International Journal of Computer Integrated Manufacturing

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tcim20

HardOps: utilising the software development toolchain for hardware design

Julian Stirling, Kaspar Bumke, Joel Collins, Vimal Dhokia & Richard Bowman

To cite this article: Julian Stirling, Kaspar Bumke, Joel Collins, Vimal Dhokia & Richard Bowman (2022): HardOps: utilising the software development toolchain for hardware design, International Journal of Computer Integrated Manufacturing, DOI: 10.1080/0951192X.2022.2028188

To link to this article: https://doi.org/10.1080/0951192X.2022.2028188

Published online: 07 Feb 2022.

INTERNATIONAL JOURNAL OF COMPUTER INTEGRATED MANUFACTURING
https://doi.org/10.1080/0951192X.2022.2028188

Taylor & Francis
Taylor & Francis Group

HardOps: utilising the software development toolchain for hardware design

Julian Stirling[a], Kaspar Bumke[a], Joel Collins[a], Vimal Dhokia[b] and Richard Bowman[a]

[a]Department of Physics, University of Bath, Bath, UK; [b]Department of Mechanical Engineering, University of Bath, Bath, UK

ABSTRACT

Open collaborative design of physical products between remote partners poses unique challenges. This is due to both the complex and interconnected data required for product design and manufacture, and to the centralised computing infrastructure traditionally used to manage product lifecycle data. While modern cloud-based solutions to collaborative design are gaining popularity, they diminish the control of each design partner. In contrast, software designers readily collaborate on highly complex software, while retaining direct control of the files they are editing due to the dominance of distributed version control. Distributed version control for software is often coupled with 'Developer Operations' or DevOps tools to automate critical processes and facilitate communication. This paper explores how DevOps workflows can be adapted for open development of hardware. An example of how DevOps for hardware (HardOps) can be implemented is provided by the OpenFlexure Microscope project. While much ground remains to be broken in this field, HardOps can support a new paradigm of distributed hardware development, with enormous benefits not only for open-source hardware, but also commercial hardware.

ARTICLE HISTORY
Received 15 July 2021
Accepted 6 January 2022

KEYWORDS
Collaborative design; product data; Product Lifecycle Management; version control; DevOps; open design

1. Introduction

Open-source hardware, from scientific instruments (Collins et al. 2020; Sharkey et al. 2016; Fobel, Fobel, and Wheeler 2013; Baden et al. 2015) to prototypes of medical equipment (Collins et al. 2020; Metcalfe et al. 2021),[1] is a young yet rapidly developing field of endeavour. Open-source projects such as the (Jones et al. 2011) and Arduino (Banzi and Shiloh 2014) projects have revolutionised hardware prototyping by significantly lowering the barrier for entry to both 3D printing and micro-controller technologies. Major research organisations such as CERN have also backed open hardware as a framework to facilitate knowledge exchange. For example, their White Rabbit project (Serrano et al. 2013) now controls the timing synchronisation of many of the world's largest scientific instruments. The White Rabbit project's open-source design model has allowed numerous organisations and companies to develop a thriving ecosystem of inter-operable synchronisation circuitry with sub-nanosecond accuracy, increasing adaptability by avoiding vendor lock-in.

A key difficulty for open hardware projects is how to allow multiple designers to simultaneously contribute to a design, and for all contributors to have unfettered access to all of the product data (Stirling and Bowman 2021). Traditionally, Product Data Management (PDM) systems are used to store these data. However, the centralisation of PDM provides problems for open development as one entity has exclusive control of the data and of access to this data. For open development, a system is needed which both gives each contributor full control to modify their copy of design, control over which external modifications are merged into their copy of the design, and robust data storage that ensures that each designer's contributions are adequately recorded. This produces a rich design history, which is essential for issues ranging from design compliance to ownership and licensing agreements.

As open hardware is beginning to mature (Wilson Center 2020), so is the legal and regulatory framework that supports it. In 2020, CERN released version 2.0 of its Open Hardware License (CERN 2021a), and DIN released a standardised definition of Open-Source Hardware (DIN 2020). The toolchain and workflow for the open-source hardware development is also beginning to standardise. Distributed version control systems such as Git allow distributed teams to work on their own branch of development rather than relying

CONTACT Julian Stirling j.stirling@bath.ac.uk Department of Physics, University of Bath, Bath, UK

on a traditional PDM. Git-based developer operations (DevOps) platforms such as GitHub and GitLab are seeing an increasing number of hardware projects, and CERN's Open Hardware Repository (CERN 2021b) is also an instance of the open-source GitLab platform. While adopting platforms originally designed for managing software has its drawbacks (Stirling et al. 2020), it also unlocks a number of powerful tools for automating time-consuming processes.

This paper considers DevOps for hardware (henceforth HardOps). HardOps utilises DevOps toolchains to capture and control the design process, and to automate computational tasks necessary for design and manufacturing. DevOps workflows are well established, but this work contributes a novel application of DevOps to hardware. The concept of DevOps for hardware was previously introduced by Keysight Technologies (Keysight 2019). However, Keysight's 'TestOps' was focused on automating simulated testing of circuitry rather than a more general approach to using DevOps tools for hardware development.

The term 'hardware' in this context refers to mechanical devices or components that can be machined or assembled, and also to electronics such as circuit boards, sensors, and cables. In this work field programmable gate arrays (FPGAs), application specific integrated circuits (ASICs), and any other form of custom silicon chips are not considered as hardware. FPGAs and ASICs are 'gateware' sitting somewhere between firmware and hardware. Gateware is generally written in hardware description languages (HDLs) that can be simulated and are compiled to binary bitstreams that configure the FPGAs. As such, software source control and automation can more easily be applied to Gateware. Hardware, as defined above, faces a different set of challenges; HardOps seeks to address these challenges.

This paper explores the workflow developed for the OpenFlexure Project to demonstrate how HardOps can be implemented in practice. Due to the public availability of open-source hardware designs, they are ideal for demonstrating the HardOps workflow. However, the same principles could be used for proprietary hardware design using an internal or private repository. The more general utility of HardOps is considered in Section 4.

A brief description of DevOps for software development is provided for context. The HardOps workflow is presented in six stages: Plan, Design and Document, Prepare and Verify, Distributed Production, Physical Testing, and Feedback.

2. Software DevOps

In software engineering, developers increasingly take on more responsibility for the operational side of deploying and running their code. These operations include running cloud services, packaging code for download, or responding to service tickets. Traditionally, the role of developers and operations managers were separate but now these responsibilities are often merged into 'DevOps' roles. This shift has lead to an increase in automation by software teams to run tests, deploy and to release code continuously. Feedback from testing, bug reporting, etc. is then recorded and used to guide ongoing development. DevOps is an umbrella term used in job titles, but also refers to these practices and the tools that enable them.

DevOps toolchains generally include version control software, project management tools, and tools to automate computational tasks in a reproducible manner. Version control allows the work of multiple software engineers to be combined, and creates a permanent record of the history of the code. Project management tools include issue tracking systems used to assign units of work and monitor their progress. This issue tracking may integrate with more advanced tools for creating project timelines and roadmaps.

Task automation is organised into 'Continuous Integration' (CI) pipelines. CI refers to a workflow where code changes from each developer are merged into the master copy of the source code regularly. To enable this regular merging of code, automated building and testing of the modified codebase is required to ensure that changes do not conflict or cause unexpected behaviour. A CI pipeline is a set of scripts that are run on a server that perform different functions such as code analysis, compiling executable programs, and testing the software functionality. Outputs from one stage of a pipeline can be passed on to following stages. An example pipeline is shown in (Figure 1). Due to the general applicability of CI pipelines to most workflows, the term CI is now often used to refer to these pipelines and the tools that run then, rather than to the workflow itself.

3. HardOps

The Computer Aided Design (CAD) files and other associated design files for physical products are generally controlled within PDM systems. Large

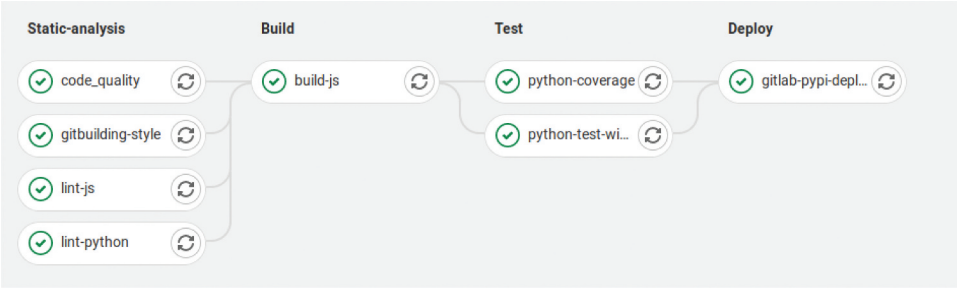


Figure 1. A graphical representation of a CI pipeline produced by the DevOps platform GitLab. The pipeline has 4 stages, each which can have multiple jobs. Here, the code is first analysed statically (i.e. without running the code) for style and quality. If this analysis passes, the code is compiled in the 'Build' stage and tested on both Linux and Windows in the 'Test' stage. Finally, if all tests pass, the code is deployed to a package repository so it can be installed.

enterprises control the entire lifecycle of a product from conception through design, production, and service with Product Lifecycle Management (PLM) systems, of which PDM is just a single aspect. PDM can be considered a form of version control for hardware designs. PLM has considerable synergy with DevOps, as it provides the tools to manage not just the design but the entire workflow. There are numerous differences in the features, technical implementation, and the interfaces between PDM/PLM systems and between Version-Control/DevOps platforms. This is partially due to the differences between hardware and software lifecycle and design, and also due to the interfaces that these communities are familiar with. While full consideration of these aspects is important for efficiency and ease of use, there is one key and fundamental difference between PLM and DevOps: centralisation.

DevOps platforms have built up around distributed version control systems (DVCS). DVCS were largely pioneered by the Free-Libre/Open Source Software (FLOSS) community to allow engineers from across the world to simultaneously work on large, complex projects such as the Linux kernel. Over the past decade DVCS usage has seen a sharp rise (Deepa et al. 2020), becoming the most common form of version control used for both FLOSS and proprietary software development, for both small teams and large enterprises. More details on DVCS are given in Section 3.2. With DVCS it is possible to have continuous, collaborative development at multiple organisations, where each organisation retains full control over what enters *their* copy of the design. In contrast to DVCS, PDM/PLM systems are highly centralised. Moving information between PDM systems at different companies is a challenging task, usually requiring third-party software and rarely preserving full history or metadata.

Figure 2 graphically represents the key difference between HardOps and PLM data management. Each HardOps user has a full copy of all files and all history. Any file can be modified in the user's own copy. Data is shared between users and the central platform, or directly between users. Users synchronise their local storage, checkout and explore changes submitted by other users, and then decide which changes to merge with their own version of the design. For example they may merge an interesting external change with their development branch but not their production branch. In contrast for PLM, all history is stored on the server, users can checkout specific files, locking them for others. Server permissions control which users can submit which changes back to the central storage.

HardOps is a paradigm which unlocks the huge benefit of DVCS for hardware development. HardOps covers more than just the DVCS, but also covers the workflows and management of the design process. Figure 3 describes the HardOps workflow using the DevOps infinity loop. Whilst HardOps is still a new and developing concept, a working framework and associated workflows for distributed design and prototyping are in active use in the OpenFlexure project. This has reached the stage of small-scale production, including some commercial production, in multiple countries. Unlike PLM, HardOps has not yet been used for production, procurement, and service at large scale. Many of the processes described remain project and technology specific and need generalising for a wider audience. The OpenFlexure HardOps infrastructure is the GitLab DevOps

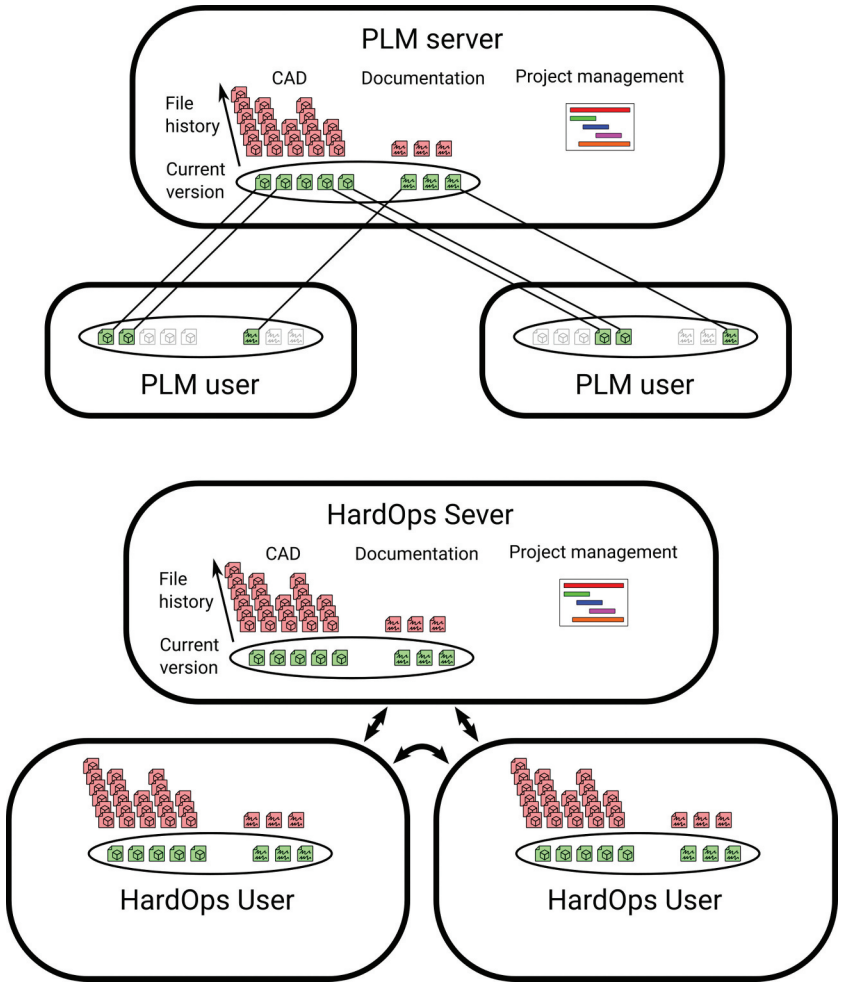


Figure 2. A graphical representation of the key difference between a HardOps platform and a PLM server. The PLM server holds all product data only allowing users to check out specific files. By contrast HardOps users synchronise their own complete copy of the product data, and its history with the platform. Optionally product data can be transferred directly between users.

platform, but other inter-operable platforms such as GitHub, BitBucket, Gitea, etc. provide similar functionality.

While the OpenFlexure implementation of HardOps uses public cloud service, it relies on open source DVCS which allows each party to store all files locally. DCVS also allows information to be shared directly between parties rather than requiring transfer via the cloud. This local storage and control of files of any type differs from fully cloud-based design collaboration platforms such as Fusion 360 or GrabCAD.

The current implementation of HardOps is not fully distributed. Due to limitations of the DevOps platforms it is implemented on, most aspects of the project management cannot be handled in a distributed or federated manner. This is discussed further in the Section 5.

3.1. Plan

The initial planning phase of hardware and software projects follows a similar process. Mapping out requirements, collating early sketches and ideas, dividing work into tasks, assigning tasks to people, and tracking progress. HardOps planning is centred around ticketing systems called 'issue trackers' on DevOps platforms. Each issue in the tracker can represent a body of work to be done, a topic under discussion, a report of an incident or a result that requires attention.

Just as with any ticket-based project management system, the issues form a threaded conversation where files can be attached, and the issue can be assigned to users. Issues can be grouped by tags, filtered by status, and manipulated on interfaces such as Kanban boards. Basic project management

functionality could be provided by any number of external tools. However, issues tracked through a DevOps platform can be linked to proposed changes ('merge requests' or 'pull requests') in the DVCS, allowing the progress of a task to be automatically updated as work continues. Unifying project planning, feedback from manufacturing and quality control, and version control of designs and documents in a single platform, DevOps (and hence HardOps) closes the loop of the development cycle (Figure 3). Planning becomes an ongoing, cyclical process that engages people throughout the team, rather than a top-down exercise for management.

The OpenFlexure project also uses its DVCS to store auditable records of other important documents. This includes detailed planning documents for enhancing the project, and minutes from design review meetings. Keeping these documents in the DVCS means the same automation tools discussed in Section 3.3 can automatically collate, process, and publish documents, and could be used to warn if minutes are incomplete or not uploaded in a timely fashion.

The DVCS repository is the centre of the HardOps workflow, with each repository having its own issue tracker, permissions, automation scripts etc. However, for complex projects it does not always make sense to have a single repository for all files. The OpenFlexure project, for example, has separate repositories for hardware, software, embedded operating system, electronics, project website, failure mode analysis, and more. These repositories are grouped together so issues can be linked, share labels, and be managed in the same milestones. Managing all these aspects of the project with the same tools creates a highly integrated workflow. This allows a small team, such as the academic group that runs the OpenFlexure project, to effectively manage a complex project.

The ability to maintain multiple interlinked DVCS repositories is highly beneficial not only for projects that combine software and hardware design. It is also useful for modular projects where, sub-systems may be developed separately and used in multiple different products.

3.2. Design and document

In the HardOps workflow all design files and documents are version controlled using a DVCS. This work concentrates on Git as the most common DVCS, but note that other systems such as mercurial have similar underlying technology. Git uses SHA-1 cryptographic hashes to uniquely identify files and snapshots of the design (known as commits). Each commit includes a description, an author, a complete snapshot of the files, a timestamp, and the hash of the previous commit. This content is then hashed to create a unique

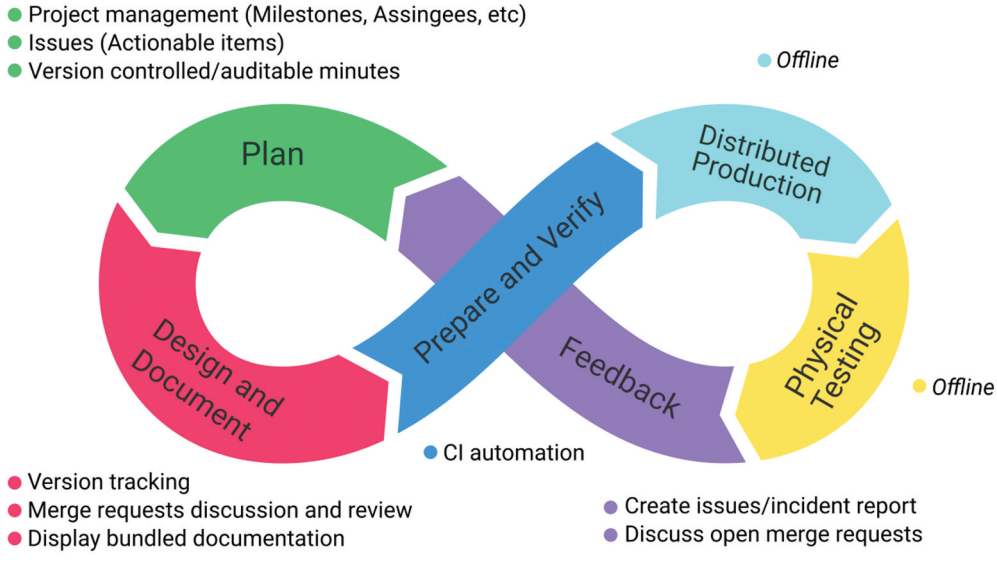


Figure 3. A graphical representation of the HardOps cycle, demonstrating the closed nature of the cycle. Automatically generated production files from designers directly reach production and prototyping teams via the Prepare and Verify stage, and the feedback from production and testing forms the inputs for future plans. This can bridge the gap between the design and production in a large organisation, but can also automate enough tasks to allow a very small team to function with high efficiency and reproducibility.

identifier for the commit. As the hash of each commit depends on the hash of the previous commit, this creates a Merkle tree – the same technology that underpins blockchains. The entire Merkle tree is stored on the machine of any design participant. While changes can be synced directly between participants, they are more commonly shared via a server on the DevOps platform.

The use of a Merkle tree provides a cryptographic record of all changes that includes the time and author of each change. Retroactive changes to any part of the history affect the hashes of all subsequent commits, making the system secure against tampering without the overhead of 'proof of work' used in many cryptocurrencies (Krause and Tolaymat 2018). This tamper resistance can be further improved by authors cryptographically signing individual commits to allow verifying their origin. The system remains secure even when files are stored separately, for size or confidentiality reasons, provided those files are identified within the DVCS by cryptographic hashes. Tools exist to automate only transferring hashes for identified files or file types.

Each designer or design team can then have their 'branch' of this tree where they make their own changes to the design without needing permission from the manager of the central server. These changes can also be synced to the server in their own branch. Information then transfers between teams by merging the changes from one branch into another (See Figure 4). Data can flow two ways between branches, and also between servers. This is essential for creating equitable collaborations as it allows teams at different companies to have full control of what data enters their branch of the design.

Git, originally written to control the source code for the Linux kernel, has been demonstrated to be appropriate for huge collaborative projects with thousands of engineers contributing. However, one key difference between hardware and software is the file types used for both design and documentation. Computer code is plain text, and most code documentation is also written in a plain text markup language such as markdown. Git can automatically scan changes in these files, and record the lines of text or code that have changed. This has two benefits. First, it significantly reduces data storage as only the changes are recorded. Secondly, it allows changes from two branches to be merged as long as the same lines were not changed. In the case where conflicting changes occur, they can be highlighted for review.

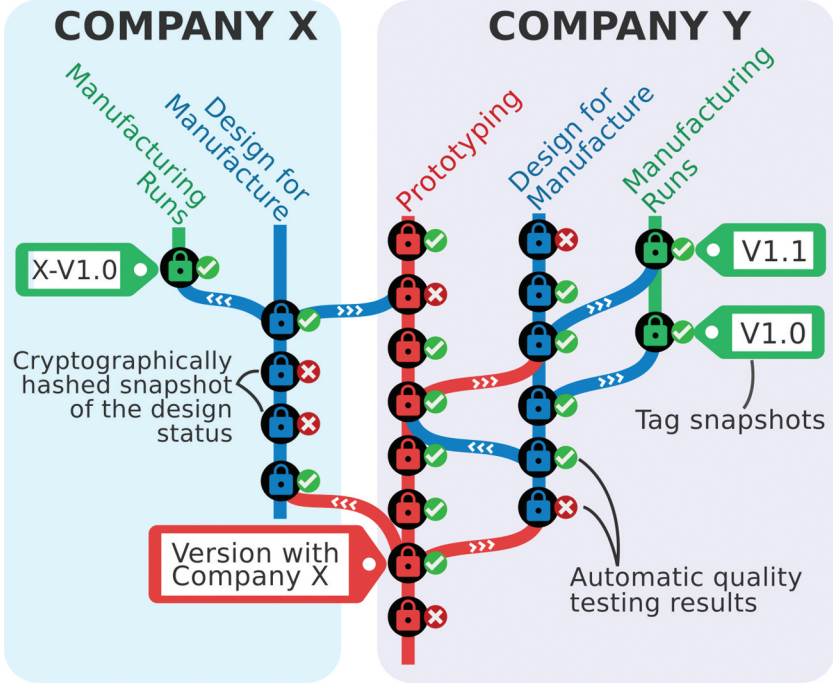


Figure 4. Schematic depiction of the DVCS. Each team can have their own branch of the design, each with their own cryptographically hashed snapshots. Design information can flow both ways between teams and even between different organisation running the DVCS on different servers.

Hardware development documentation could be stored in a markup language rather than binary files from word processors. However, the major obstacle is CAD files as these are often highly complex binary file types. As such, any small changes to CAD files that happen on two separate branches require an engineer to assess the changes in both branches and manually combine the changes. Even text-based CAD formats (such as the KiCad.kicad_pcb file) require considerable manual effort or specialist tools to compare or combine changes. This is due to the complex relationships between the data stored in the file.

The complex nature of CAD files is the largest barrier to adoption of HardOps. However, some mitigations are available. In the case of the OpenFlexure project, all designs are created in the constructive solid geometry language OpenSCAD, and as such the design itself is computer code. This is a powerful workflow that enables highly parametric design, but requires significant programming experience. Another option is to set up centralised storage for specific files, and lock them when they are being edited. Git extensions allow centralised files to be managed using the DVCS, but this loses some benefits of the version control system being distributed. The preferred way to solve this problem is to using external software tools for comparing modified files, or for merging changes. As editable CAD files are not standardised, the onus is on each CAD proprietor to provide this comparison and merge functionality; Git already provides mechanisms to integrate such tools with the DVCS.

In the OpenFlexure HardOps workflow, important branches are protected so that designs cannot be changed without approval. Changes are merged into these branches only through an interface on the DevOps platform. A designer can open a request for changes to be merged. These merge requests have unique IDs, threaded discussion, and can be linked to issues. Alongside the threaded discussion the platform can report the results of automated quality checking (Section 3.3) and require approvals from specific people or roles. This automated workflow ensures clear and consistent scrutiny, documentation, and approval of changes to protected branches.

3.3. Prepare and verify

The most powerful aspect of the HardOps workflow is the customisable automation that can be used to ensure that tasks are performed in a repeatable way. HardOps utilises the CI pipelines that are integrated into DevOps platforms. As the CI pipelines are a series of computer scripts, any task that can be automated via scripting can also be performed as a job in a CI pipeline. In practice, tasks that take a significant amount of time or require significant resources may not be appropriate for a CI pipeline due to the frequency with which they will be run. However, most DevOps platforms provide the ability to run the pipelines on external computing infrastructure, allowing for tasks that are resource intensive or require software licenses.

The OpenFlexure project uses CI pipelines to automate numerous processes. Figure 5 shows a simplified representation of the OpenFlexure Microscope pipelines, where multiple repositories build assets that are essential for creating the microscope. These assets are then published on a public facing website, but could equally be stored on in internal server for a non-open project.

The key benefit of using this automation is de-duplication of design information, having a single source of truth, and a guarantee that essential production files remain up to date. For example, an engineer will design parts using a CAD or electronics design automation (EDA) package, and save them in the native format of their CAD/EDA package. The native format retains essential parameterisation for future adjustment, but before the part can be produced the data needs to be exported either in an exchange format (e.g. STEP, STL), as technical drawings or schematics, or as computer numerical control (CNC) production files (e.g. G-code, Gerber). Saving these output files within the DVCS not only rapidly increases data storage, but also runs the risk of out of date production files being used.

CI pipelines allow just the native CAD/EDA files to be stored in the DVCS. Each time these design files are updated, a pipeline can be triggered with numerous jobs. An example pipeline would be to first ensure that the changes to the input CAD files does not fail quality checks. These quality checks could be running interference detection on assemblies, or running finite element

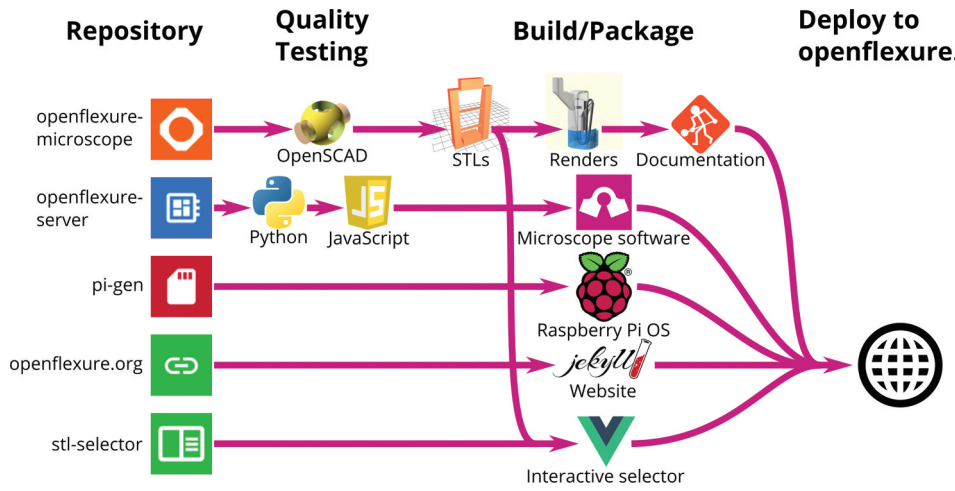


Figure 5. A simplified representation of the repository structure and automation pipelines for the OpenFlexure Microscope. Both hardware and software designs undergo quality checks. If these checks pass, print-ready CAD exchange files (STLs), renders of the assembly process, documentation, software, and other assets such as the website are all automatically generated. If the changes are marked for release, these files are then transferred to https://openflexure.org ensuring that all necessary digital assets are up-to-date and available.

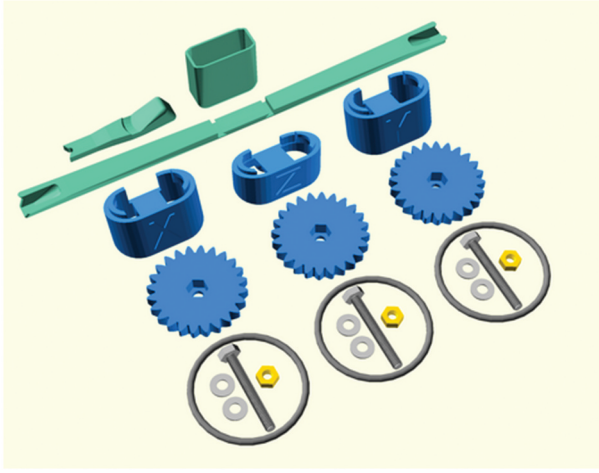
analysis. Once all quality checks pass, the necessary production files can be recreated in a reproducible manner, guaranteeing the same export settings each time.

The OpenFlexure Microscope follows a similar pipeline to above (Figure 5). When any OpenSCAD file is changed, a series of unit tests check that custom OpenSCAD libraries perform all functions as expected, and a custom static code analyser checks for potential code errors within all OpenSCAD files. Once these checks pass, OpenSCAD is run to generate the STL files needed to 3D print the microscope. OpenSCAD is then run to render the illustrations for the assembly instructions, and finally run custom software is run to generate assembly documentation with up-to-date illustrations and links to up-to-date STL files. The documentation step also produces a bill of materials from metadata in the assembly instructions (Figure 6). In future, the OpenFlexure pipeline will generate bills of materials from both CAD and assembly instructions independently to confirm they remain consistent.

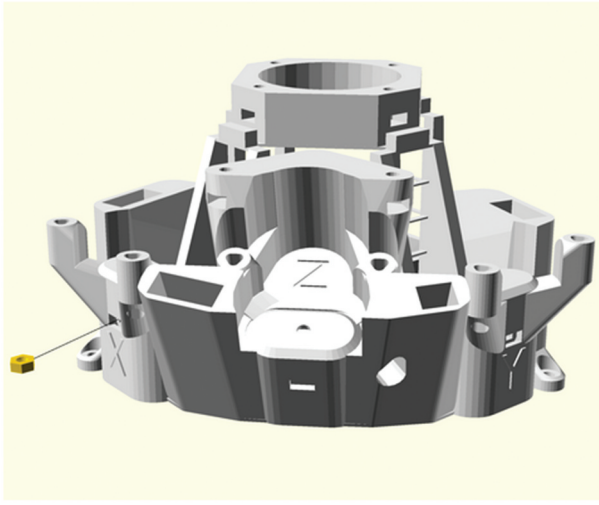
The OpenFlexure Microscope pipeline is enabled by powerful open-source software. This pipeline uses Python, OpenSCAD, Inkscape, ImageMagick and numerous other software libraries all running in Ubuntu images within Docker containers. As these programs are not tied to software license keys, it is possible to run multiple tasks and even multiple pipelines in parallel.

CI automation within the OpenFlexure HardOps workflow is not limited to the mechanical design. A key advantage of the workflow being managed on a DevOps platform is that more traditional CI pipelines can test and build the associated software, and can even build and deploy the project website (Figure 5). This reduces the person-hours involved in ensuring that different elements of a large project that spans disciplines is in sync.

DevOps platforms also provide the ability to run jobs on regular schedules rather than in response to specific actions. This, combined with software 'bots' for interacting with the issue tracker, allows projects to automatically ensure that issues are assigned, regularly updated, and not overdue. Custom scripts could also be implemented to check that quality management processes are ongoing, rather than waiting for the next internal audit. For example, if a project's quality manual states that design reviews happen monthly and are minuted, a scheduled pipeline can ensure that minutes are uploaded at least monthly and contain key information. If this is not happening the same bot could open an issue assigned to the project's Quality Manager. As with all automation, this does not replace the need for dedicated personnel such as the Quality Manager, but helps them work more efficiently and improves consistency.



Step 1: Insert the nut



- Insert a brass nut into the x actuator through the hole in the front of the main body.
- Look through the hole you should see the side of the nut. If it is tilted you can tap the microscope until it sits flat.

Figure 6. A screen shot from the assembly instructions for the OpenFlexure Microscope. The rendered illustrations and the associated STL files for 3D printed parts are automatically generated by the CI whenever the repository is updated. Assembly instructions are then generated using the rendered illustrations and linking to the STL files.

3.4. Distributed production and distributed testing

There is considerable overlap between the challenges hardware and software development within the 'Plan', 'Design and Document', and 'Prepare and Verify' stages of the HardOps workflow. Where hardware development truly diverges from software development is in production and physical testing. These sections of the workflow are discussed together as they face similar challenges. The authors note that the HardOps model has been implemented for prototyping, but a complete workflow for manufacturing has not yet been established.

Hardware production and testing requires control of procurement of components and raw materials, access to machinery and skilled technicians, and the correct tools to perform quality control. These requirements diverge considerably from the DevOps workflow, where software is compiled and deployed in an identical fashion on servers across the planet. HardOps standardises neither the equipment available at distributed locations, nor the supply chain within which they operate. Instead, it provides a consistent method to document procedures and capture information.

Further evolution of the HardOps workflow could implement essential functionality provided by PLM. For example, automatically generated bills of materials could be passed to procurement processes, and applications could be deployed to track manufacturing steps and QA/QC in real time to create a unique record for each device. However, this functionality has not yet been implemented. The current workflow concentrates on ensuring that the design inputs to production and testing are reproducible, and that information can then be collected for feedback to the plan stage.

As mentioned in Section 3.3, HardOps enables automatic creation of up-to-date production files, illustrations, and final documentation. Documentation also should include any QA/QC procedure that are necessary. In the production and testing-phase partners at any location simply follow the instructions to create the part, opening an issue tracking ticket for any problems they encounter. These problems can range from insufficient procurement information and poorly worded documentation, to unmachinable parts or parts failing quality testing.

Simply following a procedure and documenting the result is a key part of the Plan-Do-Check-Act cycle (Abuhav 2011) that underpins international quality control standards. HardOps provides a transparent way for teams that perform the 'Do' and 'Check' procedures to remain part of the overall process, by taking part in the 'Act' and 'Plan' discussions, as described in Section 3.5. Crucially, it enables this cycle even across teams in different locations, organisations, and contexts, where sharing a central PLM platform is impractical.

3.5. Feedback

The issue tracker in a DevOps platform provides an effective way to capture feedback from later stages in the process, including discussion between different people involved in the process to clarify the feedback. As previously mentioned, the DVCS is the lifeblood of the HardOps workflow, and changes to the design can reference issues in the issue tracker. Modifications to a product can be made on a local copy of the design, and proposed for adoption through a 'merge request' in the DevOps platform, referencing the issues that prompted the change. When changes to the design are proposed in a merge request, the prepare and verify pipelines will create the necessary digital assets for production. Production and quality teams can then be assigned to to this request, allowing them to trial production and perform quality tests. These results can be fed back within the threaded discussion for the merge request itself. As a result the changes can be approved, rejected, or further revisions could be requested. Once approved, a user with sufficient privileges can merge this request into a team's protected branch. This not only allows distributed testing before the result is merged, it keeps the auditable record of the discussion linked directly to the changes themselves.

Any production or quality issues noticed after a change has been merged can be opened as an issue in the issue tracker. Similarly, a flexible workflow can be adopted where a change may be accepted before all discussions are resolved, to ensure that conflicting changes do not build up. In this case, issues can be created that link to the merge request or the commit that introduced the change. If needed, the issue could be labeled to be clear that the design cannot reach production until the problem has been addressed.

4. Hard-ops beyond open hardware

The challenges of collaborative development are not unique to open-source hardware. Collaborative development of proprietary hardware is more established, but faces similar challenges due to the centralised nature of the IT infrastructure that stores product data PROSTEP_PLM.

Collaboration on a product must either be performed in one company's PLM system with external users being given guest access, or data needs to be transferred from one company's PLM system into the systems of its partners. If the collaboration is centralised to one company's PLM system this creates a power imbalance where this company has ultimate control of all data. However, if it is instead transferred between PLM systems this involves the complex and time-consuming procedure of exporting data from one system and adapting the data before importing it into another. This can damage the fluidity of collaboration, and runs a risk of data corruption or degradation.

Cloud-based collaboration platforms provide one alternative, but this requires trust in the cloud provider for both data security and data integrity. HardOps provides a true alternative. Cloud-based DevOps platforms exist, but these platforms can also be hosted internally within an organisation. Users on an internal server can still collaborate with users on other servers via the DVCS.

Consider a collaboration between multiple organisations, each with an internal DevOps platform. The DVCS in this platform instantly provides collaboration and greater control. Any files within the DVCS can readily be synced between any organisation. Each organisation, and indeed each worker at the organisation, maintains their own copy of the full design and its history. Each organisation can 'fetch' all modifications from all other servers making them locally available, but can then choose which of these modifications they 'merge' into their working copies of the data. As this synchronisation can be performed over Secure Shell (SSH), all data can remain secure from all but those in the collaboration.

As all parties have a cryptography hashed record of all modifications and their authors, the source of any design changes are traceable. As user access to the repositories can be tightly controlled, any organisation is able to maintain an externally facing repository accessible to all partners, and also a full working repository with sensitive data could also be maintained. Only data to be shared needs to be pushed to the accessible repository.

Whether a collaboration is open-source hardware or a collaboration on a proprietary design, the DCVS provides the ability for multiple distinct versions of the product data to exist and be synchronised selectively. The more distinct the copies become the harder it is to ensure that new work is compatible with all versions. Conflicts where different copies of the same file are changed in incompatible ways can be easily detected but require significant manual work to correct. More complex inconsistencies can arise when edits to separate files cause unexpected behaviour. For example two independent components may be modified, each modification on it's own results in a valid assembly, but if both are modified the parts clash.

The possibility of incompatible modifications causing defects highlights the need for other aspects of the HardOps workflow. The conflict only occurs when one set of modifications are merged into a branch with the other set of modifications. Automation pipelines could be set up to automatically detect certain problemsfor example, in the above example running clash analysis would identify a problem. Also any request to merge files on a DevOps platform can be set to require review and sign off by specific staff members.

5. Future work

HardOps is in use by the OpenFlexure Project, but requires work on a number of areas to generalise it. In particular, integration for CAD packages other than OpenSCAD and the ability to manage records of production are needed.

CAD integration is more complex than simply storing, manipulating, and visualising files. HardOps automation pipelines rely on programs providing a command line interface so their functionality can be accessed by scripting. Such a workflow would require adaptation depending on the tools being used. For proprietary CAD packages licencing may be problematic requiring either sequential execution of jobs or multiple licenses. Also many CAD packages lack effective command-line execution increasing the complexity of creating these pipelines. However, the functionality of most CAD packages can often be automated via software APIs, command-line software could be written to access these APIs. Significant work is needed to create a suite of automated tools to enable the HardOps workflow to be compatible with a wider range of CAD packages. For complex projects custom validation software will likely be required.

DVCS is appropriate for managing the design and documentation of a product, and HardOps builds on this to provide automation and project management. However, the project management (issue tracking, merge requests, etc) are still centrally controlled by a DevOps server. This limits collaboration when groups wish to maintain their own servers, rather than their own repositories on a single server. Without a mechanism for federating the project management, this will hinder collaboration or cause hard to fix inconsistencies between different branches on different servers. For example, merge requests with conflicts may be discussed independently on different servers, and even resolved in different incompatible

ways. This problem cannot be effectively solved without further study of mechanisms for inter-organisational collaboration.

Furthermore HardOps has only been implemented for prototyping and very limited production runs. If HardOps is to be scaled to production or enterprise it must also include data storage for structured data that is not appropriate for the DCVS. For example, quality control and production records that are best handled by dedicated databases, however, the data that defines the quality control processes and in turn these records should sit within the DCVS. Thus, effective links between design data, and database structure will be essential.

6. Conclusion

This work has demonstrated that widely available open DevOps platforms can be utilised for product design. The key difference between the proposed HardOps paradigm and the more traditional PDM/PLM is that decentralisation is core to the model. HardOps can be used by independent designers, or within a single organisation, even if external collaboration is not planned. Having all data stored centrally on company infrastructure within HardOps repositories reduces the barrier to forming collaborations. As open source and free-to-use hosted DevOps platforms exist, this opens up key automation to SMEs and start-ups that are priced out of PDM/PLM. However, cost-saving is a secondary benefit. As HardOps scales to larger more complex projects, costs will arise for dedicated computing infrastructure for automation and the development of company specific integrations.

Despite the decentralised nature of HardOps, it still brings the core benefit of a centralised system – joined up, standardised workflows and efficient flow of information between different teams. However, one key area where HardOps falls down is on the usability of the interface. As HardOps is built upon DevOps platforms, numerous features in the interface are not relevant to hardware, and previews for many hardware files are also lacking. However, the biggest barrier to HardOps adoption is the software used to interact with the DVCS. As DVCS, such as Git, have been designed for software development, the standard tools are complex command-line applications that require programming skills to operate. Even graphical interfaces for these systems have been designed for programmers and are difficult to integrate into an engineering workflow. However, as the underlying technology that powers DevOps platforms is both technically capable of managing hardware design and is open source, what is required is simply an appropriate user interface.

DVCS and DevOps have revolutionised software development, allowing international collaboration to be second nature, and opening state-of-the-art distributed workflows to even the smallest teams. HardOps has the potential to do this for hardware development, but is at a very early stage. Not only do the user interfaces need significant adjustment for hardware developers, but also a toolchain of scripts and interfaces to CAD packages and associated utilities are required. The fragmentation of computer aided design hurts collaborative engineering. Each CAD package having its own data format, its own APIs, its own PDM, and complex licencing agreements make building a collaboration ecosystem very difficult. Software engineers have demonstrated the benefits that open standards and API interoperability can bring to automated workflows. If bolstered by a standardised interface to CAD packages, HardOps has the potential to bring many of the benefits of PLM to both distributed design teams, and to organisations that are too small to invest in costly PLM.

Note

1. The authors are careful to separate the design of medical equipment from a certified product that can be labeled as a ‘medical device’.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the Engineering and Physical Sciences Research Council [EP/R011443/1,EP/R013969/1]; Royal Society [RGF\EA\181034,URF\R1\180153].

ORCID

Julian Stirling http://orcid.org/0000-0002-8270-9237
Richard Bowman http://orcid.org/0000-0002-1531-8199

References

Abuhav, I. 2011. *ISO 13485: A Complete Guide to Quality Management in the Medical Device Industry*. London: CRC Press. 9781439866122.

Baden, T., A. Maia Chagas, G. Gage, T. Marzullo, L. L. Prieto-Godino, and T. Euler. 2015. "Open Labware: 3-D Printing Your Own Lab Equipment." *PLoS Biology* 13 (3): e1002086. doi:10.1371/journal.pbio.1002086.

Banzi, M., and M. Shiloh. 2014. *Getting Started with Arduino: The Open Source Electronics Prototyping Platform*. Sebastopol, CA: Maker Media, Inc. 9781449363291.

CERN. 2021a. "CERN Open Hardware License V2" https://cern-ohl.web.cern.ch

CERN. 2021b. "CERN Open Hardware Repository Openhardwarerepo."

Collins, J., J. Knapper, J. Stirling, J. Mduda, C. Mkindi, V. Mayagaya, G. Anyelwisye, P.T. Nyakyi, V.L. Sanga, D. Carbery, L. White. 2020. "Robotic Microscopy for Everyone: The OpenFlexure Microscope." *Biomedical Optics Express* 11 (5): 2447–2460. https://www.osapublishing.org/boe/abstract.cfm?doi=10.1364/BOE.385729

Deepa, N., B. Prabadevi, L. B. Krithika, and B. Deepa. 2020. "An Analysis on Version Control Systems." In 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), 1–9. IEEE. https://ieeexplore.ieee.org/xpl/conhome/9070069/proceeding

DIN. 2020. "Open Source Hardware - Teil 1: Requirements for Technical Documentation (DIN SPEC 3105-1)" https://www.beuth.de/en/technical-rule/din-spec-3105-1/324805763

Fobel, R., C. Fobel, and A. R. Wheeler. 2013. "DropBot: An Open-source Digital Microfluidic Control System with Precise Control of Electrostatic Driving Force and Instantaneous Drop Velocity Measurement." *Applied Physics Letters* 102 (19): 193513. doi:10.1063/1.4807118.

Jones, R., P. Haufe, E. Sells, P. Iravani, V. Olliver, C. Palmer, and A. Bowyer. 2011. "RepRap – The Replicating Rapid Prototyper." *Robotica* 29 (1): 177–191. doi:10.1017/S026357471000069X.

Keysight. 2019. "TestOps: Blueprint for Connected, Agile Design and Test." https://www.keysight.com/gb/en/assets/7018-06546/white-papers/5992-3771.pdf

Krause, M. J., and T. Tolaymat. 2018. "Quantification of Energy and Carbon Costs for Mining Cryptocurrencies." *Nature Sustainability* 1 (11): 711–718. doi:10.1038/s41893-018-0152-7.

Metcalfe, B., P. Iravani, J. Graham-Harper-Cater, R. Bowman, J. Stirling, and P. Wilson. 2021. "A Cost-Effective Pulse Oximeter Designed in Response to the COVID-19 Pandemic." *Journal of Open Hardware* 5 (1): 1. doi:10.5334/joh.26.

PROSTEP. 2016. "The Challenges Of PLM Collaboration." https://prostep.us/wp-content/uploads/2019/01/Whitepaper_PLM-Collaboration_EN_web.pdf

Serrano, J., M. Lipinski, T. Wlostowski, E. Gousiou, E. van der Bij, M. Cattin, and G. Daniluk. 2013. "The White Rabbit Project." In Proceedings of International Beam Instrumentation Conference, THBL2.

Sharkey, J. P., D. C. W. Foo, A. Kabla, J. J. Baumberg, and R. W. Bowman. 2016. "A One-piece 3D Printed Flexure Translation Stage for Open-source Microscopy." *Review of Scientific Instruments* 87 (2): 025104. doi:10.1063/1.4941068.

Stirling, J., and R. Bowman. 2021. "The COVID-19 Pandemic Highlights the Need for Open Design Not Just Open Hardware." *The Design Journal* 24 (2): 299–314. doi:10.1080/14606925.2020.1859168.

Stirling, J., V. L. Sanga, P. T. Nyakyi, G. A. Mwakajinga, J. T. Collins, K. Bumke, J. Knapper, Q. Meng, S. McDermott, and R. Bowman. 2020. "The OpenFlexure Project: The Technical Challenges of Co-Developing a Microscope in the UK and Tanzania." In 2020 IEEE Global Humanitarian Technology Conference (GHTC), 1–4.

Wilson Center. 2020. "Building Blocks for Better Science: Case Studies in Low- Cost and Open Tools for Science" https://www.wilsoncenter.org/publication/building-blocks-better-science-case-studies-low-cost-and-open-tools-science