16th IMEKO TC10 Conference
"Testing, Diagnostics & Inspection as a comprehensive value chain for Quality & Safety
Berlin, Germany, on September 3-4, 2019

Application of modern software engineering principles for continuous quality (CQ) management in research

Sascha Eichstädt[1], Bram van der Waaij[2], Björn Ludwig[3], Erik Langius[4]

[1]Physikalisch-Technische Bundesanstalt, Braunschweig and Berlin, Germany, sascha.eichstaedt@ptb.de, +49 30 3481 2008
[2]TNO, Groningen, Netherlands, bram.vanderwaaji@tno.nl
[3]Physikalisch-Technische Bundesanstalt, Braunschweig and Berlin, Germany, bjoern.ludwig@ptb.de, +49 30 3481 7625
[4]TNO, Groningen, Netherlands, erik.langius@tno.nl

***Abstract* – Transparency of changes, automated testing of new developments against documented criteria, combination of developments from different sources and automated publication of validated results are key aspects of modern collaborative software engineering. Various infrastructures, guidelines and tools are available for continuous integration (CI) and continuous deployment (CD) of software. The same principles could be used to design infrastructures for continuous quality (CQ) management in research and the implementation of the Open Science paradigm. In this work we provide a vision of how such a CQ workflow could look like to make scientific research traceable and reproducible.**

Keywords – software development, automated quality assurance, continuous integration, continuous deployment, digitalisation, Open Science

I. INTRODUCTION

Results of scientific research need to be reproducible in order to make actual use of their findings and developments. This simple rule holds for all areas of science – from psychology to materials science and metrology. Reproducibility of research requires a clear traceability from the result reported in a publication to the raw data from measurements or other means. Quality management in research thus should ensure reproducibility by providing measures for the verification (error free) and validation (fit for purpose) of the steps that lead from the raw data to the findings reported. Therefore, most scientific institutes have documented quality guidelines [10] and are using quality management processes for their research. However, these are typically paper-based documentations of principles and infrequent tests. Often inadequate quality management in research only becomes visible, when other scientists question specific results or report flaws. These days, scientific work is usually mostly digital, and a wide range of tools for each step of the research process is available. Hence, traceable and reproducible digital research results should be feasible with available building blocks.

Access to research results includes access to the steps, software and measures that led to the results reported. This is part of the Open Science paradigm [1]. For the open research data, the FAIR principles require the data published to be findable, accessible, interoperable and reusable [2]. In order to realise these principles, processes and infrastructures need to be adapted. To this end, several scientific groups have started developing guidelines and infrastructures to support researchers in the management of research data in accordance with the FAIR principles. It can thus be expected that research data management itself will become more reliable in the near future.

In order to achieve the same for the whole research process, further measures are required. At the same time though, it must be ensured that scientists can focus on what matters most: research. To this end, principles and infrastructures from modern software engineering practice can be employed as template for a mostly automated continuous quality management in research. This enables CQ throught the whole process, not only at the end. CQ is an integrated part of the development.

II. MODERN SOFTWARE ENGINEERING

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software [3]. Development teams usually work asynchronously and geographically distributed and to this end employ tools and workflows assisting the special needs arising from that, like dealing with version conflicts for instance. The starting point for every professional software engineering activity is to ensure the utilization of proper source code management (SCM) enabling all subsequent activities. The main purpose of SCM is tracking file changes linearly over time and non-linearly over different features and bug-fixes kept in so-called "branches". The inverse action to branching is the so-called "merging" and requires modern SCM to provide tools visualizing the differences in diverging versions of the same files and to simplify the process. All wide spread SCM nowadays tackle these issues very successfully by offering well integrated diff- and merge-tools. In conjunction with detailed tracking information about who and when applied what changes and for what purpose those diff-tools allow for transparency about the whole development process.

Regarding workflows the most crucial practice modern teams follow is continuous integration (CI) which lays the foundation for asynchronous and distributed engineering [4]. It is characterized by sharing the output of every team member early and often. Frequently integrating written code into the joint code base fosters a corporate understanding of the current project's state which in turn reduces the number of misunderstandings between developers, discrepancies between developments and therefore minimizes the need and the required effort to resolve conflicts. The early integration of code into the complete process is in contrast to the traditional approach of fine-tuning every detail of a certain module or code snippet before making it available to the collaborators. Hence, this is not only a technology or code development technique but requires a cultural change as well.

Another common practices in modern software engineering is permanent testing and perceiving test writing as being part of the process. These tests range from so called "unit tests" addressing the purpose and boundary conditions of small portions of source code. Up to tests which address the functionality of the final product and its deployment into the intended operating environment altogether, often referred to as the test suite. Adding tests from early stages on and testing new features against integrability ensures quality and robustness of the produced code.

To this end, automated tools are utilized which perform these tests and keep track of the project's current state. Such testing toolchain consists of elements incorporated locally into the team members' IDEs, centralized SCM components and distributed webservices to continuously aid every step from before integrating the code into the code base to final production deployment for why they are commonly referred to as continuous integration / continuous deployment (CI/CD) pipelines. Those pipelines naturally place some demands on the environment in which they are established like quasi-permanent physical and logical access to the SCM holding the code including the test suite and interconnections between the elements themselves. The latter enforces the automation aspect, as pipeline elements can communicate with each other, e.g. trigger the next element in the toolchain or abort further build stages in case of error and inform the developer.

The goal of these efforts is to produce reliable software avoiding all undocumented and unreproducible interventions thus enabling quality assurance and management during and after the development. This holds true especially for open-source software where not only the product itself is delivered but access to the SCM and CI/CD pipeline's outputs is granted in whole or in part to everybody, with access to the final code version being the minimum requirement.

III. COMPLEX RISK CALCULATIONS USE CASE

In the northern part of the Netherlands one of world's largest gas fields is operated. This field causes induced seismicity that is linked to the gas subtraction and damages the build environment, e.g. houses. To quantify the public risk that is associated with the gas production yearly risk estimations (Hazard and Risk Assessment) are being made. These calculations are made up of a complex chain of models which are constantly being improved and modified according to the latest insights.

As policy is based on these type of calculations, Quality Assurance and traceability is of essential importance. Can we reproduce the calculations of two years ago with the software – and data versions that have been used at that time?

These calculations can be characterized as data intense complex chain of models. On a high abstraction level, the model chain consists of three main components: Seismic Source Module (SSM),

Ground Motion Module (GMM) and Damage Module (DM).

The workflow of executing and developing components of the model chain is divided in three steps. A new analysis question starts with the *development phase*, which is a playground for new features. No versioning or traceability is kept for a long period. Soon the process moves to a *staging phase* working dedicated on specific functionality. Continuous testing becomes an integral part containing unit tests for error-free code and model verification and integration tests for validation of (parts of) the entire final pipeline of models. Finally the last phase is *production phase*, using only verified models and pipelines. This environment is kept persistent for a long time, containing all used data, configurations and model software. Because the whole process is automated these cycles can be very sort in time, allowing iterative development because of quick feedback.

In the staging phase the software runs on an exact mirror of computer infrastructure that is used in de production step. When modules have passed the staging phase, modules are built to small self-executable Docker-images. This results in a package that contains the operation environment (e.g. Linux), depending software libraries and the written software itself. This package is identifiable with an unique id by committing is to a CI/CD environment (GIT).
The staging- and production phase both need support for full automation of testing, running and traceability. For this the tool Pachyderm [11] is selected which can manage analysis pipeline runs, provides full traceability of each run and can reproduce previous (old) runs.
With Pachyderm a pipeline of models can be created by putting each model in a separate docker image and connects them through data sets. See figure 1. Each data set has its own full version history, including the pipeline specifications. Running a specific pipeline results in the storage of not only the end result data set, but also all intermediate and external data sets, the originating data sets and the configuration of all models. When a change is made to any of these, for instance a parameter in the configuration of a model is updated, automatically the pipeline is rerun. But only those parts which are effected by the parameter update. For instance when the parameters of the GMM model is updated, the SSM model will not be recalculated. Only the GMM and DM model resulting in an update of those corresponding data set.

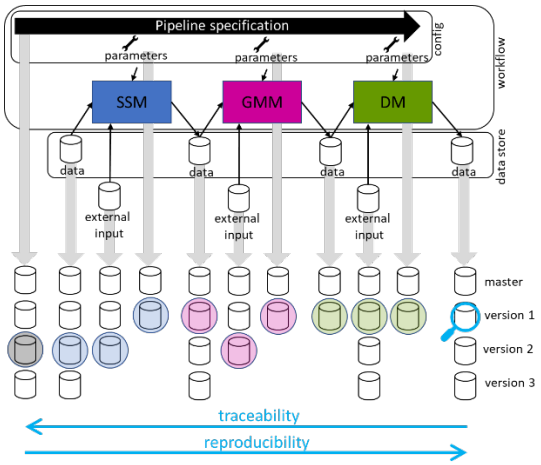

Figure 1: Pipeline versioning and traceability.

For each run full traceability information is generated (indicated by the blue arrow from right to left at the bottom of figure 1) linking the result data set to the models, their configuration, the intermediate data sets and finally back to all originating data sets.
When a question or complaint is raised, a result can be reproduced (indicated by the blue arrow from left to right at the bottom of figure 1) by rerunning the exact old run specification which was used to produce the result originally. That is precisely the traceability information generated with the original run. Therefore this must be stored as long as questions and complaints are expected.

IV. TRANSPARENCY AND OPEN SCIENCE

The shift to Open Science means sharing research results by making them available in a way that they can be accessed and utilized by the whole society [1]. However, this does not mean to provide a scientific paper as Open Access, because then only the financial barrier to access the publication itself would be removed. Access to the research itself also requires access to the relevant data and to the methods (software and configurations) that were employed to gain the results reported. This is also one important step towards reproducibility in science, because other scientists can easily verify and reproduce the steps that led to the conclusions. An excellent example for this concept is the LIGO "Gravitational Wave Open Science Centre" website. It offers the scientific publications and relevant research data as well as documented source code that was used to analyse the data [5]. In this way, reproducibility of the results and further development of the methods by the scientific community is greatly improved.
The prerequisite for a transparently documented

research process is a standardised set of minimum requirements, and the availability of appropriate research infrastructures. For research data this concept is given by the FAIR principles:

- Findable data requires persistent identifiers for the data sets
- Accessible data requires that the data can be downloaded in some way
- Interoperability requires using standardised metadata and annotation of the data
- Reusability requires that the permissions, i.e. license, for the data allows others to use the data

It is important to note, though, that the FAIR principles do not require the data to be publicly available for free as Open Access. That is, implementing a FAIR data management does not mean to make all data available for free. Instead, it means that research processes that generated the data need to be documented and structured in a way that supports the *I* of FAIR.

The *F* and *A* parts require a searchable data base for research data as well as persistent identifiers to individual data sets. Therefore, a couple of development projects have been initiated to provide infrastructures for FAIR research data management. For instance, the project CONQUAIRE aims at an infrastructure that uses git version control for research data [6]. In this way, changes to data sets are transparently documented and earlier versions of the data can be accessed easily. This provides advantages for the scientists performing the research as well as for others. The *TIB Datamanager* [7] uses a similar approach, but also provides basic visualisation tools for data sets. These kinds of infrastructures aim at providing a simple toolset for scientists to more or less automatically document, store and provide their research data.

In the same way, modern electronic lab journals provide an infrastructure for a transparently documented research process. For instance, software like *SciNote* [8] or *labfolder* [9] provide a server-based system that can be accessed via any web browser to create and edit entries. Each editing step automatically generates a new version of the entry, creating a transparent process of the research process.

V. CONTINUOUS QUALITY IN RESEARCH

Quality in research means reproducibility, reliability and transparency from the relevant data to the published conclusions and results. By the researcher during the process of doing his research as well as by other people (long) after publication of the results. In the following we discuss the meaning of CI and CD for research workflows consisting of (i) development of an idea/concept, (ii) acquisition of data, (iii) analysis of data and (iv) reporting the results as a scientific publication, presentation or other means. Based on this we then propose a concept for a continuous quality (CQ) for research.

Phase 1: Development of a concept or hypothesis

In general, a research process starts with a distinct problem, issue, question or challenge to address. Usually, this is the part of the research workflow that is the least documented. However, a good documentation and also publication of this step could be of benefit in several regards: supporting later re-iteration in case the results do not match the expectation; involvement of and feedback from other experts in the field; prevention of fraud. For instance, several scientific journal papers are now requesting their authors to pre-register their hypothesis before conducting the actual study. This is intended to prevent fraud by matching the hypothesis to the actual findings afterwards. At the same time some journals are encouraging authors and reviewers to also consider negative research results for publication, e.g. the journal F1000Research.

Technology-wise, this step can be supported, for instance, by using electronic lab notebooks (ELN) from the start of the research workflow. Therefore, the ELN needs to be able to include plain text, handwritten notes as well as documents with annotations. Furthermore, the ELN should be able to track changes in these elements. Thereby, later changes to the hypothesis, problem description, etc. are made traceable automatically. For research areas where an ELN may not be suitable, digital text-based documents can be used in conjunction with an automated versioning system like git or subversion.

Continuous integration (CI) for this initial step in the research workflow can consist, for instance, of using the ELN collaboratively with other researchers. In a sense, this documents what is common practice in science anyway: joint development of ideas, hypothesis and concepts.

Continuous deployment (CD) then means that this documentation is automatically made available to others, for instance, on a pre-print server.

Phase 2: Data acquisition

The step of data acquisition in terms of measurements, surveys or other means is typically digital as data is being captured in digital formats from the start. Thus, versioning systems can be directly applied. Ideally, this should happen

automatically by implementation of a corresponding data acquisition pipeline. Systems like CONQUAIRE, Datamanager and others are offering technological solutions for this. The researcher itself, though, has to make sure that the raw data streams are actually fed into these systems in an automated way.

With the data being available on an accessible platform and under version control, automated pipelines for data quality checks can be implemented. These quality checks can be very basic, such as, testing for NaN, Inf or other entries that indicate errors in the data acquisition process. Furthermore, tests against certain assumptions can be implemented easily. For instance, spectral measurements can be checked for non-negativity. More sophisticated quality checks may contain plausibility tests against reference data or automated data curation methods. The data quality system could then automatically inform the researchers about issues and annotate the metadata with details about its findings.

Not all issues detected need to result in withdrawal of the data set. For instance, a combination of the metadata about the issues with the subsequent analysis results could lead to unexpected findings and open up new research questions. In addition to the acquired data itself, the software for the data acquisition needs to be put under version control. Changes in the data can then also be traced to changes in the data acquisition process. Quality assurance of this software can be carried out following the principles described above. The version control and quality assurance of data acquisition software extends to a documentation of the data acquisition parameters. When the measurement setup is also software-based or other means for the automated access to the acquisition parameters are available, this part of the research workflow quality system can be automated as well. For example, measurements with an instrument with a programming interface (API) can be acquired together with metadata containing the information about the measuring device's parameters being used. Continuous integration (CI) at this step of the workflow could mean the combination of individual data sets within a larger system. For instance, in a particle accelerator many individual measurements are made and need later to be combined to form an analysis. As another example consider the sensor network example from above. The CI pipeline could, for instance, consider only measurements that satisfy the quality checks to be included in the final data set made available for the next step in the workflow.

The CD part can store the data sets to be used in analysis, the next phase. As before also publishing the data sets to a pre-print server or other place for reporting the research.

Phase 3: Data analysis

The workflow step (iii) data analysis is typically carried out in software, either in terms of self-written source code or using available software tools. For the case of self-written source code, the same versioning system approach as for the data acquisition part can be used. For quality assurance purposes the software testing principles described above can then be applied. Regarding CI principles, an interplay with the previous step in the workflow could be considered. For instance, the software for data analysis may require certain properties in the data, or the pipeline of software modules is depending on specific data properties. Moreover, continuous integration for this step could be realised by means of applying each analysis version, committed to an analysis repository, to the data sets after quality-checking the software itself. Then, the CI pipeline would automatically provide an updated analysis result for the latest software version. This, of course, is more challenging when commercial or other closed-source software tools are applied, only black box testing remains in those cases.

Phase 4: Reporting the findings

The last step in the here considered research workflow is the reporting of the research findings. This is also typically carried out in a digital format that can be put under automated version control easily. Quality assurance in this step, however, is usually limited to grammar, orthography and other basic errors. With more sophisticated machine learning tools, though, complex semantic checks are possible, too. For instance, a paper draft could for a medical survey could be checked whether it contains a well-defined hypothesis, information about the method used and sufficient details about the study.

Continuous integration for this step in the workflow can be carried out, for instance, for the case of LaTeX documents. The LaTeX source code can be put under version control and automated compiling into a PDF document can be carried out on the server. This also allows continuously integrating contributions from different co-authors. Moreover, for the automatically published data sets and data analysis code an automated citation in the report can be integrated.

Continuous deployment in the reporting step can

consist of automatically submitting the compiled document to a pre-print server. This publication could then contain also references to the exact analysis workflow trace, references to all analysis software versions, raw input data sets and configurations. With the automated and continuous quality assurance in every step, the authors can have reasonable confidence in the validity of the report being made publicly available. For intermediate work, this publication could provide automatically generated indications to what quality checks may have failed or what parts are still missing. This is common practice in open source software repositories, and it allows the users of the software to assess the maturity of the published work easily.

These steps all together can result in the following example research workflow. Assume a server-based electronic lab journal is used that allows access to its entries via document APIs. An external software module may then be configured to automatically check specified lab journal parts for consistency w.r.t. units, uncertainties as well as for completeness and other quality measures. This corresponds to unit tests in software engineering. If then all checks for that lab journal are positive, a data analysis algorithm is applied to produce a specific result for that lab journal data. Its output could again be verified by another software module against some reference or pre-specified criteria. If all checks are positive, an entry is created for the output in a research data infrastructure with metadata pointing to the persistent identifiers of the journal entry and the software version being used. Furthermore, a pre-specified documentation of the research, data analysis and findings could be created similar to automated software documentation based on comments in the code.

VI. CONCLUSIONS

Continuous quality (CQ) for research means to develop and implement automated checks for a set of measurable quality factors throughout the research workflow and lifecycle of the analysis. Similar to CI/CD pipelines in software development, automated processes can then be defined to implement FAIR principles for research data or specific aspects of the Open Science paradigm.

Continuous deployment based on CQ could strongly support the Open Science paradigm. For instance, a public dashboard for research project could summarize the CI/CD pipeline outcomes of every step in the above described research workflow. Starting from the original idea, developed collaboratively; to the acquired data; to the data analysis source code and the reporting draft: all elements could be made available in a transparent way and automatically created quality assessments.

The full implementation of CQ is – right now – more a vision than an actual project. However, the principles from software engineering are well established, several infrastructures for research data are being developed and more and more scientific communities are defining quality measures for their work. With CQ pipelines, the administrative overhead for scientists to satisfy FAIR principles and the Open Science paradigm can be minimised. At the same time, direct advantages for the scientists in their work can be realised.

VII. ACKNOWLEDGMENTS

Part of this work has been developed within the Joint Research project 17IND12 Met4FoF of the European Metrology Programme for Innovation and Research (EMPIR). The EMPIR is jointly funded by the EMPIR participating countries within EURAMET and the European Union.

REFERENCES

[1] **Vicente-Saez, R., & Martinez-Fuentes, C.** (2018). Open Science now: A systematic literature review for an integrated definition. Journal of Business Research, 88, 428-436. DOI: 10.1016/j.jbusres.2017.12.043

[2] **Wilkinson et al. "**The FAIR Guiding Principles for scientific data management and stewardship", Nature Scientific Data, vol. 3, 160018 (2016). DOI: 10.1038/sdata.2016.18

[3] IEEE 24765-2017 - ISO/IEC/IEEE International Standard - Systems and software engineering—Vocabulary (https://www.iso.org/obp/ui/#iso:std:71952:en)

[4] **Linping Chen** "Continuous delivery: overcoming adoption challenges", Journal of Systems and Software, vol. 128 (2017). DOI: 10.1016/j.jss.2017.02.013

[5] https://www.gw-openscience.org/start/ Acc. 2019-03-14.

[6] **CONQUAIRE "**Expanding the Research Data Management Service Portfolio at Bielefeld University According to the Three-pillar Principle Towards Data FAIRness" DOI: 10.5334/dsj-2019-006

[7] **TIBD** https://projects.tib.eu/datamanager/ Accessed 2019-03-14.

[8] https://scinote.net – Accessed 2019-03-14

[9] https://www.labfolder.com – Accessed 2019-03-14

[10] The **AQUA Book**: guidance on producing quality analysis for governement. HM Treasury. March 2015, isbn 978-1-910337-67-7

[11] https://www.pachyderm.io - Accessed 2019-6-19