13th IEA Heat Pump Conference
April 26-29, 2021 Jeju, Korea

Application of the DevOps methodology on heat pump controllers

Markus Nürenberg[a], Stephan Göbel[a], Philipp Mehrfeld[a], Dirk Müller[a]

[a] RWTH Aachen University, E.ON Energy Research Center, Institute for Energy Efficient Buildings and Indoor Climate, Mathieustrasse 10, 52074 Aachen, mnuerenberg@eonerc.rwth-aachen.de

Abstract

In the work, we show how the DevOps methodology supports the development of heat pump (HP) systems. Modern software development and operation control uses the DevOps method to roll out quickly new software, while keeping an overview of the effects and changes. These features are also applicable for HP controller software. Since rolling out new software to customers is a risky task, the deployment stage of the DevOps cycle runs additional hardware-in-the-loop (HiL) experiments first. If the result is sufficient, the next stage is carried out. Evidently, this requires a high level of automation. On the software side, we use the GitLab CI/CD infrastructure to validate software under static and dynamic measures. Once the HP software is ready to run on a real refrigerant circuit, it is automatically deployed to one. The HP within the HiL setup continues to run with the new software. Since the HiL experiments perform in real time, a continuous observation is necessary to abort the experiment once it fails. Thus, the HiL test bench requires a complex data handling and processing infrastructure.

Selection and/or peer-review under responsibility of the organizers of the 13th IEA Heat Pump Conference 2020.

Keywords: DevOps; heat pump; controller; software development; hardware-in-the-loop

1. Introduction

In this paper, we propose the use of the DevOps methodology to build the next generation of HP controllers. Various work (e.g. [1] and [2]) shows that more sophisticated HP controllers will achieve a major step towards more efficient HP systems. HP manufactures are bound be the obligation that their products shall work for several years, which implies a robust system. Within the software development domain, a uniformly and automatic testing of algorithms is state of the art [3]. Thus, controller software should also be designed and tested according to these standards. However, in the domain of HP software development we often find manually tested software. Especially, when software is applied on machines (like a HP) we have to assume a curtain amount of complexity. This even increases, when we add more components to a HP system and couple the on a data a hydraulic level. Disturbances permute thou the system and the controller work against that. The developer is not aware of all disturbances in advance, therefore, laboratory and field tests have to be conducted. Especially the dynamic behavior of the HP system is still not completely described by simulation. Therefore, HiL experiments are powerful method to gather a useful amount of data from a system under realistic conditions.

2. The DevOps mythology

The DevOps mythology is a relatively new concept within the agile work methods. The name “DevOps” consists of the two words “Development” and “Operations”. Development represents the whole process of factoring software code until the deployment phase. Once software is deployed, operations takes over. Typical

services are technical support, distribution of software, collecting user and technical feedback. In the best-case scenario these information then find a way back to the development department. The DevOps approach now forces these units together into a continuous workflow. However, this usually applies for software produces and services, but for hardware products. Especially for those who have a live span of a decade. The following graphic illustrates the DevOps concept. Starting in blue with the "Plan" phase of the development branch. In the next two phases, software is created and tested. After various successful tests the software is prepared for the deployment. In the "Release" and "Config." phase, the software is rolled out and databases and servers are reconfigured to match new requirements. During the "Monitoring" phase, the operations team gather data regarding the penetration of the update but also the feedback from costumers and their systems. These information lay the basis for further feature or fix development, which closes the cycle.

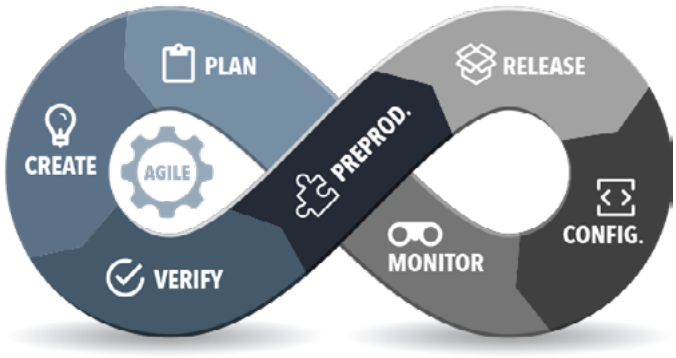

Fig. 1. The DevOps cycle [4]

3. DevOps for HP controller development

We will now propose the application of this DevOps methodology onto the development of a HP controller. Hereby, the controller could just be controlling the refrigerant cycle or the whole HP system, which consists of more other actors and sensors (e.g. pumps, storage temperature, etc.).

The start of our DevOps cycle is again the plan phase. Here, features are pronounced or derived from a requirements engineering process. Zia Babar [5] describes the process in his work. From a market research, customer surveys and user input a product backlog derives. After a grooming of the product backlog this merges into a planning release. However, smaller changes, parameter studies or design of experiments (DoE) studies, such as Mehrfeld [6] used to determine the influences on the seasonal performance of HP systems, can also be set up. If the changes is not based on a simple technical solution but more a holistic request such as "better system performance" or "faster defrost cycle", we will need to translate this into a technical problem first. One of the challenges during this process is the use of different languages, which are brought by the experts of different domains. For example has the word "faster" a different meaning in the field of thermal building processes than on an electrical scope. Therefore, a clear description of the changes and the expected is necessary and can be achieved by creating a specific "definition of done". In addition, the use of issue temples supports this process and ensures that all information, necessary to the task, are available. In fact, there are several platform tools available to support one in this task. We use GitLab as it supports the whole DevOps cycle and is hosted by the university.

After the planning, the coding process starts. Whether it is done in a high-level language such as C/C++ or with an interpreted language such as Python or Java, the source code needs to be available on the platform to run the next steps fully automated. Usually, for each issue a new branch of the project is created in order to separate contributions to different issues. All changes in the software code are committed and pushed to the platform. An ".yml" configuration file (depending on the platform) than leads through the following phases. In case of a parameter study or DoE automated code generation performs this step multiple times. Hereby, we use the CI capability of GitLab to start multiple tests sequences. These sequences are labeled pipeline and can

be tagged with various attributes to route their further way. The following graphic (Fig.2) illustrates the transition from the creating phase to the testing or verifying phase.

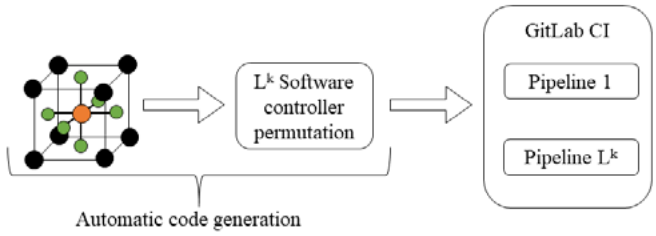

Fig. 2. DoE process as part of DevOps phase. L = Levels, k = number of factors

In this example automatic code generation sets up L^k permutation of the controller. For k being the number factors we want to change and L the number of levels each factor is set to. For a parameter study with no changes to the functionality or structure of the code, the first step of testing would be a simple unit test. When other changes are made, it is wise to run some static code tests first. Figure 3 illustrates a series of test, which can be performed to make sure the controller code matches the company standards towards naming and avoids security issues and common errors. Since errors are recognized before detecting them in expensive hardware tests that even might break the device under test money can be saved.

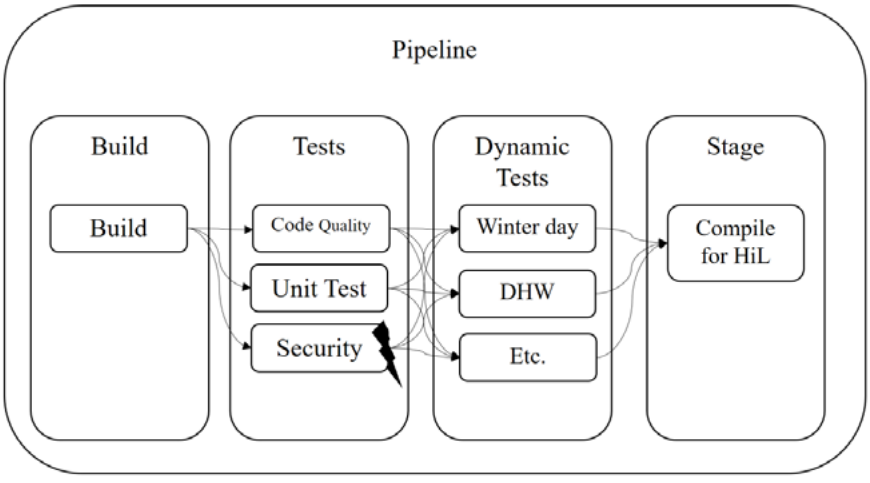

Fig. 3. Pipeline with four stages. Some of the tests are parallel, in order to skip them if necessary. In this case the security test fails but the process continues anyway. After dynamic performance tests a release candidate is compiled to run on the HP controller. The process chain ends with this stage. A HiL experiment toolchain is now able to pull the controller.

These static tests can run in sequel, parallel or as a combination. This depends on the usage of the test result. If the failure of a test represents an abort criteria, there is no need in pursuing any further test. In the example, a security test fails because of a hard coded ip address. This would not be acceptable for production, since the test is tagged accordingly for development, the testing continuous anyway. After the completion of all static tests, the unit testing starts. In a test driven development unit tests are written beforehand. This means, that any code that satisfies the test is consider acceptable. Therefore, the creation of unit tests should be done thoroughly. When focusing on the HP controller the following unit tests should be considered:

- Test for faulty input
    - This tests checks whether the controller reacts correctly to faulty sensor input (e.g. cable break) and if the HP then goes into the correct save or failure mode.

- Test for refrigerant cycle working envelop
  - The refrigerant cycle is bound by the chemical properties (decomposition risk) of the refrigerant and physical properties of the components (burst or vacuum limitations). Here, the temperature and the pressure of the refrigerant are to be handled by the controller within a working envelop.
- Test for correct state switches
  - When working with a state machine the transient conditions allow a step from one to the other state. This test should check if the conditions allow a correct operation or if a deadlock occurred.
- Test for critical comfort conditions
  - This test checks if comfort conditions like domestic hot water and space heating temperatures are met.
- Etc.

Most of these tests can be designed as a rule based sequence. If time constants or timers are involved in the controller function the test framework needs to be able to speed up the internal time, since tests would take too long to finish in real time. This quality is fundamental for any dynamic testing. With these kind of tests we do not only want to determine if a controller works, but how good according to several KPIs.

We use Modelica as modeling language and the ”Functional Mockup Interface” (FMI) [7] to connect the controller to a virtual environment. The FMI standard would allow us to use different modeling approaches as well. The models can vary from simple weather input to tapping profiles to building simulation with the duration from hours to months. If possible, a time clustering should be applied to shorten the simulation time, as Schütz et al [8] described.

If all tests are positive, the software candidate is passed to the next phase. This one is called preproduction and marks, in our case, the link between the virtual testing and hardware testing. The platform builds the controller software for the target HP platform. Thus, all libraries and additional software must be present or the process takes a detour to a manufacture’s building process. This might be necessary when third party intellectual property is involved. Figure 4 illustrates the building phase with known e.g. open source software and company owned closed source software.

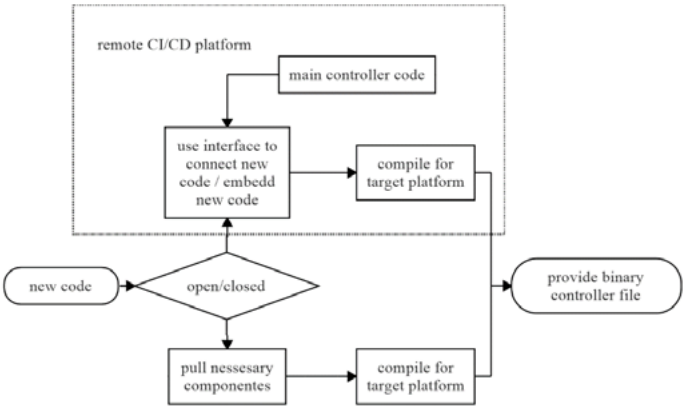

Fig. 4. Process of new code being added to the main controller code. Lower path for known e.g. open code. All necessary components such as libraries or the main code are pulled from registers and compiled according to the target systems’ specifications. The upper path couples a remote CI/CD platform or runner to the process. The main code or any library is unknown. The new code is either embedded into a function or uses an interface to connect to the main code. The code is then compiled by the remote system and a binary file of the controller is provided.

At this point we take the testing to our laboratory, which is equipped with a hardware-in-the-loop infrastructure. There, a HP system is set up according to the release plan. This marks the next phase of the DevOps cycle. This phase determines who will when get which update. When distributing updates to multiple

devices, it might be helpful to do so in waves. Hereby, occurring issues will not affect all devices at one time. Since we are focusing on one HP the HiL infrastructure will run one test at a time. The following figure 5 illustrates the digital setup of the HiL infrastructure. The config phase of the DevOps cycle stets up the production environment, in our case the HiL experiment. A service runner configures all software entities according to a predefined test plan.

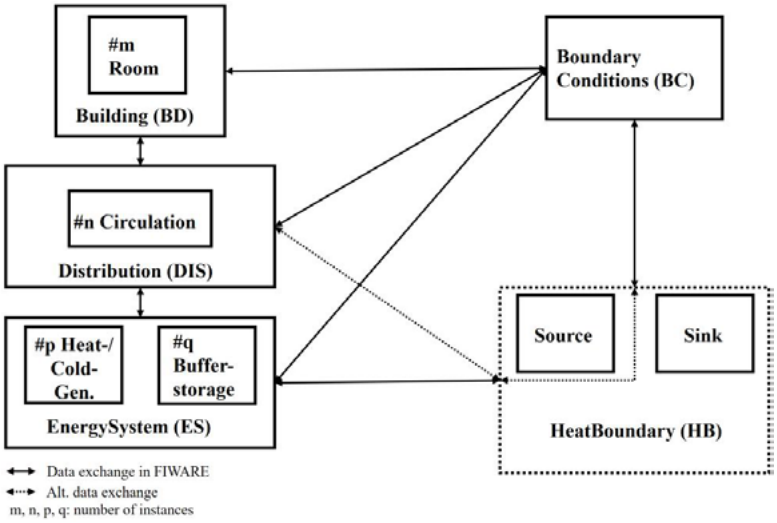

Fig. 5. HiL digital infrastructure. Each component is capsulated and is represented by hardware measurements or a simulation. The entities communicate via MQTT [9] within the FIWARE [10] platform. The installed HP is operated in a climate chamber (HB Source) and connected to a hydraulic test bench (DIS). The digital image of the HP is located in one ES instance. The building simulation BD operates as virtual sink for the HP system. Where the BC instance provides further conditions such as weather or user occupancy.

After the experiment is set up, the compiled controller is flashed onto the HP. The HP is then configured via automated procedure or manually (e.g. heating curve, etc.). From there on the monitoring phase begins. By gathering various data the experiment is evaluated in real time. Watchdog entities hook themselves onto the data streams and calculate various KPIs. In addition, they monitor the test bench behavior. Is either the test bench out of its limitations or a KPI is outside its acceptance, the experiment can be aborted early. The data is stored in a time series database and available for debugging. From there a decision can be made is the experiment and therefore the controller function or parameter set is an improvement and worth to be pursued. From here the DevOps cycle starts again with the planning phase.

4. Conclusion

In this paper we shine a spotlight on the possibilities of the DevOps for HP controller development. We start off with the general introduction of the DevOps methodology in the software development domain. Followed by a use case in the field of HP development. Hereby, we suggest an automated code generation for parameter studies, with creates a full factorial set of controllers, which have to be evaluated. Since the controller software is hosted on a CI/CD platform the following phases of the DevOps cycle are driven by the platform. After a series of tests and evaluations of the code, its performance or even feasibility is tested by dynamic simulations. Candidates, which outperform predefined KPIs are staged for a HiL infrastructure to be tested. A semi-automated process sets up the HiL experiment, while an extensive monitoring evaluates the experiment in real time. This data can then be used to measure the success of the implementation or be the basis for further work.

References

[1] J. A. Candanedo, V. R. Dehkordi and J. Tamasauskas, "IMPACT OF HEAT PUMPS AND PREDICTIVE CONTROL ON RESIDENTIAL ENERGY USE, LOAD AND GRID INTERACTION: A CANADIAN PERSPECTIVE," Hyderabad, India, 2015.

[2] D. Fischer, J. Bernhardt, H. Madani and C. Wittwe, "Comparison of control approaches for variable speed air source heat pumps considering time variable electricity prices and PV," 2017.

[3] I. J. 1. 7. I. J. 1. 7. ISO/IEC JTC 1/SC 7 Software and Systems Engineering, *Software and systems engineering - Software testing,* International Organization for Standardization, 2018.

[4] Perforce Software, "https://www.perforce.com/," Perforce Software , 14 11 2019. [Online]. Available: https://www.perforce.com/solutions/devops.

[5] B. Z., L. A. and Y. E., Modeling DevOps Deployment Choices Using Process Architecture Design Dimensions, Springer, 2015.

[6] P. Mehrfeld, M. Nürenberg, K. Huchtemann and D. Müller, "Influences on the Seasonal Performance of Heat Pump Systems Investigated Via Dynamic Simulations," Portorož, Slovenia, 2016.

[7] Modelica Association, "fmi-standard," 14 11 2019. [Online]. Available: https://fmi-standard.org/.

[8] T. S. (CA), M. H. Schraven, H. Harb, M. Fuchs and D. Müller, "Clustering algorithms for the selection of typical demand days for the optimal design of building energy systems," PORTOROŽ, SLOVENIA, 2016.

[9] OASIS Open, "OASIS Open MQTT Version 5.0," 14 11 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[10] FIWARE Foundation, "FIWARE," 14 11 2019. [Online]. Available: https://www.fiware.org/.