Software Quality Assurance During Implementation: Results of a Survey in Software Houses from Germany, Austria and Switzerland

Michael Felderer(✉) and Florian Auer

Institute of Computer Science, University of Innsbruck, Innsbruck, Austria
{michael.felderer,florian.auer}@uibk.ac.at

Abstract. *Context:* Quality assurance performed during the implementation phase, e.g., by coding guidelines, static analysis or unit testing, is of high importance to ensure quality of software, but there is a lack of common knowledge and best practices on it. *Objective:* The goal of this paper is to investigate the state-of-practice of quality assurance during the implementation phase in software houses. *Method:* For this purpose, we conducted a survey in Germany, Austria, and Switzerland where 57 software houses participated. The questionnaire comprised questions regarding techniques, tools, and effort for software quality assurance during implementation as well as the perceived quality after implementation. The results were complemented by interviews and results from other surveys on software quality in general. *Results:* Results from the survey show that the most common software quality assurance techniques used during implementation are unit testing, code reviews and coding guidelines. Most tool support is used in the areas of bug tracking, version control and project management. Due to relationships between the used tool types, it seems that the introduction of one tool leads to the adoption of several others. Also quality assurance techniques and tools are correlated. Bug fixing takes a significant ratio of the overall project effort assigned to implementation. Furthermore, we found that the more developers a software company has, the more effort is spent on bug fixing. Finally, more than half of all companies rated the quality after implementation as rather good to good. *Conclusion:* For the most important quality assurance techniques and supporting tool types clear usage patterns can be seen and serve as a basis to provide guidelines on their application in practice.

Keywords: Software quality assurance · Implementation · Software development · Software quality · Software houses · Survey

1 Introduction

Quality assurance performed during the implementation phase, e.g., by coding guidelines, static analysis or unit testing, is critical to create software of high quality [1]. According to the ISO/IEC/IEEE Standard 24765 implementation or

D. Winkler et al. (Eds.): SWQD 2017, LNBIP 269, pp. 87–102, 2017.
DOI: 10.1007/978-3-319-49421-0_7

coding is defined as "the process of translating a design into hardware components, software components, or both" [2]. The respective phase in the software development life cycle, i.e., the implementation phase, is defined as the "period of time in the software life cycle during which a software product is created from design documentation and debugged" [2], which also comprises bug fixing and therefore quality assurance activities. Nevertheless, quality assurance during implementation, which we also call integrated quality assurance, lacks a common body of knowledge and is often handled as a "black box" in the overall development process individually managed by developers. Given that a considerable large amount of the total project effort is spent on implementation [3], it is important to investigate the state-of-practice and to provide guidelines for respective quality assurance techniques and tools. This holds especially for software houses, i.e., companies whose primary products are software [4], for which quality assurance is essential to guarantee quality of their delivered software products.

The goal of this paper is to investigate the state-of-practice of quality assurance during the implementation phase in software houses. For this purpose, we present results of a survey conducted by the University of Innsbruck together with the Austrian consultancy company Software Quality Lab in software houses from Germany (D), Austria (A) and Switzerland (CH), the so called "DACH region". Overall 57 software houses from the DACH region responded to questions on software quality assurance techniques, tools, effort, and perceived quality after implementation. The results were complemented by interviews and results from related surveys. However, this is the first survey dedicated to software quality assurance during implementation in software houses, which provides due to central role of software development for the whole organization a valid source to investigate best practices. Furthermore, our survey does not only consider agile practices like other surveys (for instance, [5,6]) and especially also statistically investigates correlations between software quality assurance techniques and tools.

The results presented in this paper provide information on the state-of-practice and are equally relevant for research (by guiding it to relevant topics) and practice (by serving as a baseline for comparison).

This paper is structured as follows. Section 2 presents related work. Section 3 discusses the survey goal, design and execution. Section 4 presents results and discusses them. Finally, Sect. 5 concludes the paper.

2 Related Work

In this section, we summarize related results on software quality assurance during implementation from other surveys reporting respective results [3,5–8]. Relevant related results reported in these studies address the project effort spent on implementation, tool support during implementation as well as the usage of agile practices during implementation. In the following paragraphs we summarize these quantitative and qualitative results and later in Sect. 4 we relate them to our findings.

Regarding agile practices, Vonken et al. [5] found that the use of a coding standard correlates to the subjective satisfaction with the development process. 70 % of the participants responded to use coding standards regularly or extensively and only 4 % responded to never use any coding standard. In addition, Perez et al. [6] found that pair programming has a positive correlation with the perceived quality of the development process. Schindler [8] examined in a 2008 Austrian-wide survey that although the agile practice pair-programming is known by 71 % of all participants, it was only used by 46 %. Furthermore, all of the participants that claimed to use this practice, also admitted to not use pair-programming regularly but instead rarely or on demand [8]. Furthermore, a third of the 46 % also said to only use pair-programming in the case of complex tasks. Schindler [8] also noted that pair programming is important for knowledge exchange between senior and junior developers as well as to get new developers up to speed. Another observation made by Vonken et al. [5] is that pair programming correlates with unit testing and refactoring. A more unexpected observation made by the same authors is that unit testing and refactoring are unrelated, which is surprising as unit testing can be considered as safeguard during refactoring.

Regarding the effort spent, Garousi et al. [3] found that on average around 31 % of the total project effort was spent on implementation. This is more than two times the effort of the second most reported effort consuming activity, i.e., testing. Armbrust et al. [7] observed a higher amount of the total project effort allocated to implementation. According to their findings, on average around 48 % of the software development effort was assigned to development, the smallest amount assigned was 10 % and the highest 85 %.

Regarding tool support, Pérez et al. [6] found that version control (93 %) as well as bug notification and tracking (86 %) are commonly used tools. In contrast, tools that support the development process like continuous integration (45 %), testing (52 %) or configuration (45 %) are only used by about half of the respondents. Furthermore, Vonken et al. [5] found similar high usage ratios for version control systems (88 %). Also Garousi et al. [3] investigated the usage of tools [3] and found that static code analysis and automation tools are used by 64 % of all respondents. Whereas 24 % responded to never use and 12 % to seldom use this type of tools. Finally, Schindler [8] identified that the most frequently used tools to support agile development are unit tests (75 %), generators for documentation from source code (71 %) as well as continuous integration (39 %).

3 Survey Goal, Design, Distribution, and Analysis

This section provides the survey goal and research questions (Sect. 3.1), the survey design (Sect. 3.2), the survey distribution (Sect. 3.3), as well as the survey analysis (Sect. 3.4). Finally, Sect. 3.5 provides a summarizing timeline of the performed survey design, distribution and analysis activities.

3.1 Goal and Research Questions

The goal of this survey is to investigate the role of software quality assurance during the implementation phase in software houses from Germany, Austria and Switzerland. The target audience of the survey are therefore software houses that are located in Germany, Austria or Switzerland and do not operate in a domain that may impose restrictions on their software development, e.g., medical or automotive. Based on the goal and taking industrial relevance from experiences of the involved company Software Quality Lab into account, we raise the following four research questions (RQs):

RQ 1 *Which quality assurance techniques are used during development?*
RQ 2 *Which tool support is used for quality assurance during development?*
RQ 3 *How much effort is spent on implementation and integrated quality assurance?*
RQ 4 *How is the perceived software quality after implementation (including the integrated quality assurance)?*

3.2 Survey Design

In the survey design, we used and benefited from lessons learned and guidelines reported by other researchers in software engineering [9,10]. We therefore present the sampling plan, the questionnaire design and the performed pilot test.

Sampling Plan. The sampling plan describes how the participants are representatively selected from the target population. The first decision, whether a probabilistic, non-probabilistic or census sample should be considered, was already made by selecting the target audience. Given that no list of all companies exists that have the characteristics of to target audience, a truly probabilistic or census sample is not feasible. The first (probabilistic) would require an enumeration of all members of the target audience to select randomly participants and the later (census) can as well only be conducted if all individuals of the target audience are known. As a result, non-probabilistic sampling was chosen.

As a method to draw the sample from the population quota sampling with the two strata geographical location of the software house (Germany, Austria or Switzerland) and number of employees (less or equal 10, 11 to 100 and more than 100) was applied.

Overall 57 software houses, 19 from each of the three countries, evenly distributed over the three company sizes were selected and could be consulted within the given time and resources. Based on the activities relevant for software houses from the OECD [11] industry categories, i.e., 62 – Computer programming, consultancy and related activities, as well as 631 – Data processing, hosting and related activities; web portals, the overall number of software houses in Germany, Austria and Switzerland could be estimated based on data from governmental statistical offices. For Germany, the "IKT-BRANCHE IN

DEUTSCHLAND" [12] report identified 61,029 companies in 2013 that are classified with one of the two categories[1]. For Austria, the governmental statistical office reported 13,281 companies in the respective categories[2] in 2012. Finally for Switzerland, the federal statistical office measured 2008 in the census of companies[3] 15,466 companies that have amongst their main activities programming, information technology consulting and data processing. As a result, the total number of software houses in the DACH region can be estimated with 90,000 ($61,029 + 15,466 + 13,281 = 89,776$). Taking the population size of 90,000 into account, with the 57 participating companies a precision [9], which measures how close an estimate (resulting from the survey data) is to the actual characteristic in the population, of 87 % is achieved.

Questionnaire Design. The questionnaire was designed based on the experiences of Software Quality Lab and the involved researchers in conducting surveys as well as academic findings of related surveys (see Sect. 2) and the software engineering body of knowledge (SWEBOK) [13]. The knowledge and practical consultancy experience of Software Quality Lab was a valuable input to design the questionnaire. Furthermore, a technical report of a survey on software quality conducted by Winter et al. [14] in 2011 provided many useful insights for the questionnaire design. The questions included in the questionnaire were transformed into closed-ended questions and ordered by topic. The questionnaire was implemented and performed online with the survey tool LimeSurvey[4]. For software quality assurance during implementation five questions were raised, i.e., one question for RQ 1, one for RQ 2, two for RQ 3, and one for RQ 4, and embedded into a larger questionnaire on software quality processes. The questions of the questionnaire correspond to the research questions, where RQ 3 is split into two questions, one for the development and one for the bug fixing effort. The answer options for each of the five questions are shown in Figs. 2, 4, 6, 7, and 8, respectively. The complete questionnaire is available via the first author upon request.

Pilot Test. The questionnaire was validated *internally*, i.e., by the involved researchers and Software Quality Lab, as well as *externally* by six employees of software houses. Internally, there were several iterations and the involvement of researchers and industrialists guaranteed a high quality review from different perspectives. Externally, the reviewers provided valuable, written feedback to further improve the questionnaire.

3.3 Survey Distribution

The distribution of the questionnaires among the potential participants included a pre-notification, the invitation with the questionnaire, reminders and a thank-you letter. The survey distribution started on April 1, 2015. The participants

[1] http://bit.ly/1Sqfb3z.
[2] http://bit.ly/22IjjeS.
[3] http://bit.ly/22IkScL.
[4] http://www.limesurvey.org.

were selected by using Google Maps and searching for 'software company'. Searching for this term reveals all software companies at the related location. Furthermore, information about the number of employees for each found software house were determined. This allowed to come up with 450 participants – 50 small, 50 medium and 50 big software houses per country. Two weeks after the pre-notification emails were sent, the invitation emails with a link to the online survey were distributed. As a result, 13 participants responded to not wish to participate and 20 software houses participated. One reminder were sent in the middle of the survey (end of April 2015) to remember possible participants about the survey. Due to the low number of responses, additional 500 software companies were contacted via email. In addition, new participants were searched and contacted exclusively by phone to invite them to the survey. During three days within the last week, 200 potential software houses in Germany, Austria and Switzerland were called and asked for participation. In this three days 18 software houses could be convinced to participate. Thus, the response rate for the phone calls was 9 %, which is double the response rate of the email invitations (4 % for the first half of the survey and 3 % for the second). In the phone calls, also some of the reasons against the participation were mentioned. Amongst others, no time, no interest, already having participated in similar surveys and the absence of the respective decision maker were mentioned. The survey distribution ended on May 22, 2015.

3.4 Survey Analysis

The data was first analyzed *quantitatively* and then *qualitatively* by interviews with survey participants and evidence extracted from related work.

As the responses for each question were nominally scaled, the votes for each question were counted and then visualized in bar charts. Furthermore, Pearson correlation coefficients between answers were computed to find correlations within and between quality assurance techniques and tools to support implementation. The analysis was performed in IBM SPSS and the resulting correlation coefficients have been interpreted as suggest by Evans [15], i.e., in steps of 0.2 from very weak to very strong.

We performed 12 interviews with survey participants (i.e., 21 % of all participants) to triangulate the quantitative analysis and to identify the reasons behind some survey answers. The semi-structured interview type was chosen, because the structured interview limits the discussion freedom to enter unforeseen subtopics or ask questions that may arise during the interview. Another alternative would have been the unstructured interview. However, this form would have not allowed to ask prepared questions of interest that emerged during the analysis of the empirical survey. Telephone calls were used contact each participant in an economic and for the interviewee time- and place-flexible way. In the short interview one question on implementation was asked. In addition, the non-structured part of the interview followed subtopics of interest that were raised by the interviewee or that turned out as a result of previously conducted interviews to be of interest.

3.5 Survey Timeline

This section summarizes the survey design, distribution and analysis by providing the concrete timeline in which the respective activities were performed in 2015. Figure 1 shows the timeline for survey design, distribution and analysis activities. Activities with concrete dates in parentheses were performed in the given date range, the other activities were performed during the whole month.

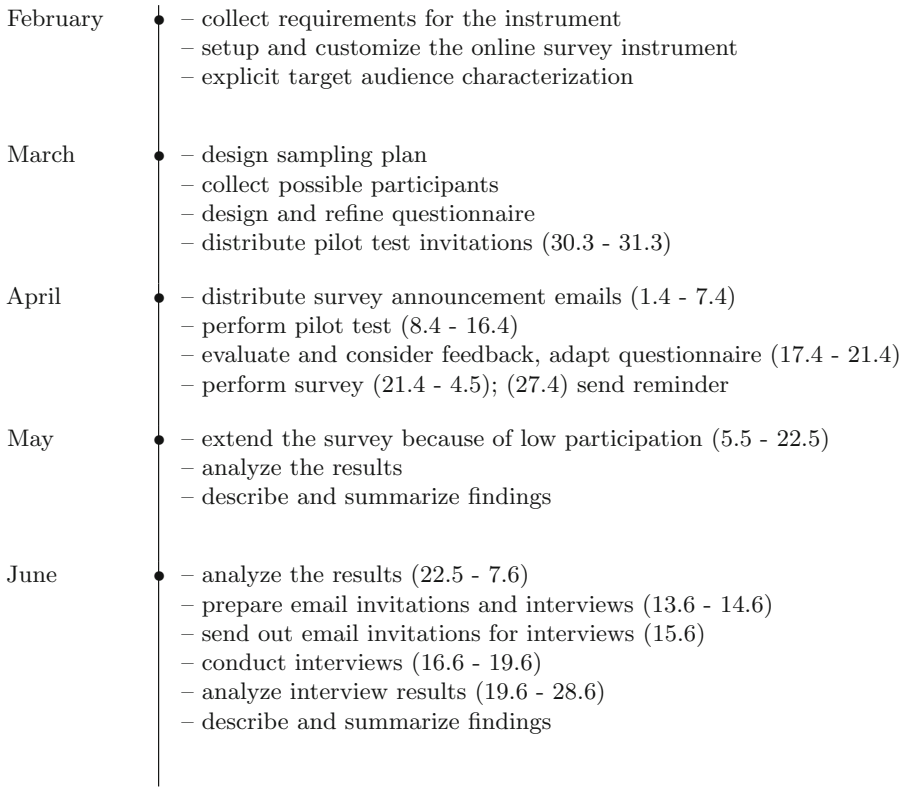

Fig. 1. Timeline of the survey.

4 Results and Discussion

In this section, we first present the demographics of our survey, then we present and discuss main findings for each of the four research questions, and finally we discuss threats to validity.

4.1 Demographics

Overall 57 software houses, 19 from Germany, 19 from Austria and 19 from Switzerland, participated in the survey. Most of the software houses (84 %) stated that they perform more than one type of software project. On average three types were stated. The three most common project types are development of web-applications (71 %), individual software (61 %), and standard software (56 %).

In the sample of 57 software houses small, medium and large companies are present with a similar frequency: 38 % of the companies are small-sized (up to 10 employees), 35 % medium-sized (11 to 100) and 26 % large-sized (more than 100 employees).

4.2 Main Findings

In this section we present the main findings for each of the four research questions.

RQ1: Quality Assurance Techniques. Figure. 2 shows the quality assurance techniques applied by the responding software houses during implementation. The most commonly used techniques are unit testing (68 %), code reviews (63 %) and coding guidelines (61 %). Only one participants responded to apply no technique at all.

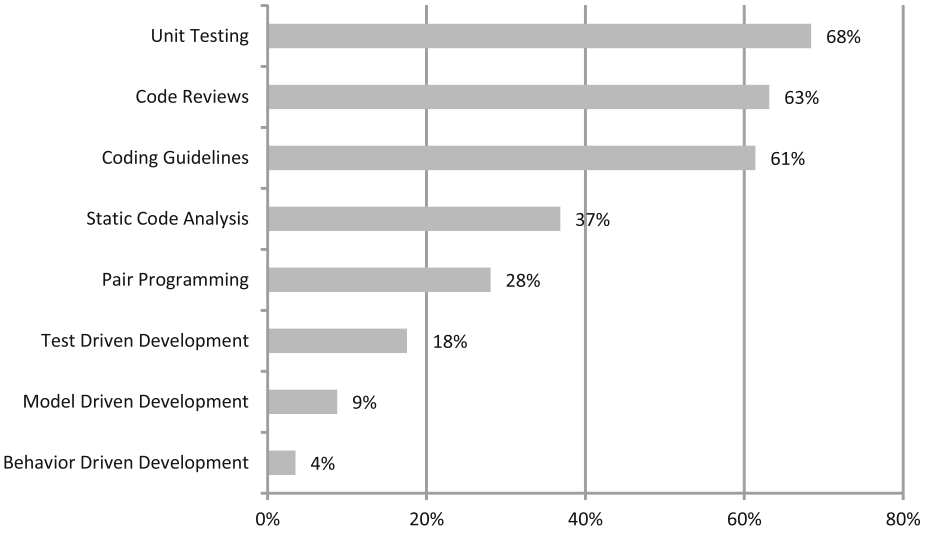

Fig. 2. Quality assurance techniques during development. One participant responded to apply no techniques at all.

So similar to Vonken et al. [5], we found a high ratio of people using coding guidelines. Furthermore, our results are also compliant with the finding of Schindler [8] that pair programming is only applied moderately.

Correlation analysis shown in Fig. 3 revealed six positive relationships between quality assurance techniques, i.e., using one technique increases the likelihood of using the related one. Static code analysis is related to coding guidelines, test-driven development (TDD) as well as unit testing. Furthermore, coding guidelines are related to TDD, unit testing to code reviews and model-driven development to behavior-driven development (BDD). Considering the relationships, it is surprising that static code analysis is not amongst the most commonly used techniques given its positive correlations to coding guidelines and unit testing which are both used by more than 60 % of the responding software houses. Another notable, relationship is between model-driven development and BDD that have only a significant positive correlation to each other, but not to other techniques.

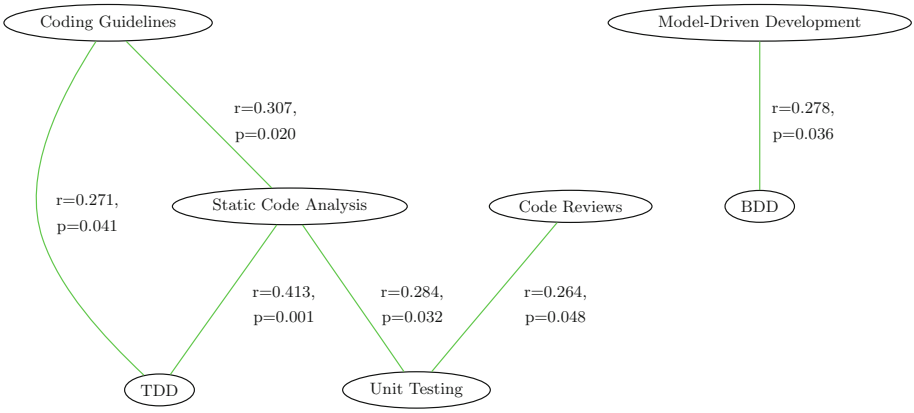

Fig. 3. Positive correlations between quality assurance techniques. Note that for all relationship $n = 57$, which is why it is not explicitly mentioned at every correlation.

Quality assurance techniques like unit testing, code reviews or the usage of coding guidelines are commonly practices according to the results of the empirical survey. In addition, reviews, tests during development, checklists and reviews at milestones are commonly used methods to control and support the software development process. Thus, it seems that implementation is well supported by respective quality assurance techniques. However, in the interviews with most participants the domain-specific side of the software, the rules and peculiarities of the domain, are seldom addressed explicitly, although they often lead to costly bugs.

RQ2: Tool Support. The participants were asked to indicate in which areas implementation is supported by tools. The results depicted in Fig. 4 show that

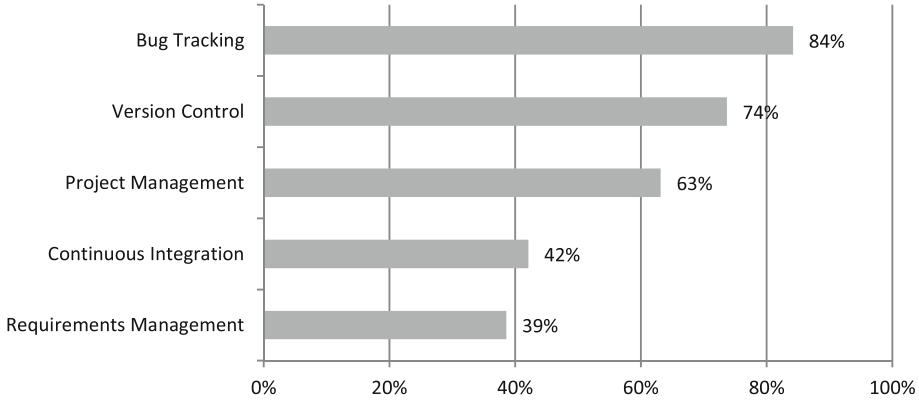

Fig. 4. Areas in which tools are used to support implementation.

bug tracking (84 %), version control (73 %) and project management (63 %) are the most common areas. Continuous integration (42 %) and requirements management (38 %) are mentioned only half as often as the other tool types.

Furthermore, relationships to quality assurance techniques were statistically analyzed. The calculation of each possible correlation revealed four significant relationships:

- Coding guidelines are in a positive, moderate strong correlation with the use of continuous integration ($r = 0.457, p = 0.000$) and version control tools ($r = 0.426, p = 0.001$).
- Static code analysis is in a positive and moderate strong correlation with continuous integration ($r = 0.454, p = 0.000$) and in a positive, weak relationship with version control. In addition, it also has a positive, weak relationship with bug tracking ($r = 0.331, p = 0.012$).
- Code Reviews are in a weak, positive correlation with version control ($r = 0.287, p = 0.030$) and bug tracking ($r = 0.367, p = 0.005$).
- Unit testing is in a weak, positive correlation with project management ($r = 0.264, p = 0.048$) and in a strong, positive with bug tracking ($r = 0.430, p = 0.001$).

Thus, coding guidelines and static code analysis are often used in environments with continuous integration and version control. Bug tracking tools are often used in environments in which also static code analysis, code reviews and unit testing are performed.

Correlations between the tool types have also been analyzed and are shown in Fig. 5: continuous integration tools are positively correlated with requirements management and version control tools, and bug tracking tools are positively correlated with version control and project management tools. One can observe that all tools are related directly or indirectly (via other tools). Thus, it seems that the introduction of one tool leads to the adoption of several others. This is

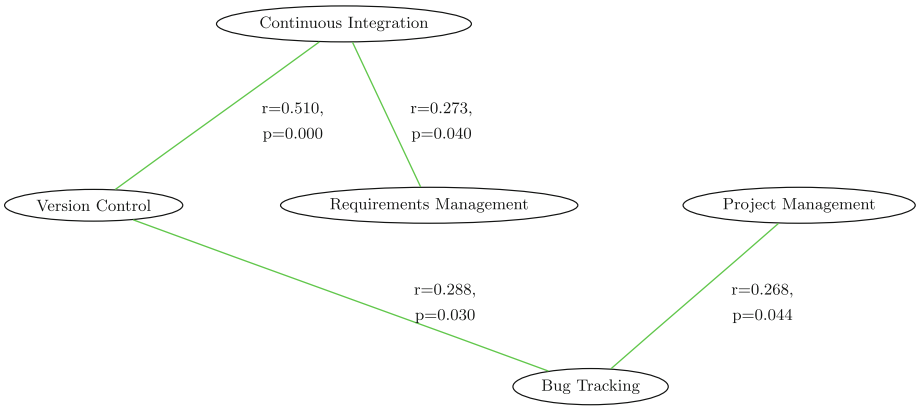

Fig. 5. Positive correlations between the tool types. Note that for all relationships hold that $n = 57$.

supported by the fact, that a high number of participants (64 %) stated to use three or more tools. Tools supporting implementation are therefore often used together.

Also several other surveys [3,5,6,8] found that different types of tools are commonly-used to support development. Our results confirm the findings of Perez et al. [6] who also found that version control and bug tracking tools are often used, but continuous integration only moderately. Furthermore, Vonken et al. [5] also found high usage rates of version control systems. Finally, Schindler [8] reported a similar usage rate of continuous integration (around 40 %) as we did.

RQ3: Effort. We asked for the ratio of the total project effort spent on implementation and integrated quality assurance. Figure 6 indicates a clear trend towards the range 41 % to 60 % of the total project effort, which was selected by 49 % of all participants. Thus, it seems that in practice most often a ratio between 41 % and 60 % of the total project effort is spent on implementation.

So in our case, the effort spent on programming is higher than reported in Garousi et al. [3] (i.e., 31 %) and compliant with the finding of Armbrust et al. [7] who reports a ratio of the overall effort spent on implementation of 48 % (and integrated quality assurance).

The most important integrated quality assurance technique during implementation is bug fixing. Figure 7 shows that with respect to the ratio of the total project effort developers spend on bug fixing, most participants (63 %) responded that up to 20 % of the total project effort is spent on bug fixing. Even higher efforts were stated by 25 % of the respondents. Moreover, 12 % could not estimate the effort for bug fixing, which may indicate that they are not aware of the effort that is used for bug fixing or do not measure it.

According to two interviewees, a reason for the relatively high bug fixing effort is that a small amount of bugs requires a large amount of time and effort

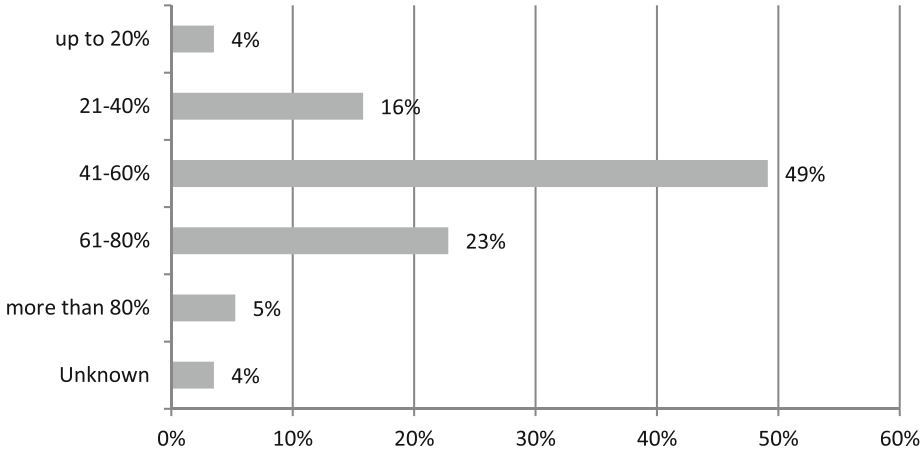

Fig. 6. Amount of total project effort dedicated to implementation.

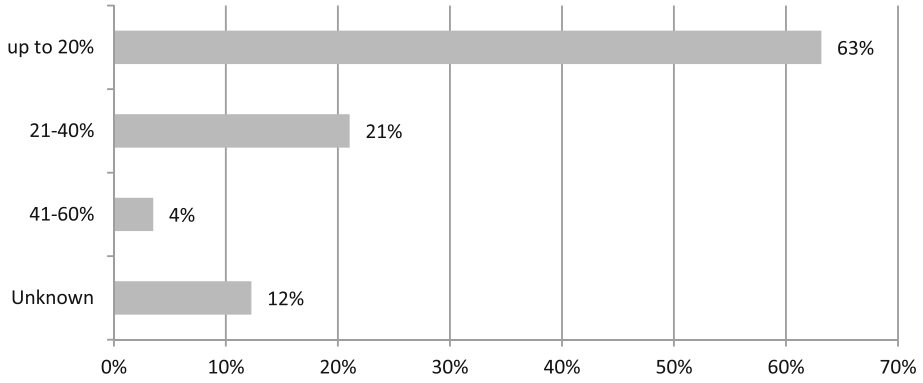

Fig. 7. Ratio of total project effort spent on bug fixing by developers.

to be fixed. An example stated by one interviewee was a wrong variable name caused by copying similar source code. Another reason that was commonly mentioned are incomplete requirements specifications that result in bugs that are not caused by wrong code, but by missing edge cases that were not properly specified. One interviewee explicitly highlighted that not technical aspects cause high bug fixing efforts, but domain aspects. Developers typically do not have the same deep understanding of the business domain of the software as for instance requirements engineers or customers have. This often results in bugs caused by missing or wrongly interpreted domain-specific aspects of requirements.

Given that more developers work on the same code, bugs may be introduced by other employees that have to fix them. As a result, it could be that a higher ratio of the total effort is spent on bug fixing with an increasing number of employees. This is also supported by the fact that with an increasing number of

developers also the system complexity increases, which makes it more difficult to find and fix bugs. The statistical analysis confirmed this correlation and identified a strong positive correlation between the number of employees in software development and the effort for bug fixing ($r = 0.485, n = 56, p = 0.000$). Thus, the more developers a software company has, the more effort is spent on bug fixing.

RQ4: Perceived Quality. We asked for the perceived software quality after implementation (including the integrated quality assurance) at the handover to system testing. Figure 8 shows that 41 % responded to perceive the quality neither as good nor as bad, 56 %, more than half of all companies, rated the quality as rather good to good. Only one participant mentioned that the quality of the software at this point in development is rather bad. Thus, it seems that the quality of the software is considered to be in an at least rather good quality in most cases. This may indicate that the applied quality assurance measures during the development are working.

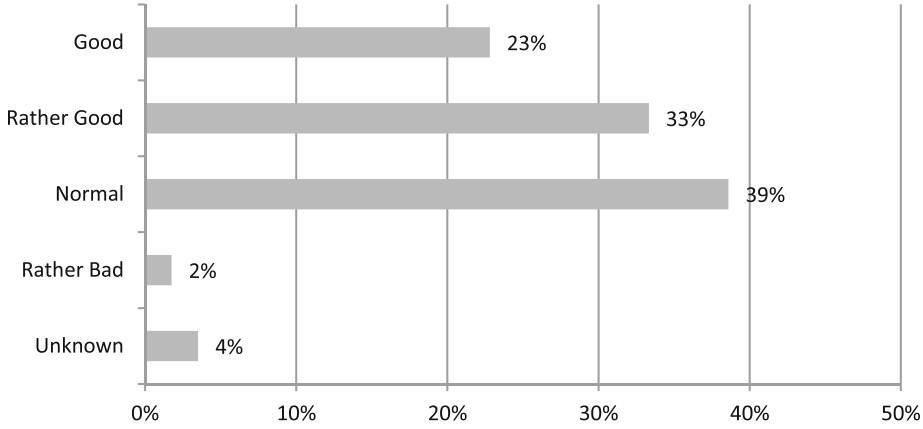

Fig. 8. Perceived software quality after implementation with integrated quality assurance.

The software quality in development may also depend on the number of developers. Thus, the correlation between the number of employees in total and the number of employees in software development, respectively, as well as the perceived software quality before testing was statistically analyzed. As a result, two weak, negative correlations were found:

- Perceived software quality is in a weak, negative correlation with the number of employees ($r = -0.334, n = 55, p = 0.013$).
- Perceived software quality is in a weak, negative correlation with the number of employees in software development ($r = -0.307, n = 55, p = 0.023$).

These two negative correlations indicate that the higher the number of employees, the lower the perceived software quality is.

4.3 Threats to Validity

In this section we discuss critical threats to validity and how we addressed them. One critical threat to validity is the limited number of participating software houses, i.e., 57. Nevertheless by estimating the overall number of software houses in the DACH region, a precision of 87 % of our results could be reached. To further increase validity of the results, the questionnaire was triangulated by interviews and evidence from related studies. Conclusions on correlation were drawn based on statistical significance.

Furthermore, it has to be mentioned that the study was only conducted in the DACH region and that the validity of the results is therefore limited to that region. But due to the facts that similar surveys were initially performed in the DACH region and then replicated in other regions with similar results [16,17] and that we could not find significant differences between the results from Germany, Austria and Switzerland, we think that the survey will deliver similar results in other regions as well. However, a replication in other regions, which is already planned as future work, is required to confirm this statement. The questionnaire itself was constructed based on the experiences of Software Quality Lab and the involved researchers in conducting surveys as well as academic findings of related surveys. Furthermore, the questionnaire was refined by internal and external reviews in several iterations.

5 Conclusion

This paper presented a survey on quality assurance during implementation in software houses from Germany, Austria and Switzerland. Overall 57 software houses participated. Results from the survey show that the most common software quality assurance techniques used during implementation are unit testing, code reviews and coding guidelines. Most tool support is used in the areas of bug tracking, version control and project management. Due to relationships between the used tool types, it seems that the introduction of one tool leads to the adoption of several others. We also found that coding guidelines and static code analysis are often used in environments with continuous integration and version control. Bug tracking tools are often used in environments in which also static code analysis, code reviews and unit testing are performed. Bug fixing takes a significant ratio of the overall project effort assigned to implementation. Furthermore, we found that the more developers a software company has, the more effort is spent on bug fixing. Finally, more than half of all companies rated the quality after implementation as rather good to good and there seems to be a negative correlation between the number of employees and the perceived software quality.

In future, we plan to replicate the survey in other regions and to perform case studies to investigate in which context (for instance, with respect to the process model applied) specific quality assurance techniques during implementation are promising. Based on the results of these empirical studies we plan to derive practical guidelines to improve quality assurance during implementation.

Acknowledgments. The authors thank Software Quality Lab GmbH and especially its CEO Johannes Bergsmann for joint operation of this survey as well as all participating companies, interview partners and colleagues who helped to make this survey possible.

References

1. Venkitaraman, R.: Software quality assurance. Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET) **2**, 261–264 (2014)
2. ISO, IEC, IEEE: Iso/iec/ieee 24765: 2010 - systems and software engineering - vocabulary. 418 (2010)
3. Garousi, V., Coşkunçay, A., Betin-Can, A., Demirörs, O.: A survey of software engineering practices in turkey. J. Syst. Soft. **108**, 148–177 (2015)
4. Roebuck, K.: Legacy Application Modernization: High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors. Emereo Publishing, Aspley (2012)
5. Vonken, F., Brunekreef, J., Zaidman, A., Peeters, F.: Software engineering in the Netherlands: the state of the practice. Technical report, Delft University of Technology, Software Engineering Research Group (2012)
6. Pérez, J., Puissant, J.P., Mens, T., Kamseu, F., Habri, N.: Software quality practices in industry-a pilot study in wallonia. University of Mons, Technical report (2012)
7. Armbrust, O., Ochs, M., Snoek, B.: Stand der praxis von software-tests und deren automatisierung. Fraunhofer IESE-REPORT NR 93 (2004)
8. Schindler, C.: Agile software development methods and practices in austrian it-industry: results of an empirical study. In: 2008 International Conference on Computational Intelligence for Modelling Control and Automation, pp. 321–326. IEEE (2008)
9. Kasunic, M.: Designing an effective survey. Technical report, DTIC Document (2005)
10. Linaker, J., Sulaman, S.M., Maiani de Mello, R., Höst, M., Runeson, P.: Guidelines for conducting surveys in software engineering v. 1.0 (2015)
11. On Indicators for the Information Society, W.P: Information economy - sector definitions based on the internet standard industry classification (isic 4). DSTI/ICCP/IIS(2006)2/FINAL (2007)
12. Bundesamt, S.: Ikt-branche in deutschland - bericht zur wirtschaftlichen entwicklung (2013)
13. Society, I.C: Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0. IEEE Computer Society Press (2014)
14. Winter, M., Vosseberg, K., Spillner, A., Haberl, P.: Softwaretest-umfrage 2011-erkenntnisziele, durchführung und ergebnisse. In: Software Engineering, pp. 157–168 (2012)

15. Evans, J.D.: Straightforward Statistics for the Behavioral Sciences. Brooks/Cole, Salt Lake City (1996)
16. Fernandez, D.M., Wagner, S., Kalinowski, M., Schekelmann, A., Tuzcu, A., Conte, T., Spinola, R., Prikladnicki, R.: Naming the pain in requirements engineering: comparing practices in Brazil and Germany. IEEE Soft. **5**, 16–23 (2015)
17. Kalinowski, M., Felderer, M., Conte, T., Spínola, R., Prikladnicki, R., Winkler, D., Fernández, D.M., Wagner, S.: Preventing incomplete/hidden requirements: reflections on survey data from Austria and Brazil. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2016. LNBIP, vol. 238, pp. 63–78. Springer, Heidelberg (2016). doi:10.1007/978-3-319-27033-3_5