Is it about Human(itie)s? Experiences from Software Projects across three Faculties

Hagen Peukert[1]

Abstract: Considering the experience with software development in research projects in three different faculties, it is argued here that the issue of underspecification and `finding out what is really wanted` is not restricted to the Humanities. Rather, it occurs in Science and Social Science seemingly at the same ratio. However, in absence of representational data, statements at which exact shares the phenomenon really occurs, lead up the garden path. Hence, the question is raised whether the focus of software development in the `Humanities` should shift to `human` per se describing the fact that underspecification is a human trait of complex planning and problem solving behavior, but independent from the faculty's epistemology. And possibly it comes to the fore in science more bluntly and it is particularly apparent in the Humanities, i.e. less blended. In fact, any specific account of software development in the Humanities is misleading because the focus on the Humanities blends the hermeneutics inherent in scientific work in general.

Keywords: Digital Humanities; Scientific Software Development; Epistemology

1 Introduction

Within the hermeneutic circle [Ga65], cognition and research evolve simultaneously to higher levels of conception and eventually both, research question and its array of answer attempts, merge into a coherent whole – the final state of explanation in the world of ideas. If software is to be programmed assisting to achieve the project goals, then the specificity of the research question is reflected in the evolution of the software functionality, which only towards the end of the project, if at all, reaches the level of concreteness needed beforehand and which software development presupposes compellingly so that the problem space can be sufficiently described.

Left aside that intensive refactoring of software after the project is a solution that works generally but does not help much if the software should produce results of a vaguely defined problem space right from the inception, the question arises which is the best strategy to be followed here: could software design patterns ever be developed to capture this process, should methods such as agile programming are further optimized or both? More importantly, however, is the question if the phenomenon is restricted to the Humanities? Given that

[1] Universität Hamburg, Zentrum für Forschungsdatenmanagement, Monetastraße 4, 20146 Hamburg, Germany
hagen.peukert@uni-hamburg.de

doi:10.18420/inf2019_ws16

the Heideggerian hermeneutics describe a general truth, one should then make similar observations in all endeavors of scientific experimentation [Se92]. In other words, it is not characteristic to Humanities, but to all faculties in which genuine knowledge is generated. So, the more plausible and correct zero hypothesis is not to simply claim that software development in the Humanities is different, but to act on the assumption that scientific software development is similar throughout all faculties. In this case, it had to be shown that Humanities software projects are significantly different from other faculties.

The argument brought forward in the paper at hand is supported by several software projects in the faculties of Earth Science (Climate Research, Sea Ice Remote Sensing), Linguistics (Quantitative Language Science, Diachronic Derivational Morphology), History (Ancient History, Greek Epigraphy), Musicology (Historical Musicology, Opera Omnia Critical Edition), and Education Science (Empirical Educational Research, Intercultural and International Comparative Education) all located at Universität Hamburg. So three different faculties – Science, Humanities, and Social Science – are object of the observations presented here. The data considered stem from qualitative interviews [Pe15], experiences from software projects [Pe19; Ar16; In17] and research projects [Pe18; Pe14].

While it is clear that the data is hard to evaluate since it does not allow to make any general statements from an arbitrary and too small sample of different methods clearly biased to Humanities projects, one has still to acknowledge that there is no reliable source of representative studies on that topic at all that goes beyond anecdotal evidence. Studies looking into scientific software development and models can be found in engineering sciences and in the area of high-performance computing, yet they do not address the differences across faculties [NFS10; Ha09; SM08; Se08]. Claims such as `Humanities projects enjoyed a special status` are likely be derived from individual observations as well, which are neither systematically collected nor comprehensibly documented. And still, the fragmentary data brought forward here contributes to the overall stock of investigations into this field and the sum of all collected observations marks a justified point of departure for systematic and representative studies and analyses. As far as the data at hand is concerned, it merely states the existence without assertions about its statistical significance and distribution.

2 Results

Tab. 1 gives a short summary of the key variables in the studied projects, whereas three symbols correspond to the following categories: `available` (++), `partly` (+), `nonexistent` (−). From the projects a few suggestions on what is important to focus on when carrying out software projects in these areas of research can be derived in six points – some of which support the suggestions made in the literature [Ha09; Se08].

First, whatever it is planned beforehand and put down as a requirement specification is subject to sudden and permanent change. In addition, the once agreed requirement specification is important for one's own reference and for a task schedule, but it is not a document of

	Social Science	Science	Humanities		
	LiMA	Remote sensing	Opera omnia	EDaK	Morphilo
Systematic testing	–	++	–	–	+
Documentation	–	+	–	++	++
Version control	–	–	++	–	++
Design patterns	–	–	–	–	–
Handling of data	++	++	+	+	+

Tab. 1: Evaluation of Software Projects

agreement to what should be done at which time. It is advisable to keep a separate document as a specification with the versioning history of the software so that at each stage of software release its corresponding requirement specification (that is valid only in that point in time) is available.

Second, as implied in the previous paragraph, very short cycles in the developmental process is highly advantageous. The sprints can be organized around single software features internally, i.e. one feature in one cycle, and attributable to the requirements specification. Yet not in line with all the literature, for the presentation at and feedback from the project holders, it makes little sense to present single features. From the experience in all projects, project leaders like to have clickable prototypes incorporating a collection of features that correspond to the project deliverable (milestone) in the respective fund proposal.

Third, visualization in the form of screenshots or clickable forms is paramount to class diagrams, entity relationship diagrams of all kinds, sequence diagrams, or any other kind of UML representation all together. Efficiency measures have secondary status at the intermediate stages before the final procurement. It makes sense to sit down and design the layout with paper and pencil right at the project meetings. Most of the implicit knowledge will become accessible in and around these discussions. Moreover, a staging server independent of the programmer's development environment takes away a lot of the burden. On the staging system, only the release bundles – best in the form of functional prototypes – should be deployed; no development takes place there. These machines are accessible to the project holders and the application developed so far can be tested `before` the feedback meeting takes place.

Implicitly derivable from the other results is, fourth, extending the data model easily is more important than the model efficiency. Since it is very likely that new data fields (representing variables) will be added while others will be deleted and the relation of the fields to each other will change all the time, a flat data model is a better starting point than an elaborated structure with little flexibility.

Fifth, precaution should be given to rights and roles. Indeed the project leader has to manage the projects progress and he or she needs to control the working processes. This is,

however, different from the tasks carried out by the staff, in which the software program is supposed to give assistance. A discussion with the entire project staff usually makes clear that administrator rights do not reflect the power hierarchy of the project, but mirror the responsibilities of the tasks in a project. Priority is to be delegated to the project tasks and not the power relations in the project.

Sixth, also not to be seen in the literature, the relation between effort and result should be carefully considered. Even if a feature is technically doable, it does not mean it must be put in the application. This is also true if the project leader makes a strong case by referring to other commercial or non-commercial software, e.g. an application, in which some software function is implemented. The cost of a feature often happens to be completely underestimated. So to restrict expectation as well as to lower the cost of the "nice-to-haves" as selling features should be clearly communicated. This is different from blocking key features!

In summary, these more direct consequences of experiencing software development across the three faculties show a homogeneous picture, i.e. the faculty (and in a wider sense the subject of investigation in that faculty) does not seem to be a dependent variable. Put briefly, there is no faculty that shows a specific problem, around which one could build a predictive model. Possibly one would find the variable `faculty` to be on the same factor as an intervening variable that is not further explored as of yet. As a first assumption, the planning behavior and complex problem solving of humans in general [Kl93; St93] has a lot to offer to be an intervening factor of that kind.

3 Conclusion

Surveying the software projects carried out in the Humanities and comparing them to best practices in computer science, it becomes clear without question that current standards in software development are widely ignored. However, doubts could be raised how best to explain this phenomenon. The special status of the Humanities within the area of science and its epistemology are stated as a possible cause. It is further suggested that new methods in software development and design patterns are sought to be engineered to optimize software development in Humanities projects. Yet, the other possibility is that the Humanities as such have no significant causal effect, but it is really the human behavior and the general problem solving mechanisms usually employed by human beings, that is, how humans approach scientific problems.

The epistemology in the Humanities then resembles the scientific process of the genuine `unknown` much closer than other faculties, in which the `unknown` is better known, i.e. methods and procedures are similar and only adopted to similar problem spaces for which the approximate boundaries of the solution, although not specifically, can be roughly provided.The important take-home message would thus be to focus on human traits and their mechanisms of problem solving as brought forward by the literature on human-centered

design rather than seeking the cause in the epistemological processes and the subject of the Humanities.

It certainly would be enlightening to see if several follow-up projects to a given Humanities project would still show the same pattern as opposed to the first attempt of a Humanities research project. In other words, as a research question matures over time the state of knowledge including the process knowledge and newly established methods underlie effects of experience [Re02]. Thus, the coming of age of a project changes the planning procedure that scientist are able to apply. Their acquaintance with similar recent project results will coin their subsequent planning behavior and approaches to solve them.

To conclude, I would like to plead for Humanities projects as a model case to figure out which methods are needed for software development in scientific projects, in which, independent of the faculty, the human capacities of foreseeing the complete array of possible outcomes is limited resulting in less precise assumptions that in the course of time and experience crystallize to a concrete heuristic.

References

[Ar16] Arbeitsbereich Alte Geschichte, 2016, URL: https://www.epigraphik.uni-hamburg.de.

[Ga65] Gadamer, H.-G.: Wahrheit und Methode : Grundzüge einer philosophischen Hermeneutik. Mohr, Tübingen, 1965.

[Ha09] Hannay, J. E.; MacLeod, C.; Singer, J.; Langtangen, H. P.; Pfahl, D.; Wilson, G.: How do scientists develop and use scientific software? In: SECSE 09, ICSE Workshop on Software Engineering in Computational Science and Engineering. Pp. 1–8, May 2009.

[In17] Institut für Historische Musikwissenschaft, 2017, URL: https://www.selle.uni-hamburg.de.

[Kl93] Kluwe, R. H.: Knowledge and performance in complex problem solving. In (Strube, G.; Wender, K.-F., eds.): The cognitive psychology of knowledge. Elsevier, Amsterdam, pp. 401–423, 1993.

[NFS10] Nguyen-Hoan, L.; Flint, S.; Sankaranarayana, R.: A survey of scientific software development. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. Pp. 12–21, Sept. 2010.

[Pe14] Peukert, H.: The Morphilo Toolset: Handling the Diversity of English Historical Texts. In (Ammermann, A.; Brock, A.; Pflaeging, J.; Schildhauer, P., eds.): Facets of Linguistics: Proceedings of the 14th Norddeutsches Linguistisches Kolloquium 2013. Peter Lang, Frankfurt, pp. 161–172, 2014.

[Pe15] Peukert, H.: Softwareentwicklung am CliSAP Arbeitsbereich Meereisfernerkundung: Erkenntnisse eines qualitativen Interviews, 2015, URL: https://www.dokserv.gwiss.uni-hamburg.de/receive/mir_mods_00000181.

[Pe18] Peukert, H.: Merging Community Knowledge and Self-Interest to Build Language Resources: Architecture and Quality Management of a Take-and-Share-Approach of Word Annotations. In (Burghardt, M.; Müller-Birn, C., eds.): INF-DH-2018. Gesellschaft für Informatik, Frankfurt, 2018.

[Pe19] Peukert, H.: Auf dem Weg zu einer digitalen Forschungsumgebung: Die Entwicklung spezieller Werkzeuge für das Projekt- und Datenmanagement. In (Duarte, J.; Gogolin, I.; Klinger, T.; Schnoor, B.; Trebbels, M., eds.): Sprachentwicklung im Kontext von Mehrsprachigkeit – Hypothesen, Methoden, Forschungsperspektiven. Springer, Berlin, in press, 2019.

[Re02] Renn, J.: Challenges from the Past. Innovative Structures for Science and the Contribution of the History of Science. In (Renn, J., ed.): Innovative Structures in Basic Research. Max-Planck-Gesellschaft, München, pp. 25–36, 2002.

[Se08] Segal, J.: Models of scientific software development. In: SECSE 08, First International Workshop on Software Engineering in Computational Science and Engineering. May 2008.

[Se92] Seiffert, H.: Einführung in die Hermeneutik : die Lehre von der Interpretation in den Fachwissenschaften. Francke, Tübingen, 1992.

[SM08] Segal, J.; Morris, C.: Developing scientific software. IEEE software 25/4, pp. 18–20, 2008.

[St93] Strauß, B.: Confoundations in complex problem solving. On the influence of the degree of correct solutions on problem solving in complex situations. Holos, Bonn, 1993.