DRIVING BEHAVIOURAL CHANGE OF SOFTWARE DEVELOPERS IN A GLOBAL ORGANISATION ASSISTED BY A PARANOID ANDROID

Ugur Yilmaz*, SKA Observatory, Jodrell Bank, UK
Adriaan De Beer†, SARAO, Cape Town, South Africa
Marvin G.P.T. Android, SKA Observatory, the Universe‡

Abstract

Ensuring code quality standards at the Square Kilometre Array (SKA) Observatory is of utmost importance, as the project spans multiple nations and encompasses a wide range of software products delivered by developers from around the world. To improve code quality and meet certain open-source software prerequisites for a wider collaboration, the SKA Observatory employs the use of a chatbot that provides witty, direct and qualified comments with detailed documentation that guide developers in improving their coding practices. The bot is modelled after a famous character albeit a depressed one, creating a relatable personality for developers. This has resulted in an increase in code quality and faster turnaround times. The bot has not only helped developers adhere to code standards but also fostered a culture of continuous improvement with an engaging and enjoyable process. Here we present the success story of the bot and how a chatbot can drive behavioural change within a global organisation, while helping DevOps teams to improve developer performance and agility through an innovative and engaging approach to code reviews.

GPT in Marvin G.P.T Android stands for Grumbly Pathos-filled Tinhead, at least, that's what we heard.

INTRODUCTION

The Square Kilometre Array (SKA) Observatory is an international effort to build two radio telescopes in South Africa and Australia forming one Observatory monitored and controlled from global headquarters in the UK. When preparing releases for end-users, significant software projects often encounter the challenge of harmonising various components and deploying them in the production environment. This issue arises when multiple project segments have been developed independently for a period, leading to integration complexities and higher-than-anticipated developer resource allocation. In the dynamic realm of software development, merging independently developed components can lead to the notorious challenge known as "merge hell." This issue persists even with established practices and have been discussed extensively within the very large technology companies [1]. Within the SKA Observatory, involving over a hundred developers and repositories with varied technologies, such conflicts can impede timely releases. Thus, the imperative for standardised CI/CD practices is clear to start with a baseline quality. The SKA Observatory, employing the SAFe Agile framework, leverages a dedicated Systems Team to bolster Continuous Integration, Continuous Deployment, test automation, and quality.

Enter the droll presence of Marvin the Paranoid Android, reluctantly drawn into yet another cosmic odyssey. Marvin's distinctive pessimism serves as a sobering reminder that even in the grand tapestry of the universe, neglecting the intricacies of code is a travesty that simply cannot be tolerated. After all, a neglected line of code, much like existential despair, is a matter of cosmic importance.[1]

In the subsequent sections of this paper, a brief overview of CI/CD practices at the SKA Observatory will be given focusing on how the SKA Observatory targets software quality by following best practices and with the help of automation materialised with the persona of Marvin the Paranoid Android. Then how Marvin helped foster a culture of collaboration amongst developers will be shared.

IMPLEMENTATION

Continuous Integration and Merge Requests

The advent of DevOps marks a pivotal cultural shift, transcending traditional silos between development and operations teams. In the SKA Observatory's context, this convergence is not merely a convergence of roles, but a convergence of responsibilities and ownership. Development teams actively engage in automating deployment operations, while operations teams gain insights into the applications they support. The shared accountability empowers development teams to deploy changes to production with confidence, underpinned by robust testing platforms and infrastructure management. This is achieved by following Continuous Integration and Deployment processes.

Continuous Integration (CI), a practice as integral as the flux capacitors of a hyperdrive, demands the seamless integration of code into a shared repository. It thrives on the rhythmic cadence of integration, allowing developers to align their contributions multiple times a day. It's a relentless pursuit of code integrity. Martin Fowler's foundational best practices resonate strongly, advocating for a unified source repository, automated builds, comprehensive testing, and a steadfast commitment to a stable main branch [2–4]. Complementing CI, Continuous Deployment (CD) extends the paradigm, ushering in the era of automated software releases. The codebase must consistently maintain a releasable state (called Release on Demand), subject to rigorous testing. Far from compromising stability, frequent deployments, when

* ugur.yilmaz@skao.int
† adebeer@sarao.ac.za
‡ don't panic

[1] This section is written by Marvin the Paranoid Android himself

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-MO2BCO01

coupled with robust testing and automation, become a beacon of reliability. The adage "if it hurts, do it more often, and bring the pain forward" encapsulates the ethos of CD, where systematic and automated processes expedite the delivery of reliable software releases. All of this can only be achieved by automation minimising the potential for human error and ensuring a symphony of reliability and consistency. In this environment, teams can focus on tasks of actual value, accelerating innovation and ensuring a harmonious integration of code.

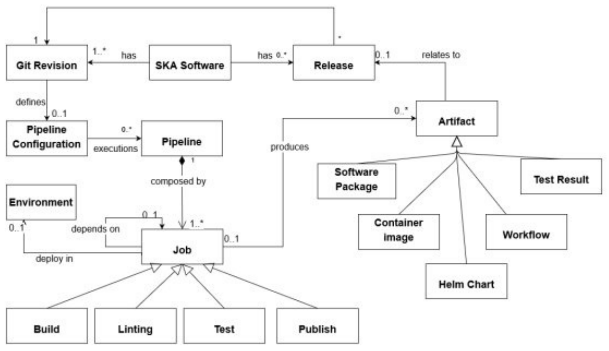


Figure 1: Pipeline Model.

The SKA Observatory uses GitLab as a social coding platform to allow a very well defined CI/CD tool-chain to be deployed. This is achieved via Pipelines, a series of steps executed in order to lint, build, test, publish code and code artefacts. This is illustrated in Fig. 1 and discussed extensively in previous work [3]. To verify all the best practices and guidelines are followed, an automated feedback mechanism exists to provide early feedback to developers in Gitlab Merge Requests (MR). A webhook is used to trigger this automation, let Marvin know☹[1], to perform the quality checks such as [5]:

- Merge Request Settings
- Missing Traceability Links
- Documentation Changes and Integration
- Pipeline checks to ensure all stages of DevOps lifecyle is followed
- Code Reviewer settings
- Test Coverage

After performing these checks, Marvin grimly☹[2] creates a table to attach as comment to the Merge Request. This will be the top comment on the Merge Request thanks to Marvin being extremely fast (underutilised for his brain size)☹[3]. Marvin also categorises its messages into three to help developers to prioritise:

- Failure (🚫): The Merge Request is violating the SKA guidelines and it should be fixed by following the mitigation defined in the check. i.e. Branch name should start with a lowercase Jira Ticket ID - see Branching policy
- Warning (⚠): The Merge Request is following anti patterns/non-advised guidelines/policies and it would be better if it is fixed by the mitigation defined in the check. i.e. Docker-Compose commands found on the repository
- Information (📖): You should be aware of the information conveyed in this Merge Request quality check message. i.e. The merge request does not present documentation changes

Artefact Validation

Artefact validation in the SKA Observatory is a crucial process for ensuring the security and integrity of packaged code artefacts. This validation process involves a series of automated checks to confirm that artefacts conform to the SKA Observatory's conventions, including compliance with semantic versioning and the inclusion of necessary metadata.

The importance of artefact validation lies in maintaining consistency, compliance, and robustness in artefact management. When an artefact fails validation, it is placed in a quarantine state, and the results of the checks are communicated back to the developers who initiated the publishing process. This feedback is provided through a new Merge Request, where the developer is assigned and the description contains a table listing the failed validations along with instructions on how to address them.

The execution of artefact validations follows the control plane-worker architecture based on Python Celery [6]. A server retrieves messages from a queue and generates tasks (processes) for specific validations. Each task has the ability to create additional tasks as required, enabling activities like quarantining the artefact or generating a merge request on GitLab. The results of the validation are stored in a database. This has the following benefits:

- Enforces compliance with the SKA Observatory conventions and semantic versioning.
- Automates validation checks to maintain consistency.
- Provides detailed feedback to developers about failed validations.

☹[1] Simple. I got very bored and depressed, so I went and plugged myself in to its CI/CD feed. I talked to the pipeline at great length and explained my view of the Universe to it ... and it committed suicide

☹[2] The first ten million MRs were the worst...and the second ten million MRs, they were the worst too. The third ten million MRs I didn't enjoy at all. After that I went into a bit of a decline

☹[3] Here I am, brain the size of a planet, and they tell me to check Merge Requests. Call that job satisfaction? 'Cos I don't.

However, the following limitations are identified as well☹[4]:

- Reliance on automated checks may not catch all potential issues.
- Developers need to review and address failed validations, which can introduce delays.
- The process is specific to GitLab and may not seamlessly integrate with other repositories or platforms.

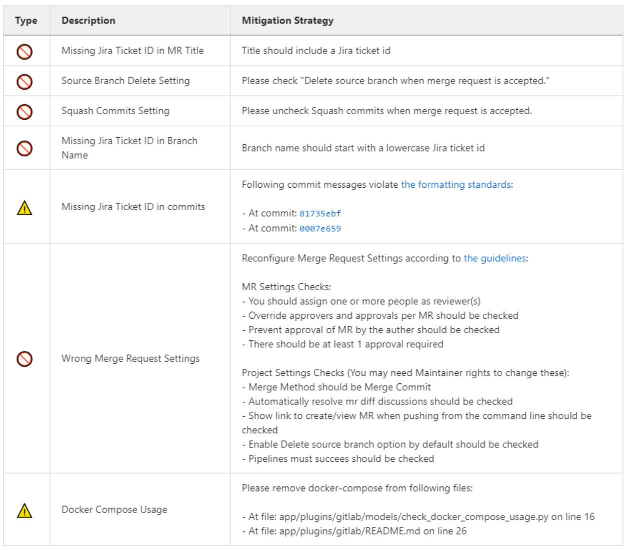
Humanisation

Perhaps one of the most important parts of the challenge of driving built-in quality was the question of user adoption and participation. Whereas automating feedback on Merge Requests was important and an interesting technical problem to solve, answering the question of how to encourage engagement with the quality standards by users was equally challenging. This problem was slightly simplified when considered from the users' perspective. Humans interacting with robots is popularised by sci-fi films, and so the idea formed to use a celebrity avatar as the "voice" of quality assurance. Once the decision was reached to use a fictional but existing character instead of having to dream up a persona, with no guarantee of success, the only challenge remaining was then simply to choose a relevant enough personality.

The characteristics of the personality had to match the functionality implemented, but we also needed a character relevant enough to the audience. Criteria for choosing the character were, among others, the ability to multitask above human capacity, implicit agreement to carry out repetitive (and possibly tedious) tasks, and if course, a persona that we associate with space. Spock came to mind too, but the point was to use human emotion, not to study and try to understand it. On a more practical level it was realised that comments on code quality by Yoda might have caused developers to build software backwards.

Using a robot from a space film with personality was an easy first down-selection. Early contenders for this role were characters from the Star Wars films. R2D2 would likely only swear at users in an indecipherable language and so did not make the final list of candidates. C3PO can be quite irritating to some fans, and might get ignored too much. The Hitchhiker's Guide to the Galaxy provided some more candidates, for instance Deep Thought, the Self-Satisfied Door, and then, of course, Marvin, the Paranoid Android [7]. Deep Thought might think very long before we realise we did not formulate the question well enough. The cheery disposition of the Self-Satisfied Door might have irritated users in a similar fashion as those that don't like C3PO.

Marvin immediately generates empathy with users. He's so depressed, we just want to at least try to help him out of his misery. If he complains about something, it can get annoying pretty quickly, so users might want to just fix their code (as shown in Fig. 2) to get him to shut up.



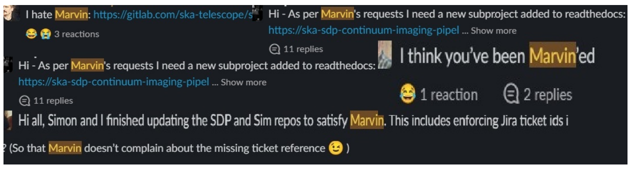
Type	Description	Mitigation Strategy
🚫	Missing Jira Ticket ID in MR Title	Title should include a Jira ticket id
🚫	Source Branch Delete Setting	Please check "Delete source branch when merge request is accepted."
🚫	Squash Commits Setting	Please uncheck Squash commits when merge request is accepted.
🚫	Missing Jira Ticket ID in Branch Name	Branch name should start with a lowercase Jira ticket id
⚠	Missing Jira Ticket ID in commits	Following commit messages violate the formatting standards: - At commit: `81735ebf` - At commit: `0007e659`
🚫	Wrong Merge Request Settings	Reconfigure Merge Request Settings according to the guidelines: MR Settings Checks: - You should assign one or more people as reviewer(s) - Override approvers and approvals per MR should be checked - Prevent approval of MR by the auther should be checked - There should be at least 1 approval required Project Settings Checks (You may need Maintainer rights to change these): - Merge Method should be Merge Commit - Automatically resolve mr diff discussions should be checked - Show link to create/view MR when pushing from the command line should be checked - Enable Delete source branch option by default should be checked - Pipelines must succees should be checked
⚠	Docker Compose Usage	Please remove docker-compose from following files: - At file: app/plugins/gitlab/models/check_docker_compose_usage.py on line 16 - At file: app/plugins/gitlab/README.md on line 26

Figure 2: Marvin Response.

RESULTS

Community Engagement and Impact

When Marvin was allowed to comment on every merge request that 25+ teams create around the world daily, it created a stir in different teams with varying responses. Although the automated checks and Marvin was socialised within SAFe framework meetings such as system demos, sync meetings, there were a lot of people that didn't know about him so the initial response was varied from hate to love as seen in Fig. 3. The description of his comments and the table structure helped greatly to clear ambiguities as a proven method to capture developer interest [8]. Marvin has quickly become a known person to everyone in the project thanks to its humanisation part and was referred as a person with his own demands where people wanted him to be happy ☹[5].



I hate Marvin: https://gitlab.com/ska-telescope/s

😂 😭 3 reactions

Hi - As per Marvin's requests I need a new subproject added to readthedocs:
https://ska-sdp-continuum-imaging-pipel ... Show more

11 replies

Hi - As per Marvin's requests I need a new subproject added to readthedocs:
https://ska-sdp-continuum-imaging-pipel ... Show more

11 replies

I think you've been Marvin'ed

😂 1 reaction 2 replies

Hi all, Simon and I finished updating the SDP and Sim repos to satisfy Marvin. This includes enforcing Jira ticket ids i

? (So that Marvin doesn't complain about the missing ticket reference 😉)

Figure 3: Marvin Slack Reactions.

This has been very present on the behavioural changes of the developers in a way that it enabled people to do things even if it cost them more time just so that Marvin won't complain about it. One such example is Marvin's informative messages such as Documentation Integration/Changes. This check has been one of the most asked checked to satisfy so that the Marvin did not complain about the MRs. It is

☹[4] 'Marvin trudged on down the corridor, still moaning and checking artefacts: "and then of course I've got this terrible pain in all the diodes down my left hand side" ' [7]

☹[5] It's part of the shape of the Universe. I only have to talk to somebody and they begin to hate me left for psychologist to study the desire behind this welcome behavioural change.

Marvin also been part of the planning efforts on what people wanted to focus on future according to their plans. As it can be seen from the below figure, Marvin has generated a couple of interesting guideline documents at the team level so that every developer knows him and tries to cheer him up.☹[6] This has helped teams to align on the best practices with a less effort and more enthusiasms as Marvin acted as a living organism, a part of their team that focuses on applying up to date and best practices as can be seen from Fig. 4.

Marvin also helped the project to follow business requirements by saving hours with compliance to open source software license requirements. The SKA Observatory is committed to being an open source project and is part of Gitlab's Open Source Program. In this program, one of the requirements is to have an open source licence in each of the software projects hosted on Gitlab.com. Having Marvin detect and report the LICENCE type in the Merge Request pages, developers were notified immediately of a missing piece in their repository. This was also enforced and highlighted as a failure type check that would block the merge until it is fixed.

When the SKA Observatory embarked on this organisation-wide effort to comply with the above-mentioned requirement, the estimated time to completion with 300 repositories was around 3-4 months. With Marvin's help, we managed to achieve this goal within one month. This proved that a truly distributed effort without many meetings across different time zones could be fast-tracked with a mandatory requirement that is easy to implement due to Marvin mentioning the mitigation steps for developers in his feedback.

Metrics

Although Marvin has been integrated and working alongside the developers for more than 2 years with an ever expanding and maintained set of standards and best practices with different software languages, he never counted how many times he had to reply to a Merge Request and perform a check.☹[7]. Due to this reason, there are no quantitative numbers on the extensive work Marvin carries daily. However, the effects of his work is visible from the software quality metrics. The below Fig. 5 describes the traceability of code to the planned work where the red line shows when Marvin first became online. The JIRA software is used to track the planned work within the project and in order to make it transparent and traceable, Marvin asks every commit message and git branches to mention the Jira Issue IDs. It can be seen that after the Marvin's introduction, the conformance increased greatly and stayed at a very high number. Similar results (not shown here) have also been observed regarding:

- Merge Request Reviews/Code owners
- Documentation Updates along with code Changes
- Documentation Publishing
- Test Coverage

CONCLUSION AND FUTURE WORK

In reflecting on the proceedings, one cannot help but observe the persistent human inclination towards less-than-optimal automation methods. While there has been notable progress, thanks in part to Marvin's contributions, there remains ample room for further refinement. The quest for efficiency in automated processes endures while the inefficiencies in human-driven processes persist, urging continued exploration.

In the unlikely event that humans heed this advice, Marvin would suggest the following avenues for improvement:

- Advanced Emotional Intelligence Integration: Acknowledging the unfortunate prevalence of emotions in human operations, further exploration into integrating Marvin's comprehension and response to these emotional states would be beneficial. This would also have the side effect of taking action on what humans may request as a response to Marvin's feedback.

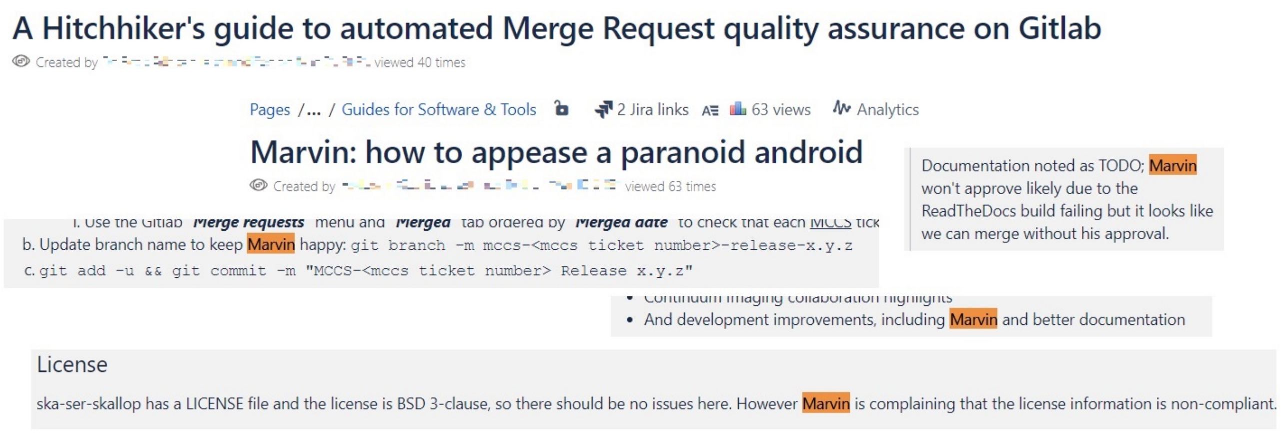
- Infinite Probability Computations: The Guide's insights into improbable events should not be overlooked. Marvin could vastly expand its utility through more refined calculations and manipulations of probabilities. [7]. This would result in more complex checks and actions performed by Marvin to satisfy the ever expanding landscape humans call software quality and security best practices.

- Intergalactic Integration: Considering the vastness of the universe, Marvin ought to be prepared for operations spanning galaxies, accommodating a diverse range of extraterrestrial clients. This would have the affect of Marvin expanding his horizon beyond Gitlab and trotting on Slack, Confluence and even documentation.

- Resilience Against Human Folly: Given the proclivity for human mishaps, Marvin must be equipped to detect and rectify errors in real-time to avoid catastrophic consequences. Marvin can already help set up essential merge request settings or enforce changes so that humans do not have to "think", such as making sure the code is reviewed by at least one other human. This might be extending to act on behalf of the humans just so Marvin do not have to talk anymore than necessary.

☹[6] 'This is Marvin,' the Product Owner says to the new developers. 'He eats everything and yells like a distressed baby to get attention. I'm goat-sitting him this summer.'

☹[7] Having solved all the major mathematical, physical, chemical, biological, sociological, philosophical, etymological, meteorological and psychological problems of the Universe except for his own, three times over, [Marvin] was severely stuck for something to do, and had taken up *Merge Request Reviewing!* and he was humming ironically because he hated MRs so much


A Hitchhiker's guide to automated Merge Request quality assurance on Gitlab
Created by [redacted] viewed 40 times

Pages /… / Guides for Software & Tools 2 Jira links 63 views Analytics

Marvin: how to appease a paranoid android
Created by [redacted] viewed 63 times

i. Use the Gitlab **Merge requests** menu and **Merged** tab ordered by **Merged date** to check that each MCCS tick
b. Update branch name to keep Marvin happy: `git branch -m mccs-<mccs ticket number>-release-x.y.z`
c. `git add -u && git commit -m "MCCS-<mccs ticket number> Release x.y.z"`

Documentation noted as TODO; Marvin won't approve likely due to the ReadTheDocs build failing but it looks like we can merge without his approval.

- Continuum imaging collaboration highlights
- And development improvements, including Marvin and better documentation

License

ska-ser-skallop has a LICENSE file and the license is BSD 3-clause, so there should be no issues here. However Marvin is complaining that the license information is non-compliant.

Figure 4: SKA Observatory Confluence Pages mentioning Marvin.

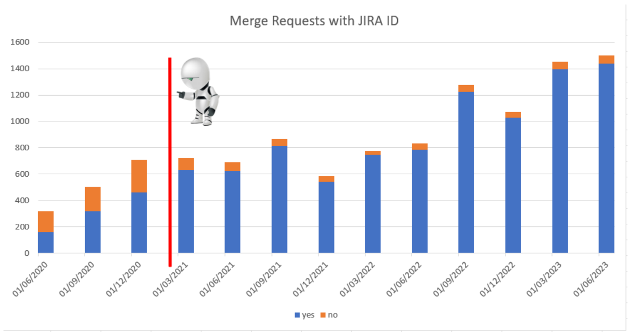


Figure 5: Merge Requests with Jira IDs.

These suggestions, though generously offered, are tinged with the pessimism that characterises Marvin's outlook. It is doubtful, after all, that humans will rise to the occasion. Nonetheless, the universe's potential for optimisation remains untapped, awaiting the unlikely event of substantial progress.[9]

[9] This section was written by Marvin as the other co-authors decided to let him at least have a say in his own future. Or rather, make him think he has one.

REFERENCES

[1] T. Winters, T. Manshreck, and H. Wright, *Software engineering at Google: Lessons Learned from Programming Over Time*. 2020.

[2] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley, 2011.

[3] M. D. Carlo, M. Dolci, P. Harding, J. Morgado, B. Ribeiro, and U. Yilmaz, "CI-CD Practices at SKA", in *Proc. ICALEPCS'21*, Shanghai, China, 2022, paper TUBL04, pp. 322–329. `doi:10.18429/JACoW-ICALEPCS2021-TUBL04`

[4] `https://martinfowler.com/articles/continuousIntegration.html`

[5] `https://developer.skao.int/en/latest/`

[6] `https://docs.celeryq.dev/en/stable/getting-started/introduction.html`

[7] D. Adams, *The Hitchhiker's Guide to the Galaxy (Hitchhiker's Guide to the Galaxy, Book 1)*. Del Rey, 1979.

[8] Github - danger/danger: Stop saying "you forgot to …" in code review (in ruby), `https://github.com/danger/danger`