Siemens Dissertation Scholarship of TU Vienna, Faculty of Informatics

Improving Agile Practices with Integrated Quality Assurance Methods

Selected Results from a Family of Empirical Studies

Dipl.-Ing. Dietmar Winkler

Institute of Software Technology and Interactive Systems,
Vienna University of Technology

dietmar.winkler@tuwien.ac.at
http://qse.ifs.tuwien.ac.at/~winkler

Scholarship

Background

- Supported research stays at international research institutions to improve research contacts and emphasis on the internationalization of the individual research work.
- Sponsored research stay up to 4 months to finish the PhD-work.

Focus of my prior research work:

- Software processes (Agile Software Development, V-Modell XT), Agile Practices (e.g., Pair Programming) & Analytical Quality Assurance Methods (e.g., Inspection & Testing).
- Based on this work several papers were published at international conferences on software engineering and empirical software engineering.

Selection of the research organization

- Fraunhofer Institute for Empirical Software Engineering, Kaiserslautern; IESE, head Prof. Dr. D. Rombach; http://www.iese.fhg.de.
- Leading international institute in applied software research and technology transfer (Number 1 institution in Europe and number 5 worldwide) [JSS ranking].
- Major competences of IESE are software engineering, quality assurance, and empirical software engineering.

Special thanks to SIEMENS, TU Vienna, IESE

Table of Contents

- **Introduction**
  - Motivation and Background
  - Related Work
  - Research Approach
- **Best Practice Software Inspection**
  - Family of Experiments to identify Best-Practice Inspection.
  - Results of a series of empirical studies.
- **Bundling Agile Practices and Systematic Quality Assurance Activities**
- **Evaluation of Integrated Pair Programming**
  - Design of the controlled experiment.
  - Evaluation results.
- **Summary and Future Work**

Motivation & Background

Major goal in software and systems Engineering:

- Development of high-quality software products within time, cost and quality constraints to achieve a high level of customer satisfaction.

Challenges and initial situation:

- Increasing complexity of software and systems products.
- Frequent changing customer requirements.
- Need for fast delivery of high-quality (and extended) software products.
- Software Product and Process Improvement (SPPI).
- etc.

Question:

- How can we handle these challenges?

Solution Approaches

These challenges require professional approaches for project planning and execution:

- Software processes help to plan and execute projects systematically.
  - Traditional Software Processes (e.g., V-Modell XT, RUP and Waterfall).
  - Agile Software processes (e.g., SCRUM and eXtreme Programming).
- Constructive methods like agile practices (e.g., Pair Programming) support engineers in constructing software products in an effective and efficient way.
- Analytical methods (e.g., Software inspection) aim at improving software products and enable an assessment of those products.

Nevertheless:

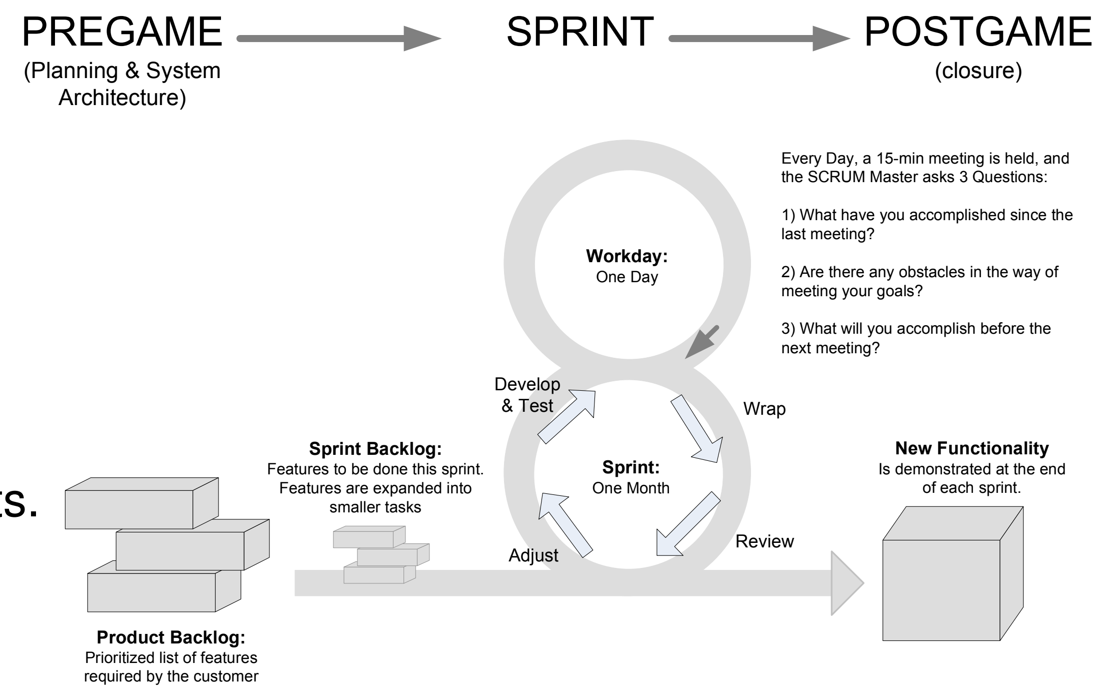
- Constructive and analytical methods provide selective support over the project life-cycle, i.e., construction of individual software products and verification and validation of individual products.
- Bundling constructive and analytical methods can bundle benefits from both disciplines and can lead to synergy effects.
  e.g., Software requirements inspection leads to defect detection lists (derived from inspection approaches) and can be reused for test-case generation on requirements level (e.g., for acceptance testing purposes).

Agile Software Processes: SCRUM

- Agile approaches respond to frequent changing requirements due to a high degree of customer interaction and enable a fast delivery of high-quality software products (builds).
- SCRUM (Schwaber et al., 2007) is an agile software engineering process from project management point of view.

Benefits and contribution:

- Applicability to new software projects and maintenance projects
  → Snapshot of the development process.
- Short iterations (monthly Sprints)
  → Fast delivery of releases.
- Product backlog vs. sprint backlog
  → response to changing requirements.
- Efficient self-organizing teams.
- Established software process in the open source community.



Agile Practices: Pair Programming

- Pair Programming (PP) is an agile practice in eXtreme Programming and Scrum.
- PP involves two roles sharing a common working environment:
  - Driver: implementation role.
  - Observer: supporting role.
  - Roles may change frequently.

Benefits and contribution:

- Increased productivity and product quality.
- Learning in Pairs (e.g., supervisor, introduction of new team members).
- Applicability for other software engineering activities e.g., Pair Reviews, Pair Testing, etc.

Basic references:

- Williams et al., 2000, 2002.
- Cockburn et al, 2001.

Challenges with Pair Programming

- In traditional Pair Programming the observer role performs implicit quality assurance tasks (e.g., continuous reviews).
- This implicit quality assurance is
  - not well defined,
  - not traceable and
  - not repeatable.
- Limitations of Pair Programming application: traditional pair programming is not suitable for environments that need well-defined, traceable and repeatable quality assurance (e.g., security-related application domains).

There is a need for

- Systematic quality assurance activities within a pair programming team.
- Software Inspection is a promising approach for pair programming extension.

Question:

- Which Software Inspection variant is most suitable for this integration purpose?
- How can we introduce a systematic software inspection approach?
- How can we show the benefits?

Software Inspection Variants

- Software Inspection aims at improving software products in early phases of development.
- Early detection and removal of defects, e.g., in the design phase, helps increase software quality and decrease rework effort and cost.

Software Inspection

- is a static analysis technique to verify quality properties of software.
- does not require executable code (applicable to design documents).
- focuses on defect types and location in the inspected object.
- Active guidance of inspectors with reading techniques and guidelines (how to traverse a software document).
- Promises to support learning (structured process which is repeatable and traceable)
- Team meetings vs. Nominal teams.

Winkler, 2008: Improvement of Defect Detection with Software Inspection Variants:
A Large-Scale Empirical Study on Reading Techniques and Experience, VDM Verlag, 2008.

References: Fagan 1976, Gilb 2000, Biffl 2001, Winkler, 2008.

Reading Techniques

- Reading Techniques aim at supporting inspectors during the inspection process by providing guidelines for systematic reading.

- Various selected reading technique variants
  - Ad-hoc: no guidance
  - Checklist based reading: sequential reading according to domain/project specific checklist-items.
  - Scenario Based reading: scenarios describe workflows from different perspectives, e.g., designer, tester, and user, by providing a sequence of steps to address individual business cases.
  - Usage based reading: use cases define individual business cases on requirements level (based on UML). Use cases can be the basis for a model-driven approach.

- Guidance might help observers in systematically support the driver in developing new pieces of software (enabling traceability and repeatability).
- Which inspection / reading technique variant might be most valuable in a given context?

Research Approach

- Step 0: Systematic Literature Review on Pair Programming and Software Inspection.
- Step 1: Identifying Best-Practice Software Inspection in a given context by conducting a family of experiments.
- Step 2: Construction of an "Integrated Pair Programming Approach" (IPP).
- Step 3: Evaluation of IPP in a given context to show its impact on quality assurance metrics, e.g., defect detection capability.

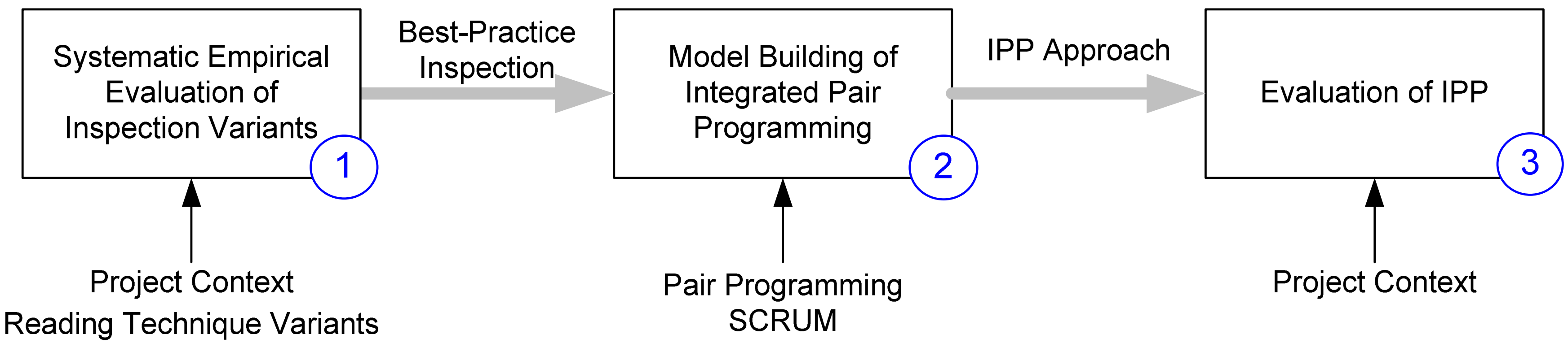


Table of Contents

- **Introduction**
  - Motivation and Background
  - Related Work
  - Research Approach
- **Best Practice Software Inspection**
  - Family of Experiments to identify Best-Practice Inspection.
  - Results of a series of empirical studies.
- **Bundling Agile Practices and Systematic Quality Assurance Activities**
- **Evaluation of Integrated Pair Programming**
  - Design of the controlled experiment.
  - Evaluation results.
- **Summary and Future Work**

Family of Inspection Experiments

- General Goal: Identification of a Best-Practice Inspection Variant in a given context.
- Quality Attribute / Metric: Defect detection capability (effectiveness and efficiency) of different reading technique approaches.

- 3 Large-Scale Empirical Studies (Controlled Experiments) in Academic Environment
- > 160 student participants each; inspection duration appx. 3 hours.

- Provided Material:
  - Requirements Specification,
  - USE case models,
  - Individual guidelines for defect detection tasks.
  - Supporting material (e.g., questionnaires) & online data capturing tools.
- Two different applications in the area of administrative software systems:
  (a) Ticket selling system and (b) Taxi management system.
- Variation points:
  - Defect types and defect severity classes.
  - Document locations (business case descriptions, architecture and design, code).

Selected Results: CBR vs. SBR variants

- Focus: Checklist-based reading technique (CBR) vs. Scenario-Based reading techniques from different perspectives (SBR-Designer, SBR-Tester, SBR-User)

- Main results:
  - Scenarios and perspectives support defect detection in related document parts (e.g., SBR-U identifies most defects in the Business Case Description and SBR-D was most effective in the architecture and design part).
  - Lower qualified inspectors are more effective and efficient using the scenarios and perspectives.
  - Different reading techniques: CBR is useful for less important defects; perspectives and scenarios spot on more important and critical defects.
  - SBR inspectors are more efficient (need on average less time for inspection) because of the active guidance of the reading technique approach.

- Next Step: Improving SBR with focus on Use Cases => UBR.

- Publication: D. Winkler: „Improvement of Defect Detection with Software Inspection Variants: A Large-Scale Empirical Study on Reading Techniques and Experience", VDM Verlag, ISBN: 3836470136 , 2008.

Selected Results: CBR variants vs. UBR

- Focus: Active guidance of inspector regarding defect detection performance.
- Checklist-based RT variants (CBR) vs. Usage-Based reading techniques.
  - CBR-Variants:
    - Generic checklist (CBR-gc): pre-defined set of checklist items.
    - Tailored checklist (CBR-tc): tailoring of requirements according to individual and subjective importance (from reviewers point of view).
  - UBR: Expert prioritization of Use-Cases.
- Main: Results
  - UBR performance is best for critical and important defects (significant differences)
  - Effectiveness & Efficiency: UBR > CBR-tc > CBR-gc.
  - Active guidance support inspection proceeding (UBR and CBR-tc).
  - UBR expert know-how has significant effects on defect detection rates.
- Next Step: Investigating UBR variants (reduction of preparation effort).
- Publication: D. Winkler, S. Biffl, B. Thurnher: „Investigating the Impact of Active Guidance on Design Inspection", 6th International Conference on Product Focused Software Process Improvement (PROFES), Oulu, Finland, June 2005.

Results: UBR variants vs. CBR

- Focus: Impact of expert ranked Use Cases on defect detection performance.
- Checklist-based RT variants (CBR) vs. Usage-Based reading techniques.
  - UBR-Variants:
    - Usage based reading (UBR): expert prioritized use cases.
    - UBR with individual use case prioritization (UBR-ir).
  - CBR: stepwise application of a context-specific checklist.
- Main: Results
  - UBR performance (with expert ranking) is best for all defect severity classes.
  - The performance advantage of UBR is greatest critical defects.
  - Effectiveness: UBR > UBR-ir > CBR.
  - Efficiency: UBR = UBR-ir > CBR.
  - UBR expert know-how has significant effects on defect detection rates.
- UBR with expert ranking turned out to be the most effective and efficient approach for defect detection => Candidate for integration in Pair Programming.
- Publication: D. Winkler, M. Halling, S. Biffl: „Investigating the Effect of Expert Ranking of Use Cases for Design Inspection“, Proceeding 30th IEEE Euromicro Conference, Rennes, France, September 2004.

Table of Contents

Bundling Benefits …

Best-Practice Software Inspection

- Applicable in all phases of the Software Life-Cycle.
- Systematic quality assurance activity.
- UBR is a well-investigated reading technique approach.
- Focus on critical defects first.
- Active guidance through guidelines and prioritized use-cases.
- Application of use cases and scenarios from requirements documents in a pre-defined order (prioritized by a group of experts) to design documents.

Pair Programming

- Flexible and agile constructive practice.
- Embedded within an agile software development process.
- Applicable for development and maintenance projects.
- Pair Learning.
- Team activity (driver & observer)
- Including implicit quality assurance activities (need for traceability and repeatability).
- Test-Driven Development approach.
- Defect detection in early products as by-product of code construction.

Integrated Pair Programming (IPP)

Expected Benefits:

- Flexible (agile) software construction including systematic product quality improvement.
- Defect detection (Best-Practice Inspection) based on requirements and code.
- Enhanced learning effects.
- Systematic and traceable quality assurance activities.
- Enhanced tasks and responsibility for the observer role.
- Application of prioritized use cases according to business value contribution.

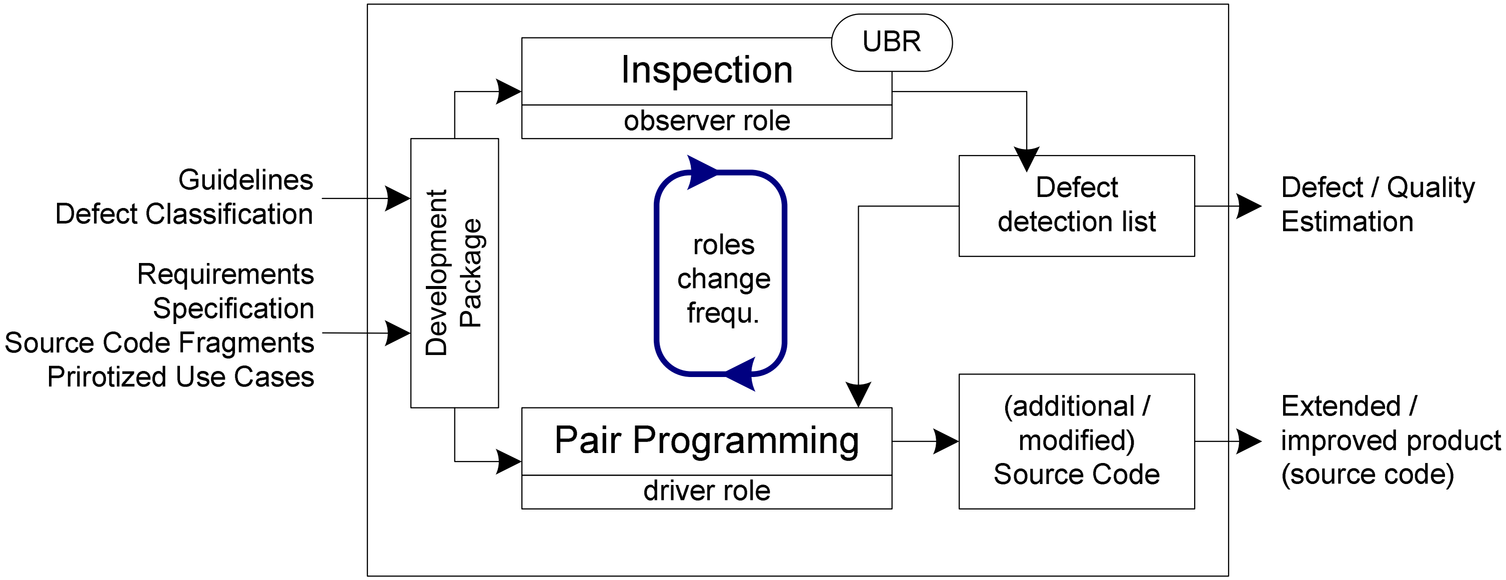


Table of Contents

Design of the Controlled Experiment

- An experiment to investigate defect detection capability of best-practice inspection and an integrated pair programming approach.
- Experiment process in 5 basic steps:
  - (a) Participant selection, (b) experience collection, (c) experiment preparation for participants, (d) study execution in two sessions including feedback after every session, and (e) data submission.
- Study material:
  - Textual requirements, prioritized use cases, source code fragments (partially implemented), guidelines, experience and feedback questionnaires.
- Expert seeded defects:
  - 60 reference defect spread over different document locations (different defect severity classes and types).
  - 29 critical, 24 important, 7 less important defects seeded in the design specification and source code.
- 41 subjects (experiment participants): graduate students in a class on quality assurance and software engineering (15 UBR, 26 pair programmers, i.e., 13 pairs).

Systems Overview: Taxi Management System

- System Overview
  - Maintenance / evolution process for a commercial application.
  - Taxi management system in two session (Central, Taxi).

- Software Artifacts
  - Textual requirements: 8 pages, 2 component diagrams.
  - Design document: 8 pages, 2 component diagrams and 2 UML charts.
  - Use case document: 24 use cases and 23 sequence diagrams.
  - Source code: some 1,400 LoC, 9-page description.
  - Guidelines and questionnaires.

Research Questions

- General idea: Integrating inspection in PP leads to more structured defect detection approaches, improves overall defect detection capability, and software product quality.

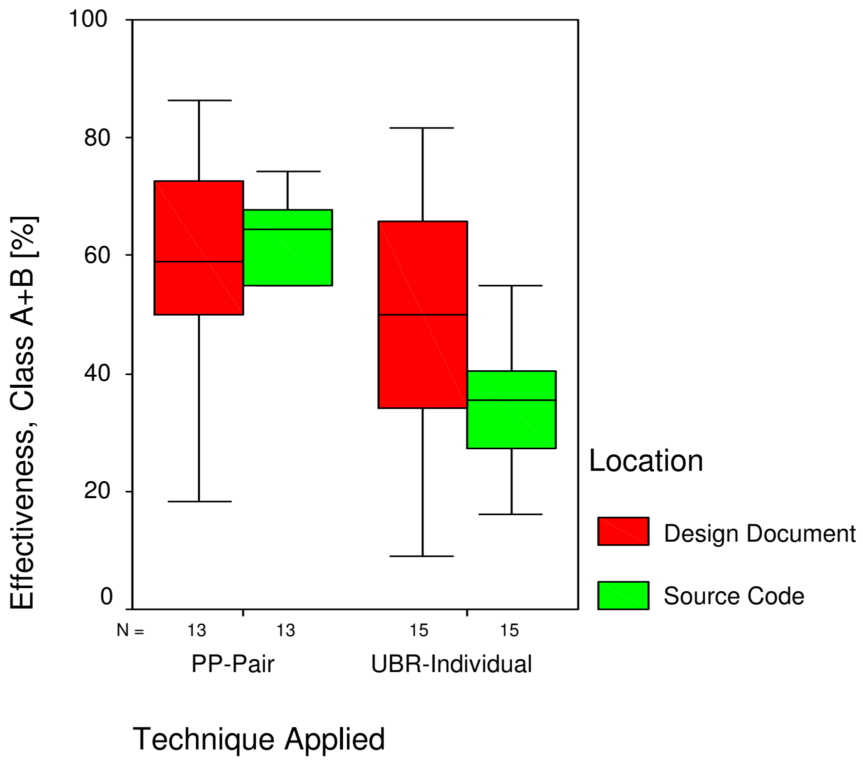
1. Hypotheses for natural work units (individual inspectors vs. pairs)
   - H1.1: Effectiveness (PP) > Effectiveness (UBR): source code documents
   - H1.2: Effectiveness (PP) < Effectiveness (UBR): natural-language text documents.
   - Note: higher overall effort applying PP, because of different “team size” (2 persons) and focus on code construction (defect detection as a by-product).

2. Similar hypothesis for “minimal teams” (2-person inspection teams vs. pairs).

3. Performance of nominal teams:
   Do mixed teams perform better than “best-practice” teams?

Results: Effectiveness of Working Units

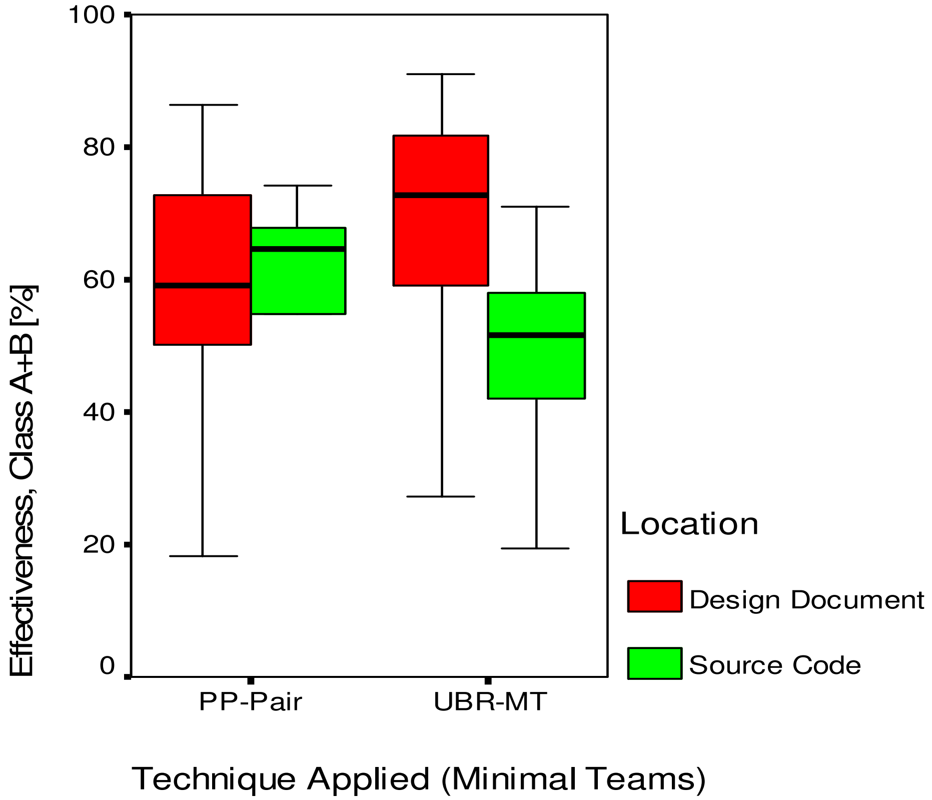
- Effectiveness is the number of defects found defects in relation to the number of seeded defects.
- Focus on important defects (risk A+B) and document location (design document, source code).
- Effectiveness (PP) > Effectiveness (UBR) for all defect severity classes and document locations.
- Significant differences for
  - Source Code and
  - Design Document & Source Code.
- No significant differences for
  - Design Document.



- ➲ The integrated PP approach outperforms inspection according to source code defects.
- ➲ Smaller differences for design documents but still advantages for PP.

Results: Effectiveness of "Minimal Teams"

- Comparability in team size → minimal teams.
  - Pair: 2 persons (original work unit).
  - UBR-MT: nominal 2-person team of individual inspectors (randomly assigned)
- Focus on important defects (risk A+B) and document location (design document, design source code).
- Significant differences for
  - Source Code.
- No significant differences for
  - Design Document and
  - Design Document & Source Code.

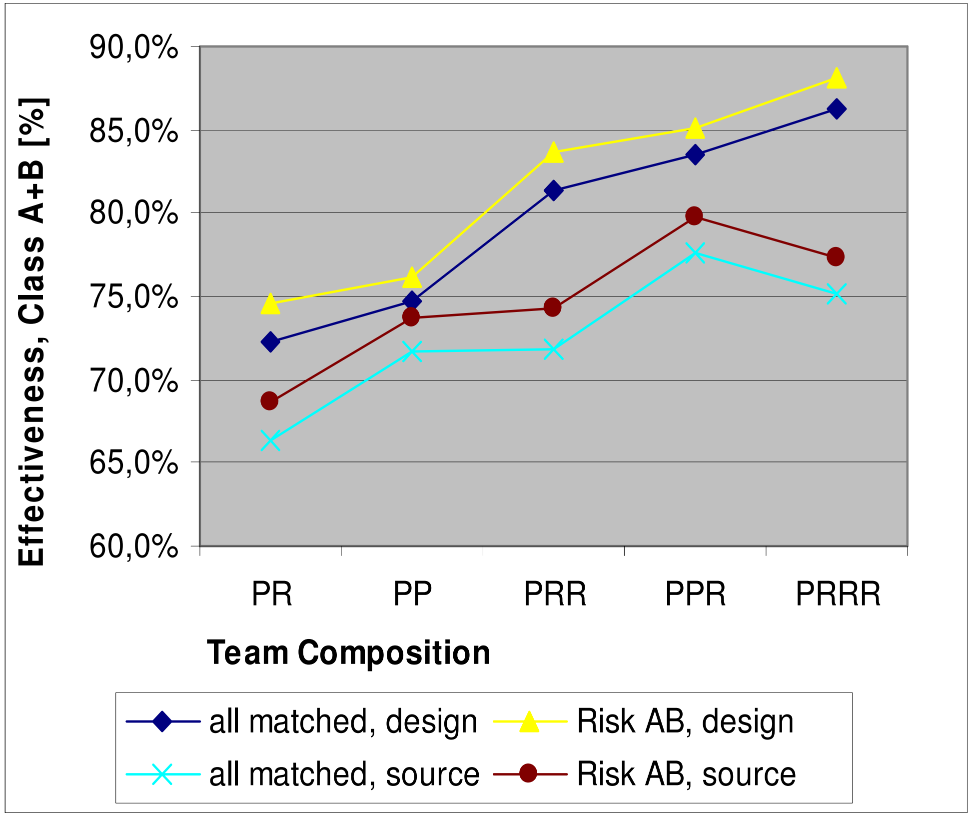


➲ PP outperforms effectiveness acc. to source code defects.

➲ Advantages for UBR-MT according to design document defects.

Results: Team Composition

- Inspection and Pair Programming focuses on different defect types and defect locations.
- Thus, we expect an improved performance of mixed teams due to synergy effects.
- A "nominal team" is a collaboration of two or more members without interaction.
- Team building: continuous increase of effectiveness for up to 4 team members.
- Increasing effectiveness for design documents (smaller gain including additional pairs).
- Increasing effectiveness for source code including additional pairs and an almost constant value on inspector integration up to 4 team members.
- PRRR: decreasing effectiveness acc. to source code defects (additional inspectors seems to hinder source code quality)



Legend:
P … Pair Programming Team (2 persons); R .. Individual Reviewer;
e.g., PRR: 1 Pair Programming Team and 2 Reviewers

Table of Contents

- **Introduction**
  - Motivation and Background
  - Related Work
  - Research Approach
- **Best Practice Software Inspection**
  - Family of Experiments to identify Best-Practice Inspection.
  - Results of a series of empirical studies.
- **Bundling Agile Practices and Systematic Quality Assurance Activities**
- **Evaluation of Integrated Pair Programming**
  - Design of the controlled experiment.
  - Evaluation results.
- **Summary and Future Work**

Summary

- Software Inspection is an analytical quality assurance technique for early defect detection tasks in development projects. Reading techniques support inspectors by providing guidance for inspection activities.

- Agile processes (e.g. Scrum) aim at providing flexibility to frequent changing requirements and fast delivery of software products.
  Agile practices (e.g., Pair Programming) is a team activity involving two roles: a driver and observer. The observer performs implicit quality assurance tasks.

- Nevertheless, observer activities are not traceable, not auditable and not repeatable => need for systematic support of pair programming teams.

- UBR inspection turned out to be the most effective and efficient systematic quality assurance activities in the area of software inspection.

- Integrated pair programming is a valuable approach for improvement software quality (increased productivity and product quality by means of defect detection capability)

Practical Relevance & Future Work

Practical Relevance

- Results of series of experiments can provide a decision support for method selection and application in industry context.
- Benefits from integrating methods and processes from different disciplines.
- An idea for a for a systematic improvement and evaluation of various methods, e.g., software inspection variants.

Future work

- A more detailed investigation of the IPP approach with focus on various aspect of quality assurance (e.g., productivity, quality of new software code, team performances and individual qualification).
- Elaboration on the generalization of pair activities (e.g., pair reviews, pair testing, pair design and architecture evaluation).
- Investigation of the applicability of the method in various domains and industry context to enhance the validity of the results.
- Systematic quality assurance strategy evaluation is an follow-up project with Fraunhofer IESE .

Thank you …

Improving Agile Practices with Integrated Quality Assurance Methods

Selected Results from a Family of Empirical Studies

Dipl.-Ing. Dietmar Winkler
Vienna University of Technology,
Institute of Software Technology and Interactive Systems
Favoritenstr. 11/188, A-1040 Vienna, Austria

http://qse.ifs.tuwien.ac.at/~winkler
dietmar.winkler@qse.ifs.tuwien.ac.at

Selected Literature References

- S. Biffl: *Software Inspection Techniques to support Project and Quality Management*, Shaker Verlag, 2001.
- A. Cockburn, L. Williams: the Cost and Benefits of Pair Programming, Extreme Programming Examined, 2001.
- M.E. Fagan: *Design and Code inspections to reduce errors in program development*, IBM Systems Journal, Vol. 15, No 3, 1976.
- T. Gilb, D. Graham: *Software Inspection*; Addison Wesley, 1993.
- K. Schwaber, T. Irlbeck: *Agiles Projektmanagement mit Scrum*, Microsoft Press, 2007.
- L.Williams, R.R. Kessler, W. Cunningham, R. Jeffries: *Strengthening the Case for Pair Programming*, IEEE Software, 2002.
- L. Williams, R.R. Kessler: „All I really need to know about pair programming I learned in Kindergarten", Communications of the ACM, 2000.
- D. Winkler: "Improvement of Defect Detection with Software Inspection Variants: A Large-Scale Empirical Study on Reading Techniques and Experience", VDM Verlag, 2008.

Selected Web References

Selected Literature References (continued)

- D. Winkler, S. Biffl, B. Thurnher: „Investigating the Impact of Active Guidance on Design Inspection", 6th International Conference on Product Focused Software Process Improvement (PROFES), Oulu, Finland, June 2005.
- D. Winkler, M. Halling, S. Biffl: „Investigating the Effect of Expert Ranking of Use Cases for Design Inspection", Proceeding 30th IEEE Euromicro Conference, Rennes, France, September 2004.

Selected Web References

- S. Biffl, Dietmar W., D. Frast: „Qualitätssicherung, Qualitätsmanagement und Testen in der Softwareentwicklung", Skriptum zur Lehrveranstaltung, 2004. http://qse.ifs.tuwien.ac.at/courses/skriptum/script.htm
- Software Engineering Body of Knowledge, http://www.swebok.org, 2004.
- Software Engineering – Best practices: http://best-practice-software-engineering.blogspot.com/
- Software Engineering – Best practices: http://best-practice-software-engineering.ifs.tuwien.ac.at/