Artifact Traceability in DevOps: An Industrial Experience Report

Zaki Pauzi*
BP plc
London, United Kingdom
zaki.pauzi@bp.com

Rajvir Thind
BP plc
London, United Kingdom
rajvir.thind@bp.com

Andrea Capiluppi
University of Groningen
Groningen, The Netherlands
a.capiluppi@rug.nl

ABSTRACT

In DevOps, the traceability of software artifacts is critical to the successful development and operation of project delivery to stakeholders. Before the introduction of end-to-end traceability in DevOps at a Data Analytics team at bp (BP plc), an international integrated energy company, the tracing of artifacts throughout a project life cycle was manual and time-consuming. This changed when traceability become more automated with end-to-end traceability capability as an offering on the platform. This paper reports on the ways of working and the experience of developers implementing DevOps for developing and putting in production a Javascript React web application, with a focus on traceability management of artifacts produced throughout the life cycle. This report highlights key opportunities and challenges in traceability management from the development stage to production.

CCS CONCEPTS

• **Software and its engineering** → **Agile software development**; *Programming teams*; • **Computer systems organization** → *Cloud computing*.

KEYWORDS

Industry, Software traceability, DevOps, Web application, Agile

ACM Reference Format:
Zaki Pauzi, Rajvir Thind, and Andrea Capiluppi. 2023. Artifact Traceability in DevOps: An Industrial Experience Report. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE '23), June 14–16, 2023, Oulu, Finland.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3593434.3593451

1 INTRODUCTION

In software engineering, the traceability of artifacts across the software life cycle is an established key continuous task that ensures the connections between artifacts are maintained appropriately [5, 15]. Traceability management is critical for a variety of reasons, such as for modelling dependencies between different components [8, 11], change impact analysis [1, 2], and compliance assessments by regulatory and certifying bodies [4, 6, 12]. Despite having established the need for traceability for decades, this continues to be a crucial component in practice today, particularly in DevOps [9].

This paper reports on the experience of two developers using a DevOps platform to develop and deploy a web app using React – a Javascript library – focusing on the traceability of the artifacts throughout the life cycle. The web app is an internal product of a Data Analytics team at bp that serves to visualise interactive graphs on team product offerings for clients, typically in use cases where Data Analytics and Science is used. The web app uses the open-sourced D3.js [1] with React [2] libraries. The developers' team provides Data Analytics as a service to clients, empowering decision-making and developing products within the Digital Technology space. Users of the web app are able to navigate and interact with custom diagrams to understand product offerings and engage with the developers' team on current and potential use cases in Data Analytics.

2 BACKGROUND

The evolution of agile software engineering continues in synergies explored with well-established concepts, such as usability and requirements engineering. In the mid-2010s, agile software engineering ventured beyond development, acknowledging operations alongside development, through DevOps [7]. The following agile principles resonate closely with the importance of traceability in DevOps:

- Welcome **changing** requirements, even late in development. Agile processes harness **change** for the customer's competitive advantage.
- Business people and developers must work **together** daily throughout the project.

In the continuous pursuit of managing change and advocating effective collaboration between stakeholders and developers, artifacts traceability in DevOps becomes critical to the success of product delivery. Artifacts traceability in DevOps has been garnering considerable interest recently with tools support (e.g., SAT-Analyzer [10, 13, 14]). For some current software-as-a-service (SaaS) solutions, end-to-end traceability is already a core feature implemented for users.

2.1 Before vs. Now

Before the current solution to collaborative software development in DevOps, traceability of artifacts in the developers' team used to be very manual and arduous. Manual labour effort was needed to trace requirements to code and other artifacts. Management of tasks was more time-consuming and without an automated traceability capability in place, collaboration in coding and testing was more

*Also affiliated with University of Groningen, email: a.z.bin.mohamad.pauzi@rug.nl

[1]https://d3js.org
[2]https://reactjs.org

EASE '23, June 14–16, 2023, Oulu, Finland
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0044-6/23/06.
https://doi.org/10.1145/3593434.3593451

challenging. Different parts of DevOps were split: for example, the code repository was disconnected from the Scrum/Kanban board and the responsibility of tagging requirements to code commits was solely on the developers' responsibility to do so, manually. Test cases had to be manually linked to the requirements and managers had less understanding of progress because requirements traceability was not well established and efforts in tagging for tracing were sporadic. Tracing these manually was seen as a chore: developers had little incentive to pick up on this, given the amount of time and effort that were deemed to be better prioritised for coding and testing.

A DevOps platform was recently introduced with end-to-end automated traceability capabilities in software development, which became an integral part of the DevOps process improvement offered by the platform vendors. This paper publishes our experience in using this DevOps platform to trace artifacts from business requirements (i.e., user stories) to code implementation (commits etc.) and test cases. The name of this DevOps platform cannot be revealed for anonymity, but the concepts and takeaways from the experience can be generalised to any DevOps platform with end-to-end traceability capabilities. Based on our experience, we highlight three key opportunities from established traceability in DevOps, along with two ongoing challenges, summarised in Section 4.

3 LEVERAGING TRACEABILITY USING DEVOPS PLATFORM

The DevOps platform is segmented into three main parts for the developer's role: i) Scrum board, ii) Code repository, and iii) Pipelines. Figure 1 shows a high-level overview of artifacts traced to one another from each of these. In this figure, the connectors are bi-directional; tracing can be done in both directions. The colours of these connectors are segregated according to the different streams of artifact traceability taking place, also denoted by the *Trace number* where each tracing is further detailed in the next sections.

3.1 Scrum board for Work Item management

Our team adopts the 'Scrum' approach [3] in DevOps. We run through iterations of Sprints led by a Scrum Master, made of plannings (prior to the next Sprint), stand-ups, reviews, and retrospectives (at the end of every Sprint). For this particular project, The Product Owner (key stakeholder) will join in some of the Scrum Events, especially on planning.

On the Scrum board, we have multiple buckets (columns) containing `Product Backlog Items` for every `User Story` (requirement): Backlog, In progress, Validate, and Done. The Scrum Master ensures the hygiene of the board and addresses blockers during stand-ups. All of these activities related to managing these Work Items can only be effective when traceability between artifacts is managed. On the Scrum board, the Work Items are the artifacts (i.e., cards on the board). These Work Items may be a `User Story`, `Product Backlog Item`, `Test Case`, `Bug`, or others.

There are three direct activity flows that involve tracing the artifacts (to and fro) within the Scrum board: (i) between requirements and code implementation, (ii) between requirements and testing, and (iii) between product backlog items and bugs. Requirements (both functional and non-functional) are mainly gathered in the initial stages of development, forming `User Stories` into `Product Backlog Items`, typically done during backlog refinement meetings.

3.2 User Story example: "User is able to export spreadsheets from graph visual diagram."

To further illustrate how traceability of artifacts is established in a requirement (`User Story`), we will be using the example of a user being able to export a graph into a spreadsheet for download, which is a functional requirement. This `User Story` will produce `Product Backlog Items` to achieve this requirement. The `Product Backlog Items` are "Button UI for export", and "Export function". These are also included with story points (useful for effort estimation) and priority number (ranking of priority to be given). These are important metrics in Sprint planning as the developers' capacity needs to be put into consideration against the estimated effort required for each `Product Backlog Item`; this is to avoid developers over- or under-commit in a Sprint. `Tasks` are then created for each of these `Product Backlog Items`, which serve the purpose of highlighting detailed tasks to complete.

3.2.1 Trace 1 (Green): Product Backlog Item ↔ Branch ↔ Pull Request. From these Product Backlog Items, we create branches for each. This ensures that the trace link between the `Product Backlog Item` and the `Branch` (in Code repository) is explicitly connected. Work is done on the `Branch` and `Commits` are made. Once complete, a `Pull Request` is raised and a code review is done between developers. The tracing of `Product Backlog Item`, `Branch`, and `Pull Request`, can be clearly seen on the DevOps platform. `Branches` are created by feature in our project. As such, we create an 'export-function' branch, which is done directly from the `Product Backlog Items`, and this can be traced both ways, including the Pull Request that is raised when the work is done. During our Sprint review, the `User Story` for this requirement is reviewed for progress.

3.2.2 Trace 2 (Red): Product Backlog Item ↔ Test Case. As the `Product Backlog Item` is created, a `Test Case` is also generated. This `Test Case` will need to be passed before the `Pull Request` can be approved. This transitive trace link between `Test case`, `Product Backlog Item`, and `Pull Request` ensures that code is tested prior to any approved Pull Requests. In this example, `Test Cases` of the "Export function" include automated unit testing to verify the specific components within the code.

3.2.3 Trace 3 (Blue): Product Backlog Item ↔ Bug. Bugs are reported through various means, for instance during user testing. These are Work Items generated and linked to new `Product Backlog Items` to address these `Bugs`. Tracing between those two artifacts is essential to keep track of bugs and planning the work in Sprint plannings to solve bugs on the go. By transitive links, the `Branch`, `Commits`, and `Pull Requests` generated due to the `Bug` are all visibly linked.

3.2.4 Transparency in linked artifacts. Effective traceability of requirements provides insights into key indicators such as readiness-to-ship requirements. Figure 2 is an example of the requirement traceability report that is monitored by the Project Manager and the

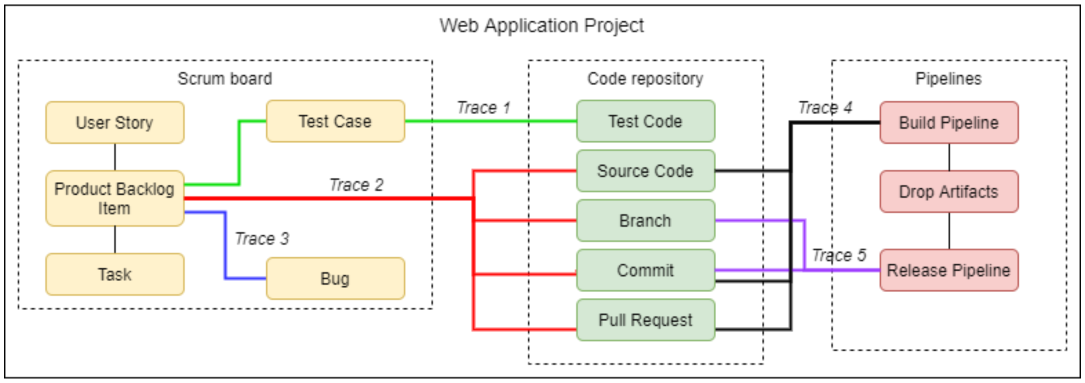
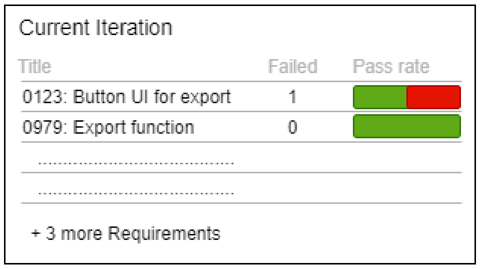


Figure 1: Overview of trace links between artifacts in Scrum board, Code repository, and Pipelines

Team Leader to track the progress of requirements. A key opportunity here is **Transparency**. Established visible traces of these Work Items can be seen by project stakeholders without having the need to raise a request and wait for the turnaround response from the Scrum Master. As such, that was the previous arrangement in place prior to using the end-to-end traceability the DevOps platform offers. On the flip side, the Product Owner for this project is now able to see and monitor the real-time progress of the Work Items through a dashboard because of the transparency of the linked Work Items.


Current Iteration

Title	Failed	Pass rate
0123: Button UI for export	1	🟩🟥
0979: Export function	0	🟩
....		
....		

+ 3 more Requirements

Figure 2: Requirements traceability report example

3.3 Code repository for version control and collaboration

Having the code repository fully integrated with the DevOps platform enabled seamless tracing of artifacts, allowing effective collaboration, tracking and reverting changes, if required. Tracing of artifacts within the repository, for example in branching, enabled us to handle conflicts easier during branch merges as code changes and modifications to files are clearly traced. A key opportunity here is **Effective code reviews**, where the developers can easily analyse and assess each others' code `Commits` and approve `Pull Requests` with a clear indication of what `Tasks` have been completed to complete a `Product Backlog Item`. Previously, this would have been done by manually checking which requirement each `Pull Request` is tagged to by adding notes in `Commit` descriptions. Code reviews were more complex because manual tracing needed to be done to identify the authors of the code and conflict resolutions would have need to be resolved manually.

3.4 CI/CD pipeline for build and release

An automated trigger is in place when `Pull Requests` are completed; the master branch is listened for any commits done shown by Trace 4 (black connector) at Figure 1. The `Build Pipeline` runs and builds the software code to generate a zipped folder of `Drop Artifacts`: this refers to the built version (binary files) of source code ready for deployment. Once the `Drop Artifacts` are generated, this triggers a `Release Pipeline` and the code is deployed to the cloud in separate environments: Dev, Test, Staging, and Prod, shown by Trace 5 (purple connector) at Figure 1.

The `Release Pipeline` deploys the code artifacts to a cloud platform app service through its default domain. The app can then be accessed by users through its registered URL. Traceability of these artifacts is end-to-end, although, within the cloud platform app service, none of the artifacts from the DevOps platform can be traced; it is one-way.

3.5 DevOps REST API

With established traceability in place, data about the management of artifacts is useful due to the links across the Scrum board, Code repository and Pipelines. A key opportunity here is **Analytics capability**, as data is consumed using a REST API endpoint. Examples of relevant data points would be the metadata of the Work Items, historical Sprints, and remaining requirements to be fulfilled. The API is used to provide the data for a monitoring dashboard to track the progress and state of the project throughout its life cycle. This was not possible previously, as the data on tracing these Work Items would not have been complete end-to-end, resulting in less accurate representations of the project state and progress. Manual tracking and tracing needed to be done to provide visibility on requirements completion, which is now, for the most part, automated.

4 KEY OPPORTUNITIES AND CHALLENGES

Based on our experience, we have identified three main opportunities in which artifact traceability in DevOps plays a key role, summarised in the following:

(1) *Transparency:* As per the agile manifesto on business people and developers working together daily, transparency of how requirements are traced to implementation and testing is critical for trust and assurance. With explicit traceability in

place, meetings about project progress are quick and effective, as stakeholders themselves already have visibility of `Work Items` traced to other artifacts.

(2) *Effective code reviews:* Developers on the team have visibility of `Pull Requests` and the code `Commits` done. Reviewing each others' codes is an essential component of quality control and best practices. The result of reviewing code in the team is to identify imminent bugs, increase code quality, and help other developers learn the source code. With traceability already in place, we did not have to do a tracing exercise like previously, for example, reading comments in the source code to analyse which requirement a function or a class is referring to.

(3) *Analytics capability:* Managers of teams and projects are able to derive insights when Work Items are effectively linked and traced. For example, measuring Sprint velocity, code coverage, and requirements traceability matrix. Customised dashboards with visuals of these have helped the managers to make informed decisions and address any impediments swiftly, where necessary. Analytics of past projects also helps to estimate future resourcing needs for upcoming projects, which is crucial to budgeting.

4.1 Ongoing challenges

Within opportunities, there are ongoing challenges that arise with end-to-end traceability in the use of DevOps within the team, namely in extra efforts of developers maintaining **Hygiene** and dissemination of **Knowledge**.

(1) *Hygiene:* Traceability can only be effective when everyone plays a role in maintaining the *cleanliness* and adhering to the ways of working. For example, a not-so-helpful `Commit` description leads to a vague `Pull Request` and time wasted on meetings to clarify work done. To address this, consistent quality checks were done by the Scrum Master to ensure that everything is in place and rules and procedures were followed. In situations where this was not the case, issues were raised during Sprint retrospectives, which serve as a checkpoint of lessons learned throughout the developers' experiences throughout the Sprint. During the retrospective, improvement in the ways of working is discussed.

(2) *Knowledge:* Every project stakeholder may not necessarily be on the same level of understanding regarding the ways of working, like formatting of `Work Items` on the Scrum board. To address this, a working document serving as a Wiki was put in place as a reference point for any new members, and even existing ones. This Wiki covers important aspects of using DevOps effectively for various roles. For the developers like ourselves, we use it to document how code reviews are done, formatting of Code repository artifacts, running tests etc. The Wiki also contains guidelines for the Scrum Master to conduct the Scrum Events throughout Sprints and caters for the stakeholders from the business (such as product sponsors). This Wiki frequently gets reviewed and updated, where necessary. Lessons learned during Sprint retrospectives are used to review ways of working and eventually, the Wiki gets updated after agreement.

5 CONCLUSION

This paper reports on our experience working on a web app development using a DevOps platform, with a focus on artifact traceability from requirements gathering to deployment. Throughout this, we have highlighted key opportunities in established artifact traceability capabilities in DevOps: Transparency, Effective code reviews, and Analytics capability. Beyond the opportunities highlighted, key metrics to report for future work are quantitative measurements (or estimates) of end-to-end traceability effects (e.g., time saved). We have also identified ongoing challenges that need to be addressed, namely on maintaining Hygiene and dissemination of Knowledge. This industrial experience report serves as a guide for others to leverage and harness the key opportunities that DevOps has to offer in artifact traceability.

REFERENCES

[1] Robert S Arnold. 1996. *Software change impact analysis.* IEEE Computer Society Press.

[2] Thazin Win Win Aung, Huan Huo, and Yulei Sui. 2020. A Literature Review of Automatic Traceability Links Recovery for Software Change Impact Analysis. In *Proceedings of the 28th International Conference on Program Comprehension* (Seoul, Republic of Korea) *(ICPC '20).* Association for Computing Machinery, New York, NY, USA, 14–24. https://doi.org/10.1145/3387904.3389251

[3] Mike Beedle, Martine Devos, Yonat Sharon, Ken Schwaber, and Jeff Sutherland. 1999. SCRUM: An extension pattern language for hyperproductive software development. *Pattern languages of program design* 4, 1 (1999), 637–651.

[4] Fergal Mc Caffery, Valentine Casey, M. S. Sivakumar, Gerry Coleman, Peter Donnelly, and John Burton. 2012. *Medical Device Software Traceability.* Springer London, London, 321–339. https://doi.org/10.1007/978-1-4471-2239-5_15

[5] Orlena Gotel, Jane Cleland-Huang, J Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, and Giuliano Antoniol. 2012. The quest for ubiquity: A roadmap for software and systems traceability research. In *2012 20th IEEE international requirements engineering conference (RE).* IEEE, 71–80.

[6] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. 2017. Semantically enhanced software traceability using deep learning techniques. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE).* IEEE, 3–14.

[7] Rashina Hoda, Norsaremah Salleh, and John Grundy. 2018. The Rise and Evolution of Agile Software Development. *IEEE Software* 35, 5 (2018), 58–63. https://doi.org/10.1109/MS.2018.290111318

[8] Hongyu Kuang, Jia Nie, Hao Hu, Patrick Rempel, Jian Lü, Alexander Egyed, and Patrick Mäder. 2017. Analyzing closeness of code dependencies for improving IR-based Traceability Recovery. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER).* 68–78. https://doi.org/10.1109/SANER.2017.7884610

[9] Dulani Meedeniya, Iresha Rubasinghe, and Indika Perera. 2020. Artefact consistency management in DevOps practice: A survey. In *Tools and Techniques for Software Development in Large Organizations: Emerging Research and Opportunities.* IGI Global, 98–129.

[10] S. Palihawadana, C. H. Wijeweera, M. G. T. N. Sanjitha, V. K. Liyanage, I. Perera, and D. A. Meedeniya. 2017. Tool support for traceability management of software artefacts with DevOps practices. In *2017 Moratuwa Engineering Research Conference (MERCon).* 129–134. https://doi.org/10.1109/MERCon.2017.7980469

[11] Reza Meimandi Parizi, Sai Peck Lee, and Mohammad Dabbagh. 2014. Achievements and challenges in state-of-the-art software traceability between test and code artifacts. *IEEE Transactions on Reliability* 63, 4 (2014), 913–926.

[12] Gilbert Regan, Fergal Mc Caffery, Kevin Mc Daid, and Derek Flood. 2013. Medical device standards' requirements for traceability during the software development lifecycle and implementation of a traceability assessment model. *Computer Standards & Interfaces* 36, 1 (2013), 3–9. https://doi.org/10.1016/j.csi.2013.07.012

[13] Iresha Rubasinghe, Dulani Meedeniya, and Indika Perera. 2018. Automated Inter-artefact Traceability Establishment for DevOps Practice. In *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS).* 211–216. https://doi.org/10.1109/ICIS.2018.8466414

[14] Iresha Rubasinghe, Dulani Meedeniya, and Indika Perera. 2021. SAT-analyser traceability management tool support for DevOps. *Journal of Information Processing Systems* 17, 5 (2021), 972–988.

[15] George Spanoudakis and Andrea Zisman. 2005. Software traceability: a roadmap. In *Handbook of software engineering and knowledge engineering: vol 3: recent advances.* World Scientific, 395–428.