Effects of Virtual Development on Product Quality: Exploring Defect Causes

J.C. Jacobs
Philips Semiconductors
Eindhoven,The Netherlands
jef.jacobs@philips.com

J.H. van Moll
Philips Semiconductors & Sioux
Technical Software Development
Eindhoven, The Netherlands
jan.van.moll@philips.com

P.J. Krause
University of Surrey
Guildford, UK
p.krause@eim.surrey.ac.uk

R.J. Kusters
Eindhoven University of Technology
Eindhoven, The Netherlands
r.j.kusters@tm.tue.nl

J.J.M. Trienekens
Eindhoven University of Technology
Eindhoven, The Netherlands
j.j.m.trienekens@tm.tue.nl

Abstract

This paper explores the effects of virtual development on product quality, from the viewpoint of "conformance to specifications". Specifically, causes of defect injection and non- or late-detection are explored. Because of the practical difficulties of obtaining hard project-specific defect data, an approach was taken that relied upon accumulated expert knowledge. The accumulated expert knowledge based approach was found to be a practical alternative to an in-depth defect causal analysis on a per-project basis. Defect injection causes seem to be concentrated in the Requirements Specification phases. Defect dispersion is likely to increase, as requirements specifications are input for derived requirements specifications in multiple, related sub-projects. Similarly, a concentration of causes for the non- or late detection of defects was found in the Integration Test phases. Virtual development increases the likelihood of defects in the end product because of the increased likelihood of defect dispersion, because of new virtual development related defect causes, and because causes already existing in co-located development are more likely to occur.

Keywords: Virtual development, Product Quality, Defect injection, Defect detection, Defect Causal Analysis

1 Introduction

The objective of this paper is to investigate the possible effects of virtual development on the quality of the delivered product, in particular the exploration of defect causes. Virtual development is the development of a product (or product family) by a virtual team. A virtual team is a team distributed across space, time, and organization boundaries, and linked by webs of interactive technology [13]. We prefer the term "virtual development" over the term "global software development" as used by Karolak [12], Carmel [3] and Herbsleb et al. [9], because the scope of development is not necessarily restricted to software only, and because the development team need not necessarily be scattered around the globe. Being in another building or on a different floor of the same building, or even at the other end of a corridor, can be sufficient to label it as global development [9].

Both Karolak and Carmel describe the issues that cause virtual development of products to be much more complex than even the most complex project managed entirely in house [3, 12]. They also suggest possible solution strategies, derived from case studies in virtual development projects. The works of Karolak and Carmel are focused on the managerial and collaboration aspects of the organization and execution of virtual development projects. The emphasis is on timely delivery of the product within budget. In this paper, we address the effects of virtual development on product quality, as this is not or only marginally addressed in frequently cited literature on virtual development, e.g. [3, 9, 12]. However, product quality is a complex and multi-faceted concept, pointed out already in 1982 by Garvin [6]. He identified five different views of quality: the *transcendental view,* the *manufacturing view*, the *product view,* the *user's view,* and the *value-for-money view.* In the context of this paper, we will consider the *manufacturing view,* usually encapsulated in the phrase "conformance to specification", as our base view on product quality. Non-conformances to specifications

Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice (STEP'04)
0-7695-2218-1/04 $20.00 © 2004 **IEEE**

(which we shall call *defects*) will typically have a negative impact on product quality whatever the point of view.

Section 2 introduces virtual development and its specific problems. It leads to the recognition of four risk categories that, given the unique aspects of virtual development, are crucial to success or failure of virtual development projects. Section 3 reports on an explorative investigation into the effects of virtual development on product quality. Practical problems to get hard defect data forced an approach relying upon accumulated expert knowledge concerning defect causes in virtual development projects. In "Defect Causal Analysis"-like meetings a team of experts took a lifecycle-centric view on virtual development to address typical causes of the injection and non- or late-detection of defects. Section 4 discusses the suitability of the alternative approach and the findings of the explorative investigation. Finally, section 4 summarizes the conclusions.

2 Problem Areas of Virtual Development

To investigate the effects of virtual development on product quality, the associated problems and risks as reported by Carmel [3] and Karolak [12] can be used as a point of departure. Carmel performed a case study, the Globally Dispersed Software Development (GDSD) study, to find the aspects in which global development differs from traditional, entirely in-house development [3]. The GDSD study concerned 17 software companies engaged in virtual development of products. Eventually, Carmel recognises the following three unique aspects:

Distance between development sites has a direct impact on project control, coordination and communication

Time zone differences between development sites make it even harder to communicate, impacting project control and coordination

Cultural differences between development sites may lead to mistrust, mis-communication and lack of cohesion.

On the basis of these findings, Carmel identified five problem areas that act as “centrifugal forces, driving the global development team apart”. We interpret Carmel’s statement of problem areas that are “driving the global development team apart” as problem areas that potentially threaten the delivery of the product in time, within budget and with the specified or implicitly expected product quality. The problem areas that Carmel recognized are:

(1) geographic dispersion, (2) control and coordination breakdown, (3) loss of communication richness, (4) loss of “teamness” and (5) cultural differences.

We interpret Carmel’s unique aspects as causes and the problems as their effects. In this view, the problem area “Geographic dispersion” seems peculiar, as it is a direct implication of the “distance” aspect. Hammar et al. also came to the same conclusion [8]. These authors replaced the problem area “Geographic dispersion” by “Differences in knowledge”, that they consider to be an effect caused by Carmel’s unique aspects Distance and Culture. However, distance or cultural differences do not necessarily cause differences in knowledge.

Karolak’s work [12] has a lot in common with that of Carmel [3]. However, Karolak uses the word “risk” where Carmel uses the word “problem” (or problem area). We prefer the term “risk”, because it implies that the issue it addresses might be a problem, not that it necessarily is a problem.

Karolak distinguishes three risk categories:

Organizational risks, concerning decision authorities, responsibilities, tasks and project structure, impacting project control, coordination and team behavior,

Technical risks, concerning methods and tools used to solve technical problems, impacting development methodology, architectural choices and eventually product quality,

Communication risks, that may lead to mistrust, misinterpretations and inadequate communications.

Although Karolak and Carmel view virtual or global development from a different perspective, they both arrive at more or less the same risks or problem areas. Nevertheless, to study the effects of virtual development on product quality, we favor Karalok's risk view, because we consider the risk categories more implication-neutral and coherent.

In one aspect Karolak and Carmel differ markedly: where Karolak considers technical aspects as a potential risk, Carmel seems to consider them as a solution. A possible explanation of these seemingly opposed views may be found in a study by Maidantchik et al. [14]. They report the experiences of managing a global development project, in which advanced technology was used to minimize, or even eliminate some of the risks. The collaborating groups that together formed the virtual development team differed in process maturity levels (as measured by CMM). They realized that for low maturity organizations it might be difficult or even impossible to introduce advanced methods and technology. For organizations with a higher level of process maturity, advanced methods and technology can be a solution, while for low maturity organizations the same methods and technology can be a problem. The study by Maidantchik et al. identifies the differences between software processes used by collaborating groups and associated process maturities as a risk category for virtual development projects [14]. Earlier, McMahon already warned for the potential danger of differences in processes and process maturities of collaborating parties

in distributed development [16]. The observations of these authors are in line with our own experiences in virtual development. Consequently, we consider process risks as an additional risk category for virtual development.

Karolak's risk categories apply to both virtual and co-located projects. However, the likelihood of risks occurring in virtual development projects is greater [12], due to the three unique aspects of virtual development as identified by Carmel [3]. This is equally so for the additional *process risk* category.

In first instance, we wanted to investigate in-depth the causes of defects in a number of virtual development projects. However, early in the preparation stage of the investigation it became apparent that such an approach was unfeasible: either the necessary defect data for determining defect root causes is unavailable, or organizations refuse to provide them. A similar observation has been reported by Chulani [5]. Moreover, a quick scan learned that organizations limit the extent of defect classification and analysis to only collect defect data for solving problems at hand; root cause analysis to eventually prevent defects from recurring in the future is hardly practiced. If applied at all, defect classification schemes like the Hewlett Packard Scheme [7], the IEEE 1044 Standard Classification for Software Anomalies [10] or Orthogonal Defect Classification [4] are rarely employed on such a scale that their scope includes the entire virtual development context (i.e. the entire set of related projects contributing to the development of the product).

These practical problems forced us to explore an approach to our investigation of defects, alternative to that of an in-depth causal analysis on a per-project basis. The problem of lack of hard data was also faced by Briand et al., albeit in a different context [1]. They successfully investigated the cost-effectiveness of inspections by relying upon expert judgments. Likewise, our alternative approach is based upon accumulated expert knowledge concerning defect causes in virtual development projects.

3 An Exploratory Investigation of Defect Causes

3.1 Investigation Goals

Apart from the goal of determining the suitability of the approach, this exploratory investigation aims at answering the research questions:

1. *What are typical causes for the injection of defects in virtual development?* These may either be (a) 'new' causes, i.e. additional to causes already present in co-located development, or (b) causes also present in co-located development but with a much higher probability of occurrence in a virtual development context.
2. *What are typical causes for non- or late-detection of defects in virtual development?* I.e. why is it that defects are not detected at all or late?

We use the word *defect* here, as a generic term for any discrepancy between:

- product information specifying the product's behavior and the behavior requested by the product development principal
- actual product behavior and the specification of its behavior
- information intended for the verification & validation of the product and its specified and requested behavior.

3.2 Investigation Approach

Outline. Six analysis meetings, with selected experts as participants, were conducted to identify causes for defect injection and non- or late-detection. The meetings very much resembled the causal analysis meetings as seen in the Defect Causal Analysis (DCA) process [2] and Defect Prevention Process (DPP) [15]. A major difference was the way in which defects to be analyzed were gathered, as shown in figure 1. Instead of selecting a sample from a project's problem database as in regular DCA, an inventory was made of defect types and associated causes on basis of the accumulated experience of the participating experts.

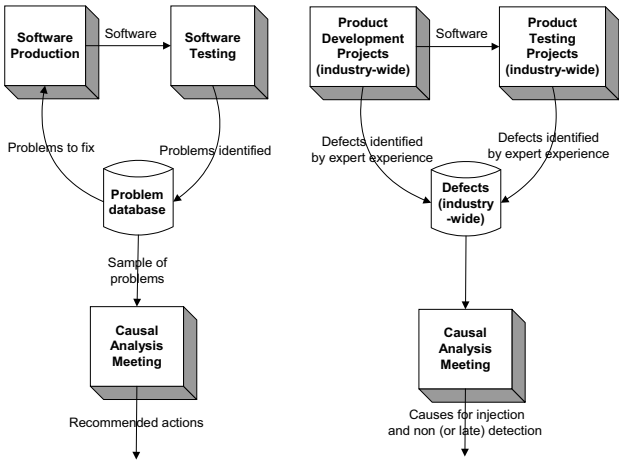

Figure 1. Context of the Causal Analysis Meeting: Standard DCA (left) vs. this investigation (right).

Selection of participants. Participants in the analysis meetings were carefully selected on the basis of their professional background and expertise. Each participant

had to have over three years of experience in the area of virtual development.
They had to be directly involved in product testing, technical product support, defect causal analysis or project management. Of these areas, defect causal analysis experience was a prerequisite for participation. An additional criterion was that they had been directly involved in multiple virtual development projects in the last two years. To ensure a sufficiently wide experience base, we have set the minimum number of experts to five. Actually, six experts participated. The selected experts, all at senior level, included two test managers, one test architect, one software architect, one project manager and one service engineer. They had all acquired experience in multiple organizations, and so correspondingly their accumulated knowledge about defect causes covered multiple organizations.

Moderatorship. A moderator chaired and guided the analysis meetings. His tasks included introducing the meeting participants to the purpose and set-up of the meeting. To safeguard the duration of the meetings, he also intervened in discussions preventing them getting too extensive. At the end of a meeting, the moderator evaluated the meeting.

Meeting and analysis process. In each meeting, a lifecycle-centric view on product creation was taken. A lifecycle-centric view was reported valuable for the investigation of the effects of virtual development on product quality: virtual development projects are typically structured as a hierarchy of lifecycles, reflecting the decomposition of projects into sub-projects [17]. An example of a generic V-lifecycle, deployed in a virtual development setting is given in figure 2.

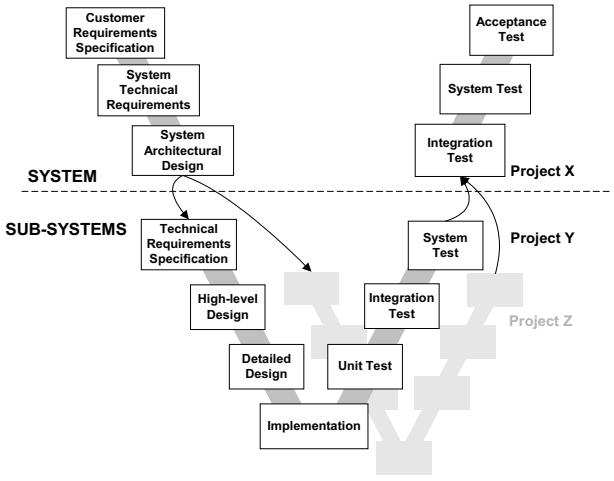

Figure 2. Generic V-lifecycle as deployed in a typical virtual development context.

For each lifecycle phase, the participants discussed about typical defects arising in that specific phase, adversely affecting product quality. Subsequently, possible causes were identified for those defects of which there was a substantiated opinion that their injection (or non-detection) is significantly influenced by the nature of the project (i.e. in a virtual development context). Identification of causes was supported by brainstorming about what possibly could go wrong in the area of communication, process, organization and technology (i.e. the risk categories). Substantiation was to be provided either by statements found in literature on defect detection and prevention or by the participants' own experiences gained from the outcomes of defect causal analyses in earlier projects. By using the cause-effect graphing technique [11], the participants tried to systematically identify all possible causes for injection and non-detection. Causes were assigned to one of the risk categories: communication, process, organization or technology. Only in case of irresolvable doubts, it was allowed to assign the cause to multiple risk categories. To ensure focus and attention from the participants, the duration of the meetings was limited to a maximum of three hours. Six analysis meetings were held, each with a different combination of the six experts. The number of participants in a meeting was deliberately kept small, as to avoid negative group effects like cognitive inertia, dominations and production blocking [18]. Each meeting built upon the results obtained in previous meetings. In this way, the collection of causes was gradually reviewed, refined and inter-subjectively extended. At the end of a meeting, an evaluation was held in which the experts were invited to give their opinion about the analysis process and how they perceived the results.

3.3 Investigation Results

Figure 3 and 4 show examples of the cause-effect graphing performed during the causal analysis meetings. Figure 3 is an analysis example of defect injection showing causes for the injection of specification defects (e.g. wrong, missing, unclear). Figure 4 is an analysis example of non-detection and lists causes for defects not being detected at system testing. For illustrative purposes the cause-effect graphs have been simplified.

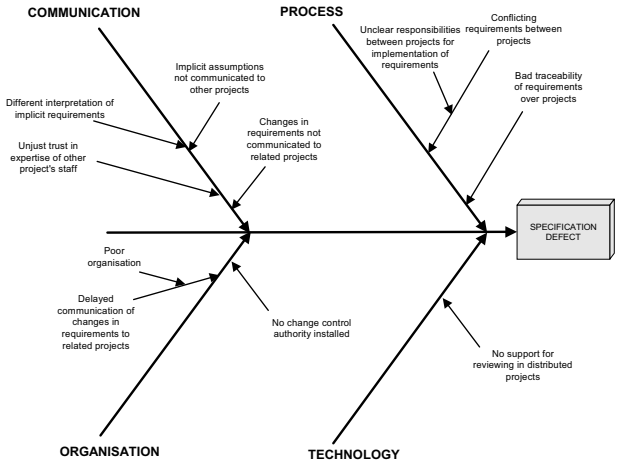

Figure 3. Example of a fish-bone diagram showing causes for injection of specification defects.

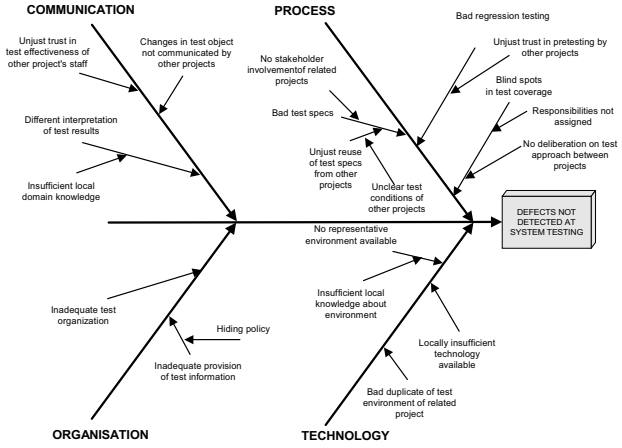

Figure 4. Example of fish-bone diagram showing causes for non- or late-detection at system testing.

A summary of the analysis results for the injection of defects is given in Appendix 1, answering research question 1: *What are typical causes for the injection of defects in virtual development?* The rows represent the various lifecycle phases given in Figure 2, while the columns represent the risk categories. Each table cell contains potential causes for the injection of defects during the corresponding lifecycle phase.

The table given in Appendix 2 has a similar construction as the table given in Appendix 1, but here the cells contain causes for non- or late-detection of defects during the corresponding lifecycle phase. It contains answers to research question 2: *What are typical causes for non- or late detection of defects in virtual development?* Note that the same cause can appear in multiple cells. However, the cause is mostly only mentioned in the cell where it was found to be most significant (i.e. present in practice). Also note explicitly that causes that would be appearing in co-located development as well are left out. Causes like these are only mentioned if their likelihood of occurrence was considered to be higher in a virtual development context than in a co-located development context.

4 Discussion

4.1 Discussion of the approach

The alternative approach yielded tangible information about defect causes in virtual developments. The evaluations, held at the end of each analysis meeting, learned that the participating experts considered the results representative for defect causes in virtual development projects. In later meetings, experts recognized and confirmed the causes that had been identified in previous meetings by different experts, without any exception.

An issue discussed was whether people can retrieve information easily and reliably from long term memory. The prevailing opinion was that the systematic lifecycle-based brainstorming, the extended cause-effect reasoning, the usage of risk categories and the interaction of participants with different viewpoints effectively stimulated the retrieval of long-term memory information. Participants independently expressed their confidence in the completeness of the results. We conclude that determination of defect causes based upon accumulated expert knowledge can be considered as a practical and valid alternative to an in-depth defect causal analysis on a per-project basis.

Future application of this approach may benefit from the following observations:

Participants Involvement. Overall, the participating experts exhibited great interest in the investigation and were highly motivated to take part in the analysis meetings. The opportunity to brainstorm in-depth about the root causes of defects was perceived as a strong motivator: in actual projects there is hardly any possibility to do so, because of time and cost constraints. Some of the experts also mentioned that the discussions with fellow-participants contributed favorably to their understanding of the problems encountered in virtual development. All expressed the interest in receiving a copy of the final investigation report.

Role of the Moderator. Strong moderatorship was needed to ensure focus in the analysis meetings. Participants tended to continue discussing issues deviating from the analysis goals. A recurrent issue

concerned the division of defect-finding responsibilities between developers and testers. Another issue tending to extensive discussions was the question of whether certain defect types can be detected at all in a specific lifecycle phase.

Cause-Effect Graphing. Cause-effect graphing was found to be a useful tool for experienced based root-cause determination. However, cause-effect graphing turned out to be a laborious process, because of the length and inter-relation of the cause-effect chains. A cause can be the effect of another cause (or a combination of other causes). Eventually, a root cause might be a complex interaction of elements pertaining to one or more of the risk categories. This makes the assignment of a cause to one single risk category at least disputable. Either objective discrimination criteria are needed for assigning a cause to a single risk category, or it should be clearly allowed to assign causes to multiple risk categories.

4.2 Discussion of the results

Injection of defects. Previously, Van Moll et al. [17] reported a case study indicating that transitions between lifecycles of sub-projects are particularly sensitive to defect injection. While their study focused on the locations of defect injections, the current study explores the causes of defect injections at the transitions. The data from the current investigation amplifies the finding that the transitions between lifecycles of sub-projects from virtual development projects are defect sensitive. The table in Appendix 1 shows a concentration of causes in the Technical Requirements Specification and Integration Test phase (of system project X). The Technical Requirements Specification shows a relatively high number of causes, and may inherently be more sensitive to injection than other phases. As in this phase, information is being processed that has been transferred from one context (that of Project X) to another context (that of Project Y), the phase is said to be situated at a transition between lifecycles. A relatively high number of potential causes increases the likelihood of defects being injected at such lifecycle transitions.

Defect injection occurring in the Technical Requirements Specification phase can be considered severe as in an actual virtual development project, a given Requirements Specification is a source for derived Requirements Specifications to multiple sub-projects. This means that defects are likely to disperse in a virtual development context.

Regardless of the project context (i.e. virtual or co-located), defects are typically injected in either requirements, design or implementation phases. In virtual development, no new types of defects (i.e. exclusively occurring in virtual development context) are to be expected. Rather, virtual development increases the likelihood of defect injection because new defect causes occur and causes already existing in co-located development are more likely. Consequently, dispersion of defects as well as an increased likelihood of injection may lead to a higher number of defects in the delivered product.

Defect injection is significantly increased in situations where changes in the requirements, design or implementation are being handled. Proper handling includes change control authority, impact analysis and the communication of changes. It was observed that a multitude of causes relates to the handling of changes. An example is that changes are not, or not clearly communicated to the appropriate parties involved or are not unanimously agreed upon. While non-adequate handling of changes is already severe in co-located development, virtual development projects even seem to aggravate the effects, resulting in additional defect injection causes.

Non-detection or late-detection of defects. Causes of non- or late-detection are concentrated in the Integration Test phase of the system-level project (project X). At this lifecycle transition, the components developed in the sub-projects are integrated and subsequently tested. Dispersed defects (especially from the Requirements Phase transitions) will become painfully visible here.

Appendix 2 shows that defects that should have been found earlier, are causing integration difficulties and project delays. Lacking or insufficient test coordination (over the entire virtual development project) seems to be the major cause of non or late-detection of defects. Project delays threaten the execution of a proper integration test. Projects tend to rely upon the subsequent system test as a fall-back, not or insufficiently realizing that certain types of integration related defects cannot be detected at this later phase.

The danger of non- or late-detection of defects especially lurks in situations of unclarity about test coordination. Test coordination includes the action of distributing test focus over the product by the various parties involved, the assigning of test responsibilities and processing of test results. Even in co-located projects, insufficient attention for test coordination and test approach by project management often results in problems with product quality. In virtual development projects the effects of lacking or insufficient test coordination seem to be aggravated, resulting in additional causes for non- or late-detection.

5 Conclusions

In this paper, we have explored defect causes in products developed by virtual teams. Because of the practical difficulties of obtaining hard project-specific

defect data, an approach was taken that relied upon accumulated expert knowledge. This approach was found to be a practical alternative to an in-depth defect causal analysis on a per-project basis.

In causal analysis-like meetings, experts identified causes for defect injection and non- or late detection of defects, considering the individual phases in a hierarchy of related projects constituting a virtual project. Causes were assigned to risk categories Communication, Process, Organization and Technology.

Causes for defect injection were primarily found at the Technical Requirements Specification phases around the transitions from one project to another, early in the lifecycle. Causes for non- or late detection of defects were primarily found at the Integration Test phases situated at the transitions from one project to another, late in the lifecycle.

A main limitation of this study is the relatively small number of participating experts. Furthermore, the results depend on expert judgement, assuming that people can retrieve information reliably from long term memory, and that the influence of negative group effects is negligible. However, we don't have indications that these limitations invalidate the results.

Acknowledgements

We owe many thanks to all experts who participated in our analysis meetings. They sacrificed many hours of their free time, discussing and analyzing defect causes and contributing to the execution of our investigation. We are also indebted to Maggie Larragy (Philips Semiconductors-RTG, Einhoven, The Netherlands) for her helpful comments.

References

[1] Briand, L.C.,Freimut, B., and Vollei, F., Assessing the Cost-Effectiveness of Inspections by Combining Project Data and Expert Opinion, Proceedings of the 11th International Symposium on Software Reliability Engineering, San Jose, 2000, pp. 124-135.

[2] Card, D., Learning from our Mistakes with Defect Causal Analysis, IEEE Software, Jan. 1998, pp. 56-63.

[3] Carmel, E., Global Software Teams: Collaborating Across Borders and Time Zones, Prentice Hall PTR, 1998.

[4] Chillarege, R., Bhandari, I., Chaar, J., Halliday, M. , Moebus, D., Ray, B., and Wong, M.,Orthogonal Defect Classification-A concept for in-process measurements, IEEE Transactions on Software Engineering, vol.18, Nov. 1992, pp. 43-956.

[5] Chulani, S., Bayesian Analysis of Software Cost and Quality Models, PhD Dissertation, University of Southern California, May 1999.

[6] Garvin, D., What does “Product Quality” really mean? Sloan Management Review, Fall 1984, pp. 25-43.

[7] Grady, R.B., Practical Software Metrics for Project Management and Process Improvement, Prentice Hall, 1992.

[8] Hammar, M., Heverius, J., Centrifugal Forces in Global Software Development – Applying the Hammerius Model in the Collaboration between IFS Sweden and IFS Sri Lanka, Master Thesis, University of Linköping, Sweden, 2000.

[9] Herbsleb, J.D., and Moitra, D., Global Software Development, IEEE Software, March/April 2001, pp. 16-20.

[10] IEEE std 1044-1993. IEEE Standard Classification for Software Anomolies, 1993.

[11] Ishikawa, K., Guide to Quality Control, White Plains, N.Y., Quality Resource, 1971, 1989

[12] Karolak, D.W., Global Software Development , IEEE CS, Los Alamitos, CA, USA, 1998.

[13] Lonchamp, J., Collaboration Flow Management: a New Paradigm for Virtual Team Support, DEXA 2002, Aix-en-Provence, 2002.

[14] Maidantchick , C., and da Rocha, A.R.C., Managing a worldwide software process. Proceedings International Workshop on Global Software Development, ICSE 2002, Orlando, Florida, USA, 2002.

[15] Mays, R.G., Jones, C.L., Holloway, G.J., and Studinski, D.P., Experiences with Defect Prevention, IBM Systems Journal, vol.29, no. 1, 1990, pp. 4-32.

[16] McMahon, P.E., Distributed Development: Insights, Challenges, and Solutions, Crosstalk, November 2001, pp. 4-9.

[17] Moll, J.H. van, Jacobs, J.C., Kusters, R.J., and Trienekens, J.J.M., Defect Detection Oriented Lifecycle Modeling in Complex Product Development, Information and Software Technology, vol.46, no. 10, 2004, pp. 665-675

[18] Nunamaker, J.F., Dennis, A.R., Valacich, J.S., Vogel. D.R., George, J.F., Group Support Systems Research: Experience from the Lab and Field, In: L.M. Jessup and J.S. Valacich (Eds), Group Support Systems, New York, MacMillan Publishing Company, 1993.

Appendix 1: Identification of causes for defect injection

Project*	Injection at	Communication	Process	Organization	Technology
X	Customer Requirements Specification				
X	System Technical Requirements			Change control authority over-concentrated in one project. Impact analysis of changes on other projects difficult.	
X	System Architectural Design		No involvement of stakeholders of related projects when creating design		No general agreement on error handling made
Y	Technical Requirements Specification	People have different interpretation of implicit requirements Unjust trust in expertise of other project's staff (e.g. have the experts do the work, despite missing specs and trust on their expertise) Expert sheltering (don't tell us how to do it, we are the experts) Lack of trust in other projects, resulting in information hiding implicit assumptions not communicated to other projects changes in requirements not communicated to related projects	unclear responsibilities between projects for the implementation of requirements Bad traceability of requirements over projects Interaction between product parts not clear (different assumptions made)	Organizational structure delays communication of changes in requirements to related projects No change control authority installed	No support for reviewing in distributed projects
Y	High-level Design	Higher-level (system) design decisions not communicated to related projects Higher-level (system) design decisions not clear	Interaction between product parts not clear (different assumptions made)		
Y	Implementation		Improper base-lining over projects	Usage of different implementation standards by other projects Conventions used by other projects are unclear (e.g. error code ranges)	Incorrect interpretation of test code by other projects No instant access to other projects implementation information (e.g. code, documentation)
X	Integration Test		Additional defects as side-effects of workarounds needed to bypass integration problems Additional defects as side-effects of hasty fixes by integration team because actual	Additional defects as side-effects of duplicate solving of problems by multiple projects (unclear responsibilities for problem solving)	

			problem solvers from projects are not available at time and location of integration		
-	Operational Documentation (e.g. service manual, operating manual, installation manual, user manual)	No contact possible between development and operations: difficult to compile user documentation.	Incorrect reuse of parts of existing documentation. Input from other projects (e.g. development documentation) is unclear Coverage of user documentation for the end product is poor, due to each project defining its documentation in isolation	Responsibility for integral user documentation not defined Insufficient knowledge (domain, product) to create documentation	No environment available assisting in compiling user documentation.

* Refers to the generic lifecycle given in Figure 2.

Appendix 2: Identification of causes for non-detection or late-detection of defects

Project*	Non-detection at	Communication	Process	Organization	Technology
X	Customer Requirements Specification				
X	System Technical Requirements				
X	System Architectural Design				
Y	Technical Requirements Specification		Overall review strategy not established No stakeholder involvement of related projects in review of specifications	Reviewers not trained in reviewing of document hierarchies	
Y	Unit Test				Stubs and drivers to simulate other units based on incorrect assumptions
Y	System Test	Incorrect assumption that certain aspects will be tested by other project	Relevant stakeholders of other projects not involved in reviewing of test specifications		
X	Integration Test	Unreported problems by other projects prevent adequate integration testing. (delay->forced skipping of tests) Deliberate deviations from agreed integration approach not communicated to other projects. Hidden test features in objects not communicated by other projects. Features not used. Problem solvers not present at time and location of integration.	Incorrect assumptions regarding integration plan (delay->forced skipping of tests) Integration approach not defined resulting in delay or order of delivery conflicts. Forced skipping of tests Objects to be integrated not available at planned time. (delay->forced skipping of tests) Stakeholders of other projects not involved in reviewing of test specifications	Responsibility for integration testing not explicitly defined. Previously informal resolution of problems in objects to be integrated (direct contact between developers in different projects) Unclear who is responsible for solving problem encountered. Duplicate solving by various projects->additional defects (delay->forced skipping of tests)	Concealed shutting down of functionality by other projects' objects, unknown to integration tester. Undocumented (incomplete) self tests by other projects' objects. Concealed simulators in objects delivered by other projects, generating output data incorrectly assumed to be all right. Behavior of simulators created by others projects unknown or

		Causes delays->skipping tests.	Solving undocumented last-minute changes in test objects interface (delay->forced skipping of tests) Overemphasis of testing. Testing only those objects experienced as problematic by the integrator at the cost of other objects. Too much focus on 'positive testing' after severe integration problems. Negative scenarios not executed. No coupling of earlier review results to integration test strategy (inadequate testing)	Change control authority over-concentrated in one project. Impact analysis of changes on other projects difficult.	misinterpreted. No general agreement by all projects on design for testability No infrastructure present for integration Integration platform not defined/unclear (delay->forced skipping of tests) Test software not available or unclear. Other integration platform used.
X	System Test	Unjust trust in detection effectiveness of other project's staff Difficulties in understanding test results produced when testing objects created in other projects Not-repaired defects not communicated by other projects. Test results and test object behavior unjustly assumed to be correct.	Blind spots in test coverage. No agreement on test approach is made with other projects. Stakeholders of other projects not involved in reviewing of test specifications. Missing cross-verification of solved defects by other projects Inadequate/no regression testing by other projects while no regression test done here. Unjustified reuse of other projects' test specifications. Specs inadequate. Residual test code incorrectly assumed as real code by other projects.	Responsibility for final system testing not explicitly defined. Testers not having a product focus. Difficult to create adequate test specifications for system parts they are unfamiliar with.	Automated tests not clear/undocumented Test automation tool used in other projects not available No adequate test environment available for objects produced by other projects (e.g. too expensive) Differences in test environments between projects (e.g. in environmental conditions like humidity, temparature) Test conditions of other projects unclear. No regression testing possible. No possibility to remotely monitor tests done in other projects Test input data used by other projects is unclear/unavailable.
X	Acceptance Test	Local differences in usage of product. Functionality by infrequent scenarios not covered.	Wrong stakeholders involved in execution of acceptance test		

* Refers to the generic lifecycle given in Figure 2.