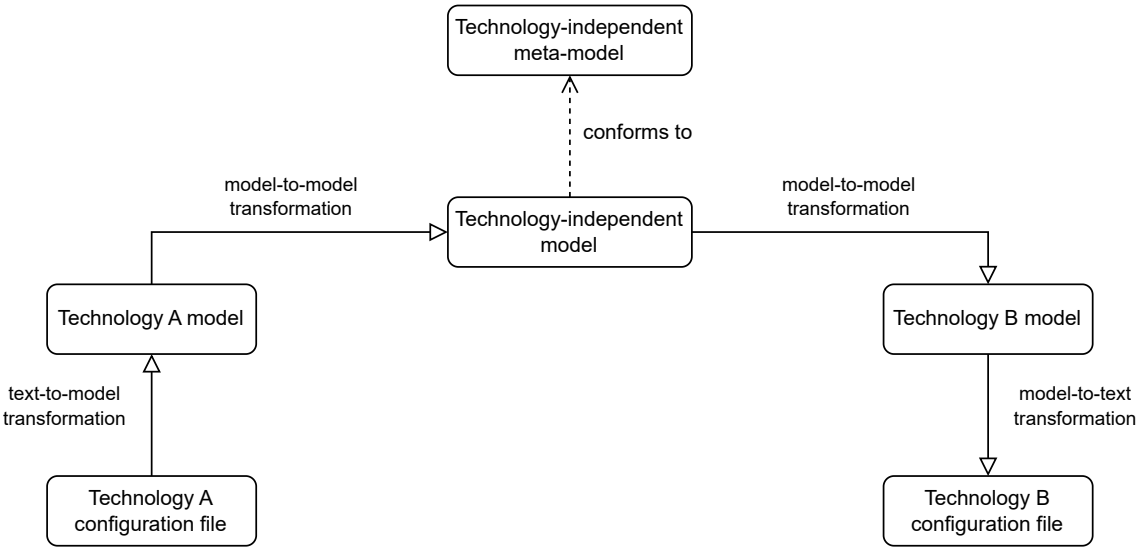


Figure 4.2: CI/CD pipeline reengineering process.

meta-model through MTM transformations. From the TIM, MTM transformations will generate a TSM for a different pipeline technology. The translated configuration file will be generated from the new TSM through MTT transformations. Fig. 4.2 shows this reengineering process as a horseshoe model [69].

4.2 Work Plan

This section details the work plan to implement the reengineering process described in fig. 4.2.

Tab. 4.1 is a chronogram of the work already undertaken towards the goal during the first semester. Efforts centered around better defining the problem at hand, understanding trends in CI/CD usage, and learning about MDE and how to use it to create a solution.

During the second semester, we will center the work on implementing and evaluating the proposed solution.

The first step will be working on the technology-independent meta-model for CI/CD pipelines. This work will be based on a pre-existing meta-model. However, this meta-model does not have the required amount of detail. Translation between various CI/CD technologies will require detailed knowledge of their functionalities and how to map them to a TIM. Because of this, the pre-existing meta-model must be refined before it can serve as the basis for the reengineering process. The EMF will be used to handle modeling work, including the meta-model definition, due to its extensive modeling capabilities and integration with other modeling tools.

Afterward, the focus will be on creating TSMs of CI/CD technologies through text-to-model transformations. This will be done by leveraging Xtext, which, from a grammar, can generate a text parser that transforms text to an EMF model. The goal will be to create grammars for the DSLs of CI/CD technologies, therefore handling text-to-model transformations automatically.

The TSMs generated must then be transformed into TIMs that conform to the technology-independent meta-model. This will be done through model-to-model transformations.

At this point in the development, there will be an evaluation phase. The goal will be to ascertain whether the TIMs that are being generated are semantically equivalent to the CI/CD configuration files they were generated from.

After ensuring the TIMs are being generated correctly, work will move on to the forward engineering part of the reengineering process. This will require handling transformations from TIMs back into TSMs through model-to-model transformations. The TSMs will conform to the meta-models inferred from the grammars previously specified in Xtext.

The final step of the reengineering process to implement will be generating pipeline configuration files from the TSMs. This will be done using Xtext's code generation capabilities.

Lastly, there will be a validation phase to check if the code generated from the models is syntactically correct and to determine the conversion rate between technologies.

Concurrently with all the aforementioned development phases, there will also be work on thesis writing throughout the semester.

Tab. 4.2 lays out the timeline of this work plan.

Tasks	September			October					November				December				January				
	W2	W3	W4	W1	W2	W3	W4	W5	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W5
Problem Specification																					
Mining Software Repositories																					
Data Visualization																					
Paper Writing																					
Report Writing																					

Table 4.1: Work Plan Gantt Chart - First Semester.

Tasks	February				March				April				May					June			
	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W5	W1	W2	W3	W4
Meta-model refinement																					
Technology-specific models																					
Transformations to TIMs																					
Evaluation																					
Transformations to TSMs																					
Code generation																					
Validation																					
Thesis Writing																					

Table 4.2: Work Plan Gantt Chart - Second Semester.

Chapter 5

Conclusions

There is a synergy between CI/CD and MDE. Both aim to shorten the development life-cycle, deal with complexity, and improve the software development process and product quality [29].

As section 2.1 details, CI/CD has never been more relevant to the SDLC, and this shows no signs of decreasing. However, there is a plethora of CI/CD technologies. Each of these technologies has pros and cons relative to the project they're applied to. Over time, evolution in both the software project and the CI/CD technologies it uses can make CI/CD technology migration the right path forward.

CI/CD technologies have a high learning curve [108]. This can make migrations hard to execute [108] and lengthy [64]. Despite this, as seen in section 2.2, CI/CD technology platforms don't provide much support for software developers migrating technologies.

MDE's principle of abstracting up can be a boon to developers migrating CI/CD. As section 2.3 details, it is possible to define a meta-model to represent CI/CD pipelines in a platform-independent manner. Through TTM, MTM, and MTT transformations, this meta-model could serve as the basis for an automated reengineering of CI/CD pipelines. This would greatly diminish concerns about the learning curve of specific CI/CD technologies, allowing developers to focus on the pros and cons of a technology for their project, thus avoiding lock-in and making the project more agile.

As seen in section 2.4, many attempts have been made at bringing MDE to CI/CD. The attempts vary in their approaches, from just using models to help in DevOps process documentation to applying reengineering to simplify software deployment. These works give confidence in the feasibility of this project and its novelty, as none attempt to model CI/CD pipelines in the aforementioned manner.

References

[1] *About the MOF Query/View/Transformation Specification Version 1.3*. URL: https://www.omg.org/spec/QVT/1.3/About-QVT (visited on 12/27/2023).

[2] *Acceleo | Home*. URL: https://eclipse.dev/acceleo/ (visited on 12/27/2023).

[3] Roberto Acerbis et al. "Developing eBusiness Solutions with a Model Driven Approach: The Case of Acer EMEA". In: *Web Engineering*. Ed. by Luciano Baresi, Piero Fraternali, and Geert-Jan Houben. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 539–544. ISBN: 978-3-540-73597-7.

[4] Adetokunbo A A Adenowo and Basirat A Adenowo. "Software Engineering Methodologies: A Review of the Waterfall Model and Object-Oriented Approach". In: *International Journal of Scientific & Engineering Research* 4.7 (2013), pp. 427–434.

[5] S.A.I.B.S. Arachchi and Indika Perera. "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management". In: *2018 Moratuwa Engineering Research Conference (MERCon)*. 2018 Moratuwa Engineering Research Conference (MERCon). May 2018, pp. 156–161. DOI: 10.1109/MERCon.2018.8421965. URL: https://ieeexplore.ieee.org/abstract/document/8421965?casa_token=vM2GUu7Bq7cAAAAA:wZUfCkE4AhbTQo8WZkG7ECIfw0KAsoPmDqj5x9DVaOYkfcc6oSSxOSFsSPUyzahjRvti4ELGc28 (visited on 11/29/2023).

[6] *Architecture Analysis and Design Language (AADL)*. URL: https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=191439,191439 (visited on 12/27/2023).

[7] Matej Artač et al. "Model-driven continuous deployment for quality DevOps". In: *Proceedings of the 2nd International Workshop on Quality-Aware DevOps*. ISSTA '16: International Symposium on Software Testing and Analysis. Saarbrücken Germany: ACM, July 21, 2016, pp. 40–41. ISBN: 978-1-4503-4411-1. DOI: 10.1145/2945408.2945417. URL: https://dl.acm.org/doi/10.1145/2945408.2945417 (visited on 01/04/2024).

[8] *Automating migration with GitHub Actions Importer*. GitHub Docs. URL: https://docs.github.com/en/actions/migrating-to-github-actions/automated-migrations/automating-migration-with-github-actions-importer (visited on 12/06/2023).

[9] Zia Babar, Alexei Lapouchnian, and Eric Yu. "Modeling DevOps Deployment Choices Using Process Architecture Design Dimensions". In: *The Practice of Enterprise Modeling*. Ed. by Jolita Ralyté, Sergio España, and Óscar Pastor. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2015, pp. 322–337. ISBN: 978-3-319-25897-3. DOI: 10.1007/978-3-319-25897-3_21.

[10] Kiyana Bahadori and Tullio Vardanega. "DevOps Meets Dynamic Orchestration". In: *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*. Ed. by Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 142–154. ISBN: 978-3-030-06019-0. DOI: 10.1007/978-3-030-06019-0_11.

[11] K. Beck. "Embracing change with extreme programming". In: *Computer* 32.10 (Oct. 1999), pp. 70–77. ISSN: 00189162. DOI: 10.1109/2.796139. URL: http://ieeexplore.ieee.org/document/796139/ (visited on 12/09/2023).

[12] Moritz Beller, Georgios Gousios, and Andy Zaidman. "Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub". In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). Buenos Aires, Argentina: IEEE, May 2017, pp. 356–367. ISBN: 978-1-5386-1544-7. DOI: 10.1109/MSR.2017.62. URL: http://ieeexplore.ieee.org/document/7962385/ (visited on 12/17/2023).

[13] Johan Bergelin and Per Erik Strandberg. "Industrial requirements for supporting AI-enhanced model-driven engineering". In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 2022, pp. 375–379.

[14] Francisco Javier Bermúdez Ruiz, Jesús García Molina, and Oscar Díaz García. "On the application of model-driven engineering in data reengineering". In: *Information Systems* 72 (Dec. 1, 2017), pp. 136–160. ISSN: 0306-4379. DOI: 10.1016/j.is.2017.10.004. URL: https://www.sciencedirect.com/science/article/pii/S0306437915300508 (visited on 12/26/2023).

[15] *bliki: UmlMode*. martinfowler.com. URL: https://martinfowler.com/bliki/UmlMode.html (visited on 12/19/2023).

[16] N. Bohr. "On the constitution of atoms and molecules". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 26.151 (July 1913), pp. 1–25. ISSN: 1941-5982, 1941-5990. DOI: 10.1080/14786441308634955. URL: https://www.tandfonline.com/doi/full/10.1080/14786441308634955 (visited on 12/19/2023).

[17] Richard C. Gronback Boldt Nick. *Graphical Modeling Framework | The Eclipse Foundation*. URL: https://eclipse.dev/modeling/gmp/ (visited on 12/27/2023).

[18] Francis Bordeleau et al. "Towards Modeling Framework for DevOps: Requirements Derived from Industry Use Case". In: *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*. Ed. by Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 139–151. ISBN: 978-3-030-39306-9. DOI: 10.1007/978-3-030-39306-9_10.

[19] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering. Cham: Springer International Publishing, 2017. ISBN: 978-3-031-01421-5. DOI: 10.1007/978-3-031-02549-5. URL: https://link.springer.com/10.1007/978-3-031-02549-5 (visited on 12/19/2023).

[20] Antonio Bucchiarone, Antonio Cicchetti, and Annapaola Marconi. "Exploiting Multi-level Modelling for Designing and Deploying Gameful Systems". In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS). Munich, Germany: IEEE, Sept. 2019, pp. 34–44. ISBN: 978-1-72812-536-7. DOI: 10.1109/MODELS.2019.00-17. URL: https://ieeexplore.ieee.org/document/8906924/ (visited on 01/04/2024).

[21] Fabio Calefato, Filippo Lanubile, and Luigi Quaranta. "A Preliminary Investigation of MLOps Practices in GitHub". In: *Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '22. Helsinki, Finland: Association for Computing Machinery, 2022, pp. 283–288. ISBN: 9781450394277. DOI: 10.1145/3544902.3546636. URL: https://doi.org/10.1145/3544902.3546636.

[22] Giuliano Casale et al. "DICE: Quality-Driven Development of Data-Intensive Cloud Applications". In: *2015 IEEE/ACM 7th International Workshop on Modeling in Software Engineering*. 2015 IEEE/ACM 7th International Workshop on Modeling in Software Engineering (MiSE). Florence: IEEE, May 2015, pp. 78–83. ISBN: 978-1-4673-7055-4. DOI: 10.1109/MiSE.2015.21. URL: https://ieeexplore.ieee.org/document/7167407/ (visited on 12/07/2023).

[23] *Chef Software DevOps Automation Solutions | Chef*. Chef Software. URL: https://www.chef.io/ (visited on 01/04/2024).

[24] Lianping Chen. "Continuous Delivery: Huge Benefits, but Challenges Too". In: *IEEE Software* 32.2 (Mar. 2015). Conference Name: IEEE Software, pp. 50–54. ISSN: 1937-4194. DOI: 10.1109/MS.2015.27. URL: https://ieeexplore.ieee.org/abstract/document/7006384?casa_token=zca1QQTWpkwAAAAA:3KlDQ3L7xFSm4tG1

NGoTPSWU5IKRCyOmVFYurgy0421HIz4u77Jm9q1Hcmlwy8ZMTI9tbKxtB-8 (visited on 11/29/2023).

[25] Wei Chen et al. "MORE: A Model-Driven Operation Service for Cloud-Based IT Systems". In: *2016 IEEE International Conference on Services Computing (SCC)*. 2016 IEEE International Conference on Services Computing (SCC). San Francisco, CA, USA: IEEE, June 2016, pp. 633–640. ISBN: 978-1-5090-2628-9. DOI: 10.1109/SCC.2016.88. URL: http://ieeexplore.ieee.org/document/7557508/ (visited on 01/04/2024).

[26] E.J. Chikofsky and J.H. Cross. "Reverse engineering and design recovery: a taxonomy". In: *IEEE Software* 7.1 (Jan. 1990). Conference Name: IEEE Software, pp. 13–17. ISSN: 1937-4194. DOI: 10.1109/52.43044. URL: https://ieeexplore.ieee.org/abstract/document/43044 (visited on 12/26/2023).

[27] Federico Ciccozzi et al. "Blended Modelling - What, Why and How". In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). Munich, Germany: IEEE, Sept. 2019, pp. 425–430. ISBN: 978-1-72815-125-0. DOI: 10.1109/MODELS-C.2019.00068. URL: https://ieeexplore.ieee.org/document/8904858/ (visited on 12/27/2023).

[28] *CircleCI Orbs*. CircleCI. URL: https://circleci.com/orbs/ (visited on 12/17/2023).

[29] Alessandro Colantoni, Luca Berardinelli, and Manuel Wimmer. "DevOpsML: towards modeling DevOps processes and platforms". In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems. Virtual Event Canada: ACM, Oct. 16, 2020, pp. 1–10. ISBN: 978-1-4503-8135-2. DOI: 10.1145/3417990.3420203. URL: https://dl.acm.org/doi/10.1145/3417990.3420203 (visited on 12/07/2023).

[30] Alessandro Colantoni et al. "Towards blended modeling and simulation of DevOps processes: the Keptn case study". In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '22: ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems. Montreal Quebec Canada: ACM, Oct. 23, 2022, pp. 784–792. ISBN: 978-1-4503-9467-3. DOI: 10.1145/3550356.3561597. URL: https://dl.acm.org/doi/10.1145/3550356.3561597 (visited on 12/07/2023).

[31] Benoit Combemale and Manuel Wimmer. "Towards a Model-Based DevOps for Cyber-Physical Systems". In: *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*. Ed. by Jean-Michel Bruel,

Manuel Mazzara, and Bertrand Meyer. Vol. 12055. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 84–94. ISBN: 978-3-030-39305-2. DOI: 10.1007/978-3-030-39306-9_6. URL: http://link.springer.com/10.1007/978-3-030-39306-9_6 (visited on 12/07/2023).

[32] Ana C. Franco Da Silva et al. "OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker". In: *Proceedings of the 6th International Conference on the Internet of Things*. IoT'16: The 6th International Conference on the Internet of Things. Stuttgart Germany: ACM, Nov. 7, 2016, pp. 181–182. ISBN: 978-1-4503-4814-0. DOI: 10.1145/2991561.2998464. URL: https://dl.acm.org/doi/10.1145/2991561.2998464 (visited on 01/04/2024).

[33] Alexandre Decan et al. "On the Use of GitHub Actions in Software Development Repositories". In: *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME). Limassol, Cyprus: IEEE, Oct. 2022, pp. 235–245. ISBN: 978-1-66547-956-1. DOI: 10.1109/ICSME55016.2022.00029. URL: https://ieeexplore.ieee.org/document/9978190/ (visited on 01/31/2024).

[34] Elisabetta Di Nitto et al., eds. *Model-Driven Development and Operation of Multi-Cloud Applications: The MODAClouds Approach*. SpringerBriefs in Applied Sciences and Technology. Cham: Springer International Publishing, 2017. ISBN: 978-3-319-46031-4. DOI: 10.1007/978-3-319-46031-4. URL: http://link.springer.com/10.1007/978-3-319-46031-4 (visited on 01/04/2024).

[35] *DICE Project · dice-project/DICE-Knowledge-Repository Wiki*. URL: https://github.com/dice-project/DICE-Knowledge-Repository/wiki/DICE-Project (visited on 01/04/2024).

[36] Paul M. Duvall, Steve Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Google-Books-ID: PV9qfEdv9L0C. Pearson Education, June 29, 2007. 313 pp. ISBN: 978-0-321-63014-8.

[37] *Ecore - Eclipsepedia*. URL: https://wiki.eclipse.org/Ecore (visited on 12/27/2023).

[38] Badr El Khalyly et al. "A new metamodel approach of CI/CD applied to Internet of Things Ecosystem". In: *2020 IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)*. 2020 IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS). Kenitra, Morocco: IEEE, Dec. 2, 2020, pp. 1–6. ISBN: 978-1-72816-921-7. DOI: 10.1109/ICECOCS50124.2020.9314485. URL: https://ieeexplore.ieee.org/document/9314485/ (visited on 12/07/2023).

[39] *EMFText*. EMFText. URL: http://devboost.github.io/EMFText/ (visited on 12/27/2023).

[40] Nicolas Ferry et al. “ENACT: Development, Operation, and Quality Assurance of Trustworthy Smart IoT Systems”. In: *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*. Ed. by Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 112–127. ISBN: 978-3-030-06019-0. DOI: 10.1007/978-3-030-06019-0_9.

[41] *Finding and customizing actions*. GitHub Docs. URL: https://docs.github.com/en/actions/learn-github-actions/finding-and-customizing-actions (visited on 12/17/2023).

[42] Franck Fleurey et al. “Model-Driven Engineering for Software Migration in a Large Industrial Context”. In: *Model Driven Engineering Languages and Systems*. Ed. by Gregor Engels et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 482–497. ISBN: 978-3-540-75209-7. DOI: 10.1007/978-3-540-75209-7_33.

[43] Martin Fowler. *Continuous Integration*. martinfowler.com. URL: https://martinfowler.com/articles/continuousIntegration.html (visited on 11/29/2023).

[44] *Functional Mock-up Interface*. URL: https://fmi-standard.org/ (visited on 12/27/2023).

[45] Hugo Gião, Jácome Cunha, and Rui Pereira. *Model-Driven Approaches for DevOps: A Systematic Literature Review*. Manuscript submitted for publication. 2023. URL: https://h4g0.github.io/Survey_MDE_DevOps.pdf.

[46] Hugo Gião et al. 2023. URL: https://anonymous.4open.science/r/DevOps-Repositories-122E.

[47] Hugo Gião et al. 2023. URL: https://anonymous.4open.science/r/github-devops-mining-C0D7.

[48] Hugo Gião et al. *Chronicles of CI/CD: A Deep Dive into its Usage Over Time*. Manuscript submitted for publication. 2023.

[49] Hugo Gião et al. *CI/CD repos with tools*. Nov. 2023. DOI: 10.6084/m9.figshare.24578740.v2. URL: https://figshare.com/articles/dataset/Untitled_Item/24578740.

[50] Hugo Gião et al. *CI/CD repositories from GitHub*. Nov. 2023. DOI: 10.6084/m9.figshare.24578746.v1. URL: https://figshare.com/articles/dataset/CI_CD_repositories_from_GitHub/24578746.

[51] Hugo Gião et al. *CI/CD repositories with tool history*. Nov. 2023. DOI: 10.6084/m9.figshare.24578752.v1. URL: https://figshare.com/articles/dataset/CI_CD_repositories_with_tool_history/24578752.

[52] *GitHub Actions pipeline integration*. URL: https://codefresh.io/docs/docs/integrations/github-actions/ (visited on 12/06/2023).

[53] Mehdi Golzadeh, Alexandre Decan, and Tom Mens. “On the rise and fall of CI services in GitHub”. In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Honolulu, HI, USA: IEEE, Mar. 2022, pp. 662–672. ISBN: 978-1-66543-786-8. DOI: 10.1109/SANER53432.2022.00084. URL: https://ieeexplore.ieee.org/document/9825792/ (visited on 11/12/2023).

[54] Marvin Grieger et al. “Concept-Based Engineering of Situation-Specific Migration Methods”. In: *Software Reuse: Bridging with Social-Awareness*. Ed. by Georgia M. Kapitsaki and Eduardo Santana de Almeida. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 199–214. ISBN: 978-3-319-35122-3. DOI: 10.1007/978-3-319-35122-3_14.

[55] Richard Gronback. *Eclipse Modeling Project | The Eclipse Foundation*. URL: https://eclipse.dev/modeling/emf/ (visited on 12/27/2023).

[56] Mubin Ul Haque, Leonardo Horn Iwaya, and M. Ali Babar. “Challenges in Docker Development: A Large-Scale Study Using Stack Overflow”. In: *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ESEM ’20. Bari, Italy: Association for Computing Machinery, 2020. ISBN: 9781450375801. DOI: 10.1145/3382494.3410693. URL: https://doi.org/10.1145/3382494.3410693.

[57] Jordan Henkel et al. “Learning from, Understanding, and Supporting DevOps Artifacts for Docker”. In: *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. New York, NY, United States: Association for Computing Machinery, 2020, pp. 38–49. DOI: 10.1145/3377811.3380406.

[58] Willem-Jan van den Heuvel et al. “ChainOps for Smart Contract-Based Distributed Applications”. In: *Business Modeling and Software Design*. Ed. by Boris Shishkov. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2021, pp. 374–383. ISBN: 978-3-030-79976-2. DOI: 10.1007/978-3-030-79976-2_25.

[59] Michael Hilton et al. “Usage, costs, and benefits of continuous integration in open-source projects”. In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ASE’16: ACM/IEEE International Conference on Automated Software Engineering. Singapore Singapore: ACM, Aug. 25, 2016, pp. 426–437. ISBN: 978-1-4503-3845-5. DOI: 10.1145/2970276.2970358. URL: https://dl.acm.org/doi/10.1145/2970276.2970358 (visited on 12/17/2023).

[60] Helena Holmstrom et al. “Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance”. In: *2006 IEEE International Conference on Global Software Engineering (ICGSE’06)*. 2006, pp. 3–11. DOI: 10.1109/ICGSE.2006.261210.

[61] Geir Horn and Pawel Skrzypek. “MELODIC: Utility Based Cross Cloud Deployment Optimisation”. In: *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. 2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA). Krakow: IEEE, May 2018, pp. 360–367. ISBN: 978-1-5386-5395-1. DOI: 10.1109/WAINA.2018.00112. URL: https://ieeexplore.ieee.org/document/8418097/ (visited on 01/04/2024).

[62] Jerome Hugues et al. “TwinOps - DevOps meets model-based engineering and digital twins for the engineering of CPS”. In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS ’20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems. Virtual Event Canada: ACM, Oct. 16, 2020, pp. 1–5. ISBN: 978-1-4503-8135-2. DOI: 10.1145/3417990.3421446. URL: https://dl.acm.org/doi/10.1145/3417990.3421446 (visited on 12/07/2023).

[63] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, July 27, 2010. 956 pp. ISBN: 978-0-321-67022-9.

[64] *Introduction to CircleCI migration - CircleCI*. URL: https://circleci.com/docs/migration-intro/ (visited on 12/06/2023).

[65] Ramtin Jabbari et al. “What is DevOps? A Systematic Mapping Study on Definitions and Practices”. In: *Proceedings of the Scientific Workshop Proceedings of XP2016*. XP ’16 Workshops. Edinburgh, Scotland, UK: Association for Computing Machinery, 2016. ISBN: 9781450341349. DOI: 10.1145/2962695.2962707. URL: https://doi.org/10.1145/2962695.2962707.

[66] Santiago P. Jácome-Guerrero, Marcelo Ferreira, and Alexandra Corral. “Software Development Tools in Model-Driven Engineering”. In: *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*. 2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT). Mérida, Mexico: IEEE, Oct. 2017, pp. 140–148. ISBN: 978-1-5386-3956-6. DOI: 10.1109/CONISOFT.2017.00024. URL: http://ieeexplore.ieee.org/document/8337945/ (visited on 12/26/2023).

[67] *Jenkins pipeline integration/migration*. URL: https://codefresh.io/docs/docs/integrations/jenkins-integration/ (visited on 12/06/2023).

[68] Miguel Jiménez et al. “DevOps’ shift-left in practice: an industrial case of application”. In: *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment: First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers 1*. Springer. 2019, pp. 205–220.

[69] R. Kazman, S.G. Woods, and S.J. Carriere. "Requirements for integrating software architecture and reengineering models: CORUM II". In: *Proceedings Fifth Working Conference on Reverse Engineering (Cat. No.98TB100261)*. Proceedings Fifth Working Conference on Reverse Engineering (Cat. No.98TB100261). Oct. 1998, pp. 154–163. DOI: 10.1109/WCRE.1998.723185. URL: https://ieeexplore.ieee.org/abstract/document/723185 (visited on 02/02/2024).

[70] Abdelmadjid Ketfi and Noureddine Belkhatir. "Model-driven framework for dynamic deployment and reconfiguration of component-based software systems". In: *Proceedings of the 2005 symposia on Metainformatics - MIS '05*. the 2005 symposia. Esbjerg, Denmark: ACM Press, 2005, 8–es. ISBN: 978-1-59593-719-3. DOI: 10.1145/1234324.1234332. URL: http://portal.acm.org/citation.cfm?doid=1234324.1234332 (visited on 01/04/2024).

[71] Jörg Christian Kirchhof et al. "MDE for machine learning-enabled software systems: a case study and comparison of MontiAnna & ML-Quadrat". In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 2022, pp. 380–387.

[72] Jörg Christian Kirchhof et al. "MontiThings: Model-Driven Development and Deployment of Reliable IoT Applications". In: *Journal of Systems and Software* 183 (Jan. 2022), p. 111087. ISSN: 01641212. DOI: 10.1016/j.jss.2021.111087. URL: https://linkinghub.elsevier.com/retrieve/pii/S0164121221001849 (visited on 12/07/2023).

[73] Ankur Kumar, Mohammad Nadeem, and Mohammad Shameem. "Assessing the Maturity of DevOps Practices in Software Industry: An Empirical Study of HELENA2 Dataset". In: *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*. EASE '22. Gothenburg, Sweden: Association for Computing Machinery, 2022, pp. 428–432. ISBN: 9781450396134. DOI: 10.1145/3530019.3531335. URL: https://doi.org/10.1145/3530019.3531335.

[74] Rakesh Kumar and Rinkaj Goyal. "Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (ADOC)". In: *Computers & Security* 97 (Oct. 1, 2020), p. 101967. ISSN: 0167-4048. DOI: 10.1016/j.cose.2020.101967. URL: https://www.sciencedirect.com/science/article/pii/S0167404820302406 (visited on 01/03/2024).

[75] GitHub Staff Kyle Daigle. *Octoverse: The State of Open Source and rise of AI in 2023*. Nov. 2023. URL: https://github.blog/2023-11-08-the-state-of-open-source-and-ai/.

[76] Hemank Lamba et al. "Heard it through the Gitvine: an empirical study of tool diffusion across the npm ecosystem". In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software*

Engineering. ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Virtual Event USA: ACM, Nov. 8, 2020, pp. 505–517. ISBN: 978-1-4503-7043-1. DOI: 10.1145/3368089.3409705. URL: https://dl.acm.org/doi/10.1145/3368089.3409705 (visited on 02/01/2024).

[77] Juan De Lara, Esther Guerra, and Jesús Sánchez Cuadrado. "When and how to use multilevel modelling". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24.2 (2014), pp. 1–46.

[78] Pei Liu et al. "A First Look at CI/CD Adoptions in Open-Source Android Apps". In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. ASE '22. Rochester, MI, USA: Association for Computing Machinery, 2023. ISBN: 9781450394758. DOI: 10.1145/3551349.3561341. URL: https://doi.org/10.1145/3551349.3561341.

[79] Welder Pinheiro Luz, Gustavo Pinto, and Rodrigo Bonifácio. "Adopting DevOps in the real world: A theory, a model, and a case study". In: *Journal of Systems and Software* 157 (Nov. 2019), p. 110384. ISSN: 01641212. DOI: 10.1016/j.jss.2019.07.083. URL: https://linkinghub.elsevier.com/retrieve/pii/S0164121219301517 (visited on 01/04/2024).

[80] Leszek A. Maciaszek and Joaquim Filipe, eds. *Evaluation of Novel Approaches to Software Engineering: 9th International Conference, ENASE 2014, Lisbon, Portugal, April 28-30, 2014. Revised Selected Papers*. Vol. 551. Communications in Computer and Information Science. Cham: Springer International Publishing, 2015. ISBN: 978-3-319-27217-7. DOI: 10.1007/978-3-319-27218-4. URL: http://link.springer.com/10.1007/978-3-319-27218-4 (visited on 12/09/2023).

[81] Manar Majthoub, Mahmoud H. Qutqut, and Yousra Odeh. "Software Re-engineering: An Overview". In: *2018 8th International Conference on Computer Science and Information Technology (CSIT)*. 2018 8th International Conference on Computer Science and Information Technology (CSIT). Amman: IEEE, July 2018, pp. 266–270. ISBN: 978-1-5386-4152-1. DOI: 10.1109/CSIT.2018.8486173. URL: https://ieeexplore.ieee.org/document/8486173/ (visited on 12/26/2023).

[82] *Managing Plugins*. Managing Plugins. URL: https://www.jenkins.io/doc/book/managing/plugins/ (visited on 12/17/2023).

[83] *Manually migrating to GitHub Actions*. GitHub Docs. URL: https://docs.github.com/en/actions/migrating-to-github-actions/automated-migrations (visited on 12/06/2023).

[84] Fran Melchor et al. "A Model-Driven Approach for Systematic Reproducibility and Replicability of Data Science Projects". In: *Advanced Information Systems Engineering*. Ed. by Xavier Franch et al. Lecture Notes in Computer Science. Cham: Springer International

Publishing, 2022, pp. 147–163. ISBN: 978-3-031-07472-1. DOI: 10.1007/978-3-031-07472-1_9.

[85] Tom Mens and Pieter Van Gorp. "A Taxonomy of Model Transformation". In: *Electronic Notes in Theoretical Computer Science* 152 (Mar. 2006), pp. 125–142. ISSN: 15710661. DOI: 10.1016/j.entcs.2005.10.021. URL: https://linkinghub.elsevier.com/retrieve/pii/S1571066106001435 (visited on 12/20/2023).

[86] *metaDepth: A framework for deep meta-modelling*. URL: http://metadepth.org/ (visited on 12/27/2023).

[87] *MetaObject Facility | Object Management Group*. URL: http://www.omg.org/mof/ (visited on 12/27/2023).

[88] Bart Meyers et al. "A Model-Driven Engineering Framework to Support the Functional Safety Process". In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). Sept. 2019, pp. 619–623. DOI: 10.1109/MODELS-C.2019.00094. URL: https://ieeexplore.ieee.org/abstract/document/8904799 (visited on 01/03/2024).

[89] *Migrate from GitHub Actions - CircleCI*. URL: https://circleci.com/docs/migrating-from-github/ (visited on 12/27/2023).

[90] *Modelica*. URL: https://modelica.org/ (visited on 12/27/2023).

[91] *MPS: The Domain-Specific Language Creator by JetBrains*. JetBrains. URL: https://www.jetbrains.com/mps/ (visited on 01/04/2024).

[92] James Newkirk. "Introduction to agile processes and extreme programming". In: *Proceedings of the 24th international conference on Software engineering*. 2002, pp. 695–696.

[93] OASIS. *Topology and orchestration specification for cloud applications (TOSCA) Version 1.0, Committee Specification 01*. URL: http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html. (visited on 01/04/2024).

[94] Object Management Group. *OMG Unified Modeling Language Specification*. Mar. 2000.

[95] Efi Papatheocharous and Andreas S. Andreou. "Evidence of Agile Adoption in Software Organizations: An Empirical Survey". In: *Systems, Software and Services Process Improvement*. Ed. by Fergal McCaffery, Rory V. O'Connor, and Richard Messnarz. Vol. 364. Series Title: Communications in Computer and Information Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 237–246. ISBN: 978-3-642-39178-1. DOI: 10.1007/978-3-642-39179-8_21. URL: http://link.springer.com/10.1007/978-3-642-39179-8_21 (visited on 12/09/2023).

[96] R. Pérez-Castillo et al. "Reengineering technologies". In: *IEEE Software* 28.6 (2011), pp. 13–17. ISSN: 0740-7459. DOI: 10.1109/MS.2011.145.

[97] Puppet by Perforce. accessed: 16/11/2023. 2023. URL: https://www.puppet.com/resources/history-of-devops-reports.

[98] Puppet by Perforce. *State of Devops Report 2013*. 2013.

[99] Puppet by Perforce. *State of Devops Report 2015*. 2015.

[100] Puppet by Perforce. *State of Devops Report 2017*. 2017.

[101] Kai Petersen, Claes Wohlin, and Dejan Baca. "The Waterfall Model in Large-Scale Development". In: *Product-Focused Software Process Improvement*. Ed. by Frank Bomarius et al. Vol. 32. Series Title: Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 386–400. ISBN: 978-3-642-02151-0. DOI: 10.1007/978-3-642-02152-7_29. URL: http://link.springer.com/10.1007/978-3-642-02152-7_29 (visited on 12/09/2023).

[102] *Plan a migration from another tool to GitLab CI/CD | GitLab*. URL: https://docs.gitlab.com/ee/ci/migration/plan_a_migration.html (visited on 12/06/2023).

[103] *Puppet Infrastructure & IT Automation at Scale | Puppet by Perforce*. URL: https://www.puppet.com/ (visited on 01/04/2024).

[104] Thijs Reus, Hans Geers, and Arie van Deursen. "Harvesting Software Systems for MDA-Based Reengineering". In: *Model Driven Architecture – Foundations and Applications*. Ed. by Arend Rensink and Jos Warmer. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 213–225. ISBN: 978-3-540-35910-4. DOI: 10.1007/11787044_17.

[105] Franklin Magalhães Ribeiro et al. "A Model-Driven Solution for Automatic Software Deployment in the Cloud". In: *Information Technology: New Generations*. Ed. by Shahram Latifi. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2016, pp. 591–601. ISBN: 978-3-319-32467-8. DOI: 10.1007/978-3-319-32467-8_52.

[106] Luis F. Rivera et al. "UML-driven automated software deployment". In: *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*. CASCON '18. USA: IBM Corp., Oct. 29, 2018, pp. 257–268. (Visited on 01/03/2024).

[107] D. L. H. Rosenberg. *Software Re-engineering*. Goddard Space Flight Center, NASA.

[108] Pooya Rostami Mazrae et al. "On the usage, co-usage and migration of CI/CD tools: A qualitative analysis". In: *Empirical Software Engineering* 28.2 (Mar. 7, 2023), p. 52. DOI: 10.1007/s10664-022-10285-5. URL: https://doi.org/10.1007/s10664-022-10285-5.

[109] Winston W Royce. "Managing the development of large software systems: concepts and techniques". In: *Proceedings of the 9th international conference on Software Engineering*. 1987, pp. 328–338.

[110] Julio Sandobalin. "A Model-Driven Approach to Continuous Delivery of Cloud Resources". In: *Service-Oriented Computing – ICSOC 2017 Workshops*. Ed. by Lars Braubach et al. Vol. 10797. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 346–351. ISBN: 978-3-319-91763-4. DOI: 10.1007/978-3-319-91764-1_29. URL: https://link.springer.com/10.1007/978-3-319-91764-1_29 (visited on 12/07/2023).

[111] *Search*. GitHub Docs. URL: https://docs.github.com/en/rest/search?apiVersion=2022-11-28 (visited on 01/05/2024).

[112] S. Sendall and W. Kozaczynski. "Model transformation: the heart and soul of model-driven software development". In: *IEEE Software* 20.5 (Sept. 2003), pp. 42–45. ISSN: 0740-7459. DOI: 10.1109/MS.2003.1231150. URL: http://ieeexplore.ieee.org/document/1231150/ (visited on 12/20/2023).

[113] *Sirius - Eclipsepedia*. URL: https://wiki.eclipse.org/Sirius (visited on 12/27/2023).

[114] Eliezio Soares et al. "The effects of continuous integration on software development: a systematic literature review". In: *Empirical Software Engineering* 27.3 (May 2022), p. 78. ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-021-10114-1. URL: https://link.springer.com/10.1007/s10664-021-10114-1 (visited on 11/01/2023).

[115] Ian Sommerville. *Software engineering*. 9th ed. OCLC: ocn462909026. Boston: Pearson, 2011. 773 pp. ISBN: 978-0-13-703515-1.

[116] Hui Song et al. "Model-based fleet deployment in the IoT–edge–cloud continuum". In: *Software and Systems Modeling* 21.5 (Oct. 2022), pp. 1931–1956. ISSN: 1619-1366, 1619-1374. DOI: 10.1007/s10270-022-01006-z. URL: https://link.springer.com/10.1007/s10270-022-01006-z (visited on 12/07/2023).

[117] *Stack Overflow Insights - Developer Hiring, Marketing, and User Research*. URL: https://survey.stackoverflow.co/ (visited on 02/05/2024).

[118] Daniel Ståhl and Jan Bosch. "Modeling continuous integration practice differences in industry software development". In: *Journal of Systems and Software* 87 (Jan. 2014), pp. 48–59. ISSN: 01641212. DOI: 10.1016/j.jss.2013.08.032. URL: https://linkinghub.elsevier.com/retrieve/pii/S0164121213002276 (visited on 11/29/2023).

[119] *State of Continuous Delivery Report 2023: The Evolution of Software Delivery Performance*. CD Foundation. URL: https://cd.foundation/state-of-cd-2023/ (visited on 02/05/2024).

[120] Case Study. *Cloud native computing foundation*. Nov. 2023. URL: https://www.cncf.io/.

[121] *SysML Open Source Project: What is SysML? Who created it?* SysML.org. URL: https://sysml.org/index.html (visited on 12/27/2023).

[122] *TCS - Eclipsepedia*. URL: https://wiki.eclipse.org/TCS (visited on 12/27/2023).

[123] *The State of Developer Ecosystem in 2023*. JetBrains: Developer Tools for Professionals and Teams. URL: https://www.jetbrains.com/lp/devecosystem-2023 (visited on 02/01/2024).

[124] William M. Ulrich and Philip Newcomb. *Information Systems Transformation: Architecture-Driven Modernization Case Studies*. Google-Books-ID: hDzIedYPG7AC. Morgan Kaufmann, Feb. 4, 2010. 449 pp. ISBN: 978-0-08-095710-4.

[125] *Using scripts to test your code on a runner*. GitHub Docs. URL: https://docs.github.com/en/actions/examples/using-scripts-to-test-your-code-on-a-runner (visited on 12/06/2023).

[126] Bogdan Vasilescu et al. "Quality and productivity outcomes relating to continuous integration in GitHub". In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE'15: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. Bergamo Italy: ACM, Aug. 30, 2015, pp. 805–816. ISBN: 978-1-4503-3675-8. DOI: 10.1145/2786805.2786850. URL: https://dl.acm.org/doi/10.1145/2786805.2786850 (visited on 12/17/2023).

[127] Eelco Visser. "A Survey of Rewriting Strategies in Program Transformation Systems". In: *Electronic Notes in Theoretical Computer Science*. WRS 2001, 1st International Workshop on Reduction Strategies in Rewriting and Programming 57 (Dec. 1, 2001), pp. 109–143. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(04)00270-1. URL: https://www.sciencedirect.com/science/article/pii/S1571066104002701 (visited on 12/20/2023).

[128] Eclipse Web. *Eclipse OCL (Object Constraint Language)*. projects.eclipse.org. Jan. 31, 2013. URL: https://projects.eclipse.org/projects/modeling.mdt.ocl (visited on 12/27/2023).

[129] Eclipse Web. *Eclipse Xpand*. projects.eclipse.org. Jan. 31, 2013. URL: https://projects.eclipse.org/projects/modeling.m2t.xpand (visited on 12/27/2023).

[130] Eclipse Web. *Java Emitter Templates (JET2)*. projects.eclipse.org. Jan. 31, 2013. URL: https://projects.eclipse.org/projects/modeling.m2t.jet (visited on 12/27/2023).

[131] Frank Weil and LLC UniqueSoft. "Legacy Software Reengineering". In: *Unique Soft LLC* (2015).

[132] Michael Wenz. *Graphiti Home | The Eclipse Foundation*. URL: https://eclipse.dev/graphiti/ (visited on 12/27/2023).

[133] Johannes Wettinger et al. “Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel”. In: *Future Generation Computer Systems* 56 (Mar. 2016), pp. 317–332. ISSN: 0167739X. DOI: 10.1016/j.future.2015.07.017. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167739X15002496 (visited on 01/04/2024).

[134] David Gray Widder et al. “A conceptual replication of continuous integration pain points in the context of Travis CI”. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE ’19: 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Tallinn Estonia: ACM, Aug. 12, 2019, pp. 647–658. ISBN: 978-1-4503-5572-8. DOI: 10.1145/3338906.3338922. URL: https://dl.acm.org/doi/10.1145/3338906.3338922 (visited on 01/31/2024).

[135] David Gray Widder et al. “I’m leaving you, Travis: a continuous integration breakup story”. In: *Proceedings of the 15th International Conference on Mining Software Repositories*. ICSE ’18: 40th International Conference on Software Engineering. Gothenburg Sweden: ACM, May 28, 2018, pp. 165–169. ISBN: 978-1-4503-5716-6. DOI: 10.1145/3196398.3196422. URL: https://dl.acm.org/doi/10.1145/3196398.3196422 (visited on 12/17/2023).

[136] N. Wirth. *Algorithms + Data Structures = Programs*. Series In Automatic Computation. Prentice-Hall, 1976. URL: https://books.google.pt/books?id=XRhOxgEACAAJ.

[137] Emma Witman. *What is GitHub? How to start using the code hosting platform that allows you to easily manage and collaborate on programming projects*. Business Insider. URL: https://www.businessinsider.com/guides/tech/what-is-github (visited on 01/05/2024).

[138] Yiwen Wu et al. “An Empirical Study of Build Failures in the Docker Context”. In: *Proceedings of the 17th International Conference on Mining Software Repositories*. MSR ’20. Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 76–80. ISBN: 9781450375177. DOI: 10.1145/3379597.3387483. URL: https://doi.org/10.1145/3379597.3387483.

[139] Michael Wurster et al. “The essential deployment metamodel: a systematic review of deployment automation technologies”. In: *SICS Software-Intensive Cyber-Physical Systems* 35.1 (Aug. 2020), pp. 63–75. ISSN: 2524-8510, 2524-8529. DOI: 10.1007/s00450-019-00412-x. URL: http://link.springer.com/10.1007/s00450-019-00412-x (visited on 01/04/2024).

[140] *Xtext - Language Engineering Made Easy!* URL: https://eclipse.dev/Xtext/ (visited on 12/27/2023).

[141] Tianyin Xu and Darko Marinov. "Mining Container Image Repositories for Software Configuration and Beyond". In: *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*. ICSE-NIER '18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 49–52. ISBN: 9781450356626. DOI: 10.1145/3183399.3183403. URL: https://doi.org/10.1145/3183399.3183403.

[142] Mansooreh Zahedi, Roshan Namal Rajapakse, and Muhammad Ali Babar. "Mining Questions Asked about Continuous Software Engineering: A Case Study of Stack Overflow". In: *Proceedings of the Evaluation and Assessment in Software Engineering*. EASE '20. Trondheim, Norway: Association for Computing Machinery, 2020, pp. 41–50. ISBN: 9781450377317. DOI: 10.1145/3383219.3383224. URL: https://doi.org/10.1145/3383219.3383224.

[143] Liming Zhu, Len Bass, and George Champlin-Scharff. "DevOps and Its Practices". In: *IEEE Software* 33.3 (May 2016), pp. 32–34. ISSN: 0740-7459. DOI: 10.1109/MS.2016.81. URL: http://ieeexplore.ieee.org/document/7458765/ (visited on 12/07/2023).