Why Good Developers Write Bad Code: An Observational Case Study of the Impacts of Organizational Factors on Software Quality

Marina Gabriela do Amaral Santos

Software Quality and Measurement – 2016/2

Mathieu Lavallée, Pierre N. Robillard

2015 IEEE/ACM 37th IEEE International Conference on Software Engineering

Introduction

Current studies on software development environments have focused on improving conditions for the software development team: implementation of appropriate processes, hiring skills, use of appropriate communication tools and equipment, etc. Among these software development conditions are factors from the organization. However, while such factors are perceived as important, there is little empirical evidence of their effect on the quality of software products.

Introduction

- This study is based on ten months of observation of an in-house software development Project within a large telecomunications company.
- The observation was conducted during mandatory weekly status meetings, where technical and managerial issues were raised and discussed.
- This paper describes cases depicting the complexity of organizational factors and reports on ten issues that have had in negative impact on quality, followed by suggested avenues for corrective action.

Related Work

A work done in 1998 by Jaktman [3] showed that organizational values can have an impact on software architecture. She presents two interesting cases: the first revolves around a company who made software development dependent on the marketing department.

As Jaktman writes: “This resulted in software quality activities being given a low priority”. The outcome was a poor architecture, with a lot of code duplication and high error rates.

Related Work

The second case presents a company who was officially committed to quality, but was ambivalent about how to implement it. Management promoted arious quality approaches, but ultimately resources failed to materialize.

The *“frequent changes to product priorities affected the code: incomplete implementation of software changes left dead code and code fragments”*.

C. B. Jaktman, "The influence of organisational factors on the success and quality of a product-line architecture,"

Methodology

- Project:
  - a two-year project for an internal client called “Module” used in the company’s internal business processes.
- Technologies involved:
  - Module includes legacy software written in COBOL, Web interfaces, interactions with mobile devices and multiple databases.
- Purpose:
  - To manage work orders . To do this, it needs to extract data from multiple sources within the enterprise (employee list, equipment list, etc) and send it to multiple databases (payroll, quality control, etc).

Methodology

- Data Collection:
  - The study is based on non-participant observation of the software development team's weekly status meetings.
  - These meetings consisted of a mandatory all-hands discussion for the eight developers assigned mostly full-time to the project, along with the project manager.
  - The meetings also involved, as needed, developers from related external modules, testers, database administrators, security experts, a quality control specialist, etc.
  - The meetings involved up to 15 participants, and up to five stakeholders through conference calls.

Methodology

- Validation:
  - A lot of information about the developers and product. However, they provided only the perspective of the meeting participants.
  - The observer performed a semi-structured interview with two managers from different departments (operations and marketing) in order to obtain their views.
- Threats to Validity:
  - The non-participant observer was not familiar with the technical terms used by developers, a common issue for non-participant observation studies.
  - As a single case study, generalization of the issues identified can be disputed. However, given the current small number of non-participant observational studies in software engineering, it is surmised that the data collected in this study remain interesting for future researchers.

Observações

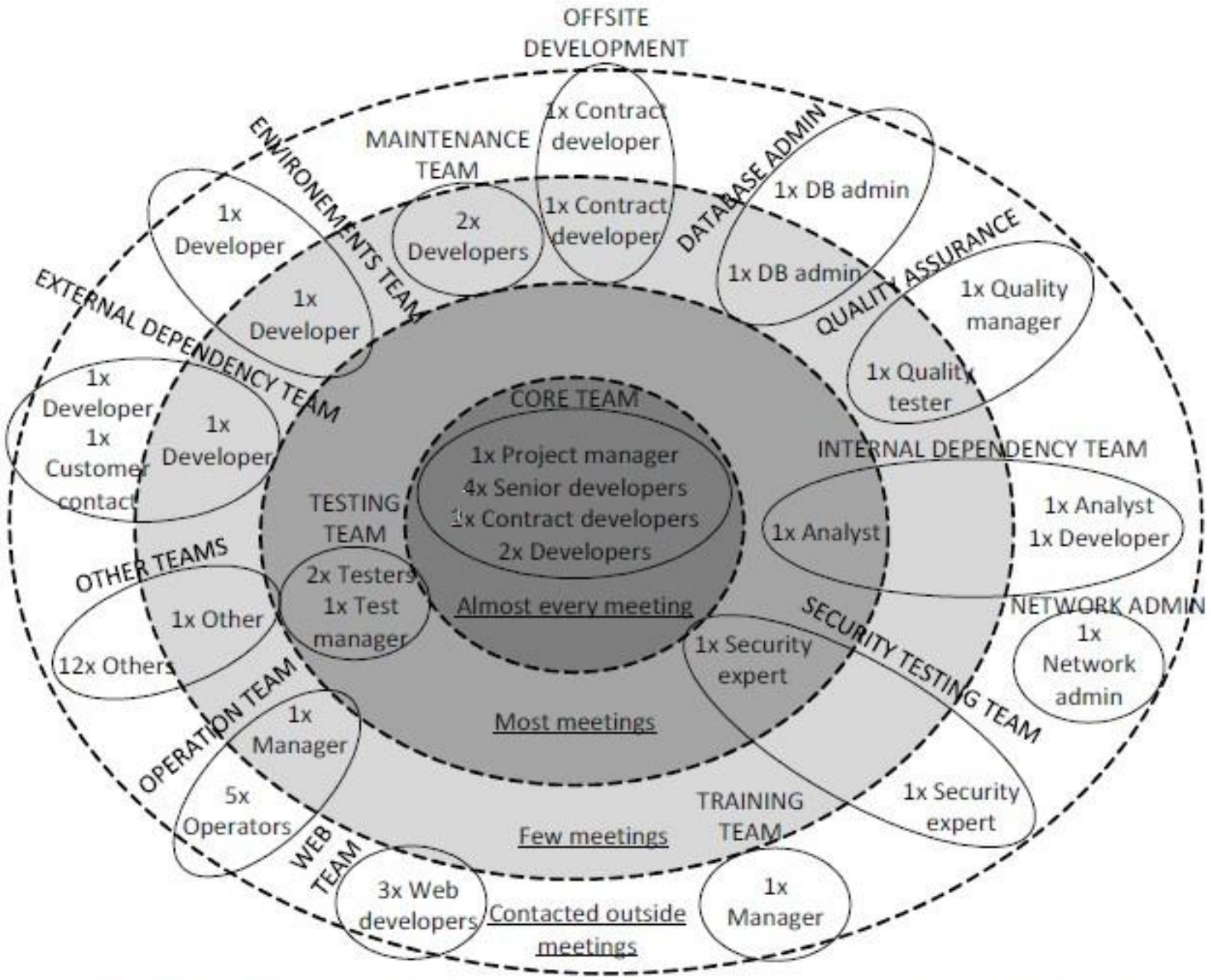

Figure 1. People and teams involved in the project, and how often they were in contact with the core development team.

TABLE I. ORGANIZATIONAL ISSUES WITH AN IMPACT ON QUALITY AS OBSERVED DURING THE STUDY.

Observed issue	Observed evidence	Observed quality flaw	Suggested corrective action
Documenting complexity	Large amount of old poorly documented software. Difficulty in managing large amounts of documentation.	Some requirements are discovered late in the development process. Some undocumented features were not implemented; others were patched together very quickly.	Once a project is completed, the team must ensure that the "What" and "Why" of each software item are properly documented.
Internal dependencies	Lots of interdependencies between software modules. Conflicts between projects on the scheduling of deployment.	Compatibility between modules is patched quickly and haphazardly.	In the cases of parallel development of inter-dependent software modules set up a negotiation table to solve conflict between the development teams.
External dependencies	Long-term dependencies to third-party libraries. Change requests to third parties results in costly delays.	Contractual obligations with third parties force poor quality design choices in order to minimize change requests.	Make sure that the development team is aware of the CMMI-ACQ or ISO12207 processes for negotiating with third parties.
Cloud storage	Contractual obligations with third parties do not support vulnerability testing.	Vulnerability testing is performed late and quickly.	Make sure that testers are involved when negotiating with a third party for a potentially vulnerable software component.
Organically grown processes	Processes emerge as the need arise, usually after crises. Defects found by users are documented but do not reach developers. Environment setup process unclear for developers. Patches in testing follow a stringent process, even when nothing works.	Frontiers between processes hinder information exchanges; developers must work with missing details.	Plan organization-wide process reviews to detect isolated processes and to promote information flows between processes.

TABLE I. ORGANIZATIONAL ISSUES WITH AN IMPACT ON QUALITY AS OBSERVED DURING THE STUDY.

Observed issue	Observed evidence	Observed quality flaw	Suggested corrective action
Budget protection	Cheaper to build a wrapper instead of solving the issue once and for all. A software item has twelve such wrappers.	Developers patch instead of fixing issues because fixing would cost too much.	Planned special budget items to support long lasting corrections or corrections that are likely to benefit many modules.
Scope protection	Explicit calls by team members to "protect the scope" of the project. Requirements are transferred to other projects as much as possible.	Project constraints means that teams must protect the scope of their project against change requests from other teams.	Projects with strict deadlines are risky, and should be carefully monitored to avoid last minute unplanned activities.
Organizational politics	Change request refused because the team did not contact the right person. Environmental issues resolved when the right manager was contacted. Inter-module issues resolved when upper manager applied pressure.	Managers and developers obtain better results by calling in favors from outside the software engineering process. Novices who do not know who to contact to obtain information will produce worse software.	Team members should maintain a careful balance between the flows of information within formal development processes and informal human interactions.
Human resource planning	The two developers assigned since the beginning of the project are contractual developers whose contract is about to expire.	The team will lose all knowledge of the beginning of the project, including details on the requirements and analysis phases. Documentation of the Module will be incomplete and/or inaccurate.	Team members should make sure that knowledge is appropriately distributed amongst them. For example, pair programming is a practice which can promote knowledge sharing.
Undue pressure	Managers and senior developers give direct orders and threats to the team.	The issuers of the orders and threats might not have all relevant information which could results in ill-defined priorities. Bypassing the normal team structure undermines its decision-making process.	Any intrusion into the team dynamics by outsiders should be done very carefully.

Conclusion

- The objective of this paper is to provide accurate empirical data on the state of software engineering in practice in a professional environment.
- While these issues might not affect project success, our observation shows that they do affect software quality. And quality factors can have a major impact on maintenance costs.
- If the Module developed in this project is successfully deployed, it will likely be used for the decades to come. The design flaws introduced because of the organizational issues presented here will no doubt come back to haunt at least a generation of developers to come, as the code written today will be tomorrow's legacy code.
- **Is this the kind of legacy we want to leave for future software developers?**

Why Good Developers Write Bad Code: An Observational Case Study of the Impacts of Organizational Factors on Software Quality

Marina Gabriela do Amaral Santos

Software Quality and Measurement – 2016/2