Mutual Programming
A Practice to Improve Software Development Productivity

Status:	Approved
Author(s):	Gerold Keefer; Hanna Lubecka
Version:	1.1
Last change:	09.08.02 11:24
Project phase:	Implementation
Document file name:	MutualProgramming.doc
Approval Authority:	Gerold Keefer
Distribution:	Public
Security Classification:	-
Number of pages:	18

Notes:

1. Picture on title page courtesy Robert Peters from the movie “Mutual Love Life”, 1999.
2. Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office.
3. CMMI, CMM Integration, PSP, and TSP are service marks of Carnegie Mellon University.

1 Introduction

There have been several proposals for processes and practices at the personal and at the team level that aim to increase predictability, productivity, and quality of software engineering tasks. The most prominent are

- The Personal Software Process and the related Team Software Process (PSP/TSP), and
- Extreme Programming and its practice Pair Programming (XP/PP).

We will briefly introduce both approaches in the following sections.

1.1 PSP/TSP

If you have no basic idea about the PSP and TSP processes, please have a look at [5, 6].
There is sufficiently strong empirical evidence based on industry data for significant increases in both schedule reliability and software quality as a result of the application of PSP/TSP [1]. This can be seen from Figure 1.

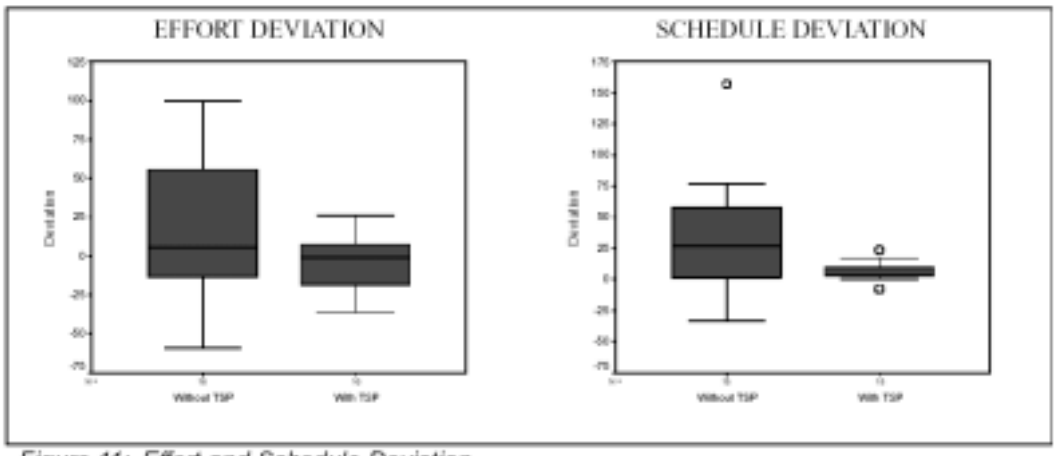


Figure 11: Effort and Schedule Deviation

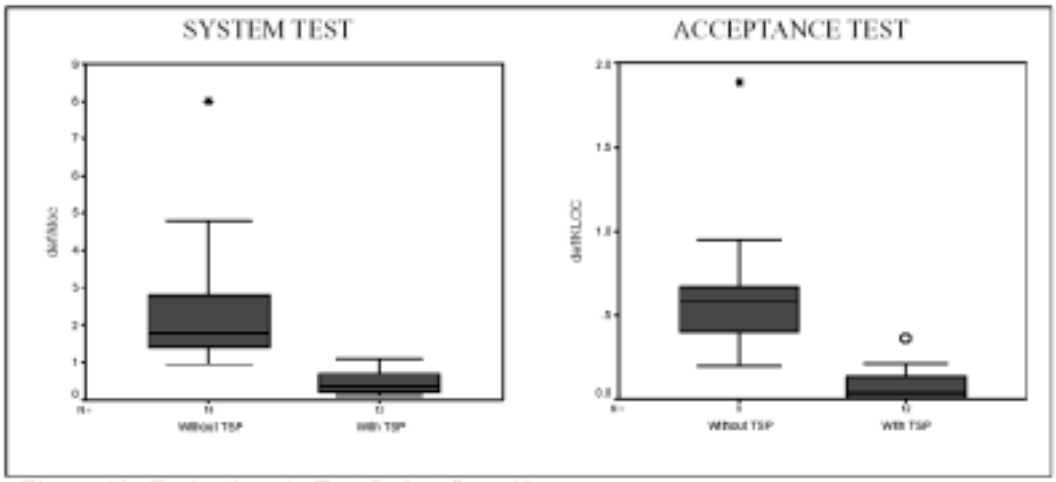


Figure 12: Reductions in Test Defect Densities

Figure 1: Increased schedule reliability and software quality through PSP/TSP use (from [1]).

In general the observed variability of work products created with PSP/TSP is reported to be lower than without PSP/TSP. In controlled experiments with students, however, no increases or even decreases in quality were observed [12].
A very important fact concerning the PSP/TSP is that close to all reports indicate problems with the adoption of the practices. In fact the TSP was developed for the reason that the PSP was not viable in the field. In [16] Humphrey and Webb state:

"Although the training was generally well received, use of the PSP in TIS started to decline as soon as the classes were completed. Soon, none of the engineers who had been instructed in PSP techniques was using them on the job."

The introduction of the TSP may have positively influenced the adoption of the PSP, however, despite those benefits neither PSP nor TSP are a widespread industry practice. The major obstacles are most likely

- considerable necessary effort to learn the PSP/TSP and
- high probability that the PSP/TSP will not be used after a training, because it is considered to be too restrictive and cumbersome.

1.2 XP/PP

Extreme Programming is currently probably the most controversial software development methodology topic. We have expressed considerable concerns about the usefulness of this method in [7]. Despite those concerns there is no denying that XP/PP receives continuing high focus and various organizations implemented at least parts of the approach.

A central element of Extreme Programming is the practice of pair programming. Laurie Williams, who conducted an empirical study on pair programming, describe the practice as follows:

"Two programmers working side-by-side, collaborating on the same design, algorithm, code or test. One programmer, the driver, has control of the keyboard/mouse and actively implements the program. The other programmer, the observer, continuously observes the work of the driver to identify tactical (syntactic, spelling, etc.) defects and also thinks strategically about the direction of the work. On demand, the two programmers can brainstorm any challenging problem. Because the two programmers periodically switch roles, they work together as equals to develop software."

Although there are numerous claims about productivity and quality improvements with pair programming, the empirical evidence we have so far does not consistently support such claims:
In fact, all of the empirical studies suggest that the total development time increases and improvements in quality are not consistently reported [10].
In Figure 2 you will find results of the most recent study on pair programming that also included the PSP in the experiment [10]. The results indicate that the consumed person hours nearly double with pair programming and that program quality measured by number of re-submissions is only marginal better with pair programming.

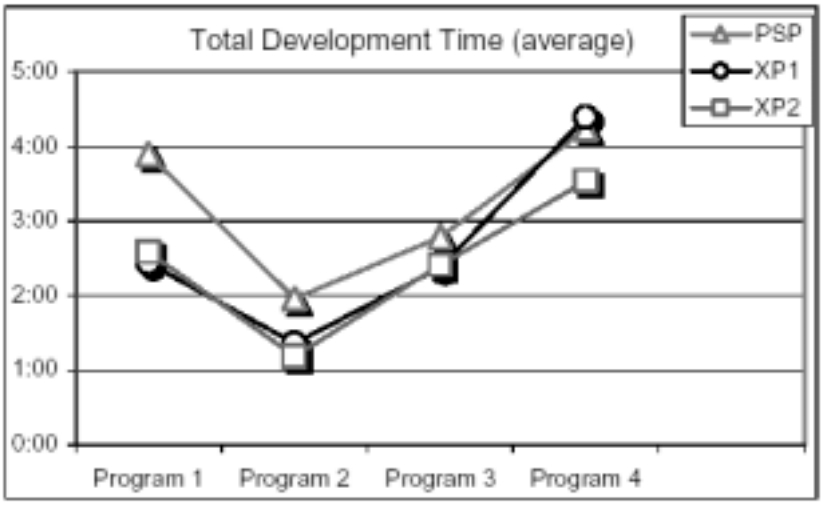


Figure 2. Comparison of average values of total development time.

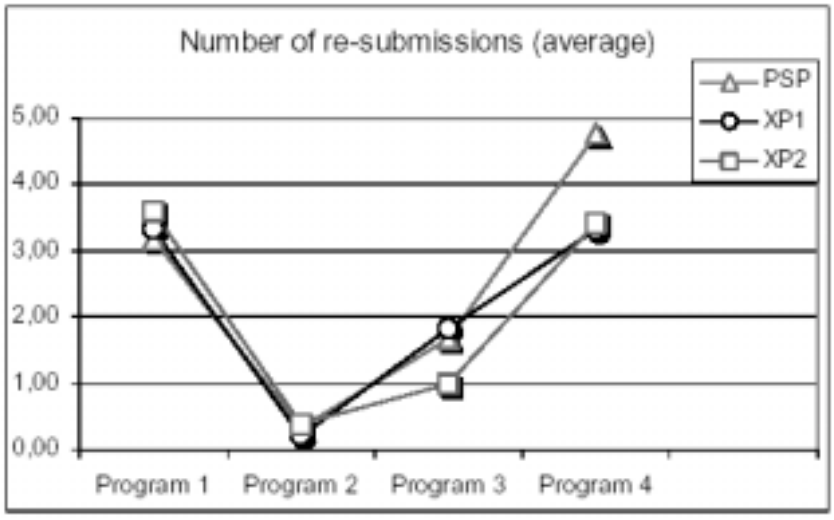


Figure 7. The number of re-submissions of corrected programs.

Figure 2: Results of the Nawrocki study on pair programming (XP2), solo programming/test-centred (XP1), and PSP (PSP) from [10].

Another interesting result of this study is that the PSP and PP are both lacking behind a solo programming technique called XP1 that the authors describe as "*Experimentation and test-centred quality assurance*, *simple solution*, and *risk minimization".*

The ease of adoption of XP/PP is very much dependent on the prevailing development culture in the respective organization. Extreme Programming has drawn a lot of interest in very short time. However, one of the most controversial practices is the practice of pair programming. It also follows from common sense that not everybody will easily transition to a situation that requires working several hours a day at the same machine together with a co-worker.

1.3 Comparison of PSP/TSP and XP/PP

PSP/TSP:

Pros	Cons
Quality Increase	Adoption problems
Productivity Increase	Overhead
Reliability Increase	
Quantitative/Measurable	
Peer pressure	
Formal Inspections	
Knowledge Transfer	
Partly Suitable for distributed development	

XP/PP:

Pros	Cons
Knowledge Transfer	Productivity Decrease
Defect Prevention	Not suitable for distributed development.
Peer Pressure	Faster exhaustion.
No overhead	No formal inspections.

2 Mutual Programming (MP)

2.1 Purpose

MP is a software development practice targeted at the individual and small team level that aims to increases productivity, quality, and reliability of software engineering work. It is designed to be used primarily during the implementation phase of software development projects where the majority of effort is spent and most defects are injected.

2.2 Scope

MP does not say anything about organizational or project management issues beyond the level of individuals, pairs, and supporting staff like coaches and moderators. MP is not a software development process that an organization can solely rely on and it does not describe contents of work products and specific tasks, beyond what is necessary to implement the practice. No element of MP is entirely new. MP is a set of existing best practices put together in a transparent, measurable, and appealing way.

2.3 Constraints

MP may not be suitable for environments with highly fragmented tasks (e.g. many small tasks shorter than half a day), such as system administration environments or product support. So far the benefits of MP are entirely hypothetical and not supported by empirical findings. However, close to all of the elements of MP, such as DLDA or automated unit testing, have been shown to be beneficial when applied in isolation.

3 MP Requirements

1. MP shall be easy to adopt and sustain.
2. MP shall allow for the easy measurement of performance with regard to quality, productivity, and estimation reliability.
3. MP shall include defect prevention measures.
4. MP shall increase quality, productivity, and estimation reliability.
5. MP shall be suitable for distributed development settings.
6. MP shall be suitable for development of work products other than code.
7. MP shall be suitable for organizations of any size.

4 Principles

4.1 Levelled approach with stepwise introduction

The experience of the PSP/TSP is that rigorous and formal practices are not easily adopted. MP facilitates adoption with a levelled approach that carries developers from an initial “green belt”-level through “blue belt” and “red belt” to the “black belt”-level.

Black Belt (2 Developers, Coach)
Mandatory Coaching
Effort Tracking
3 Point Estimation Method
Formal Work Product Risk and Opportunity Assessment
Inspections
Automated Unit Testing
DLDA
Red Belt (2 Developers. Moderator)
Effort Tracking
3 Point Estimation Method
Work Product Previews
Fagan Inspections
Automated Unit Testing with Test Coverage Measurement
DLDA
Blue Belt (2 Developers)
Work Product Previews
Automated Unit Testing/Inspections
DLDA
Green Belt (1 Developer)
DLDA

We reckon that it should be possible to advance from one stage to the next within 2-5 months.

4.2 Open for measurement and comparison

Clear entry and exit criteria as well as mandatory measurements support the quantitative evaluation of the practice.

4.3 Applicable in close to all development environments

MP can be done in a range of settings, including remote settings.

4.4 Flexible and Adaptable

With MP collaborative work is in no way prohibited. The important point is that the ultimate responsibilities (development vs. QA) are clearly separated.

For explicit prototyping to investigate issues discovered during previews or assessments it is allowed to bypass elements of MP, however, any official work product of a project has to be created according to the required practices.

5 Roles

5.1 Developer

The person that executes the implementation of the respective work product and is responsible for how this implementation is conducted. The developer is allowed to support the QA, however, the responsibility is separated.

5.2 QA (Quality Assurer)

The person that previews, inspects, and tests the work product. The QA is allowed to support the Developer, however, the responsibility is separated.

5.3 Moderator

The person that moderates the inspection meetings. The role is precisely described in Fagan's paper [3]. In remote settings the inspection meetings are substituted by inspection conference calls.

5.4 Coach

The person that is experienced with MP and has practiced it at the black belt level. The coach leads the formal work product assessments and is the approval authority for release.

6 Implementation

6.1 Green Belt

Team Setting

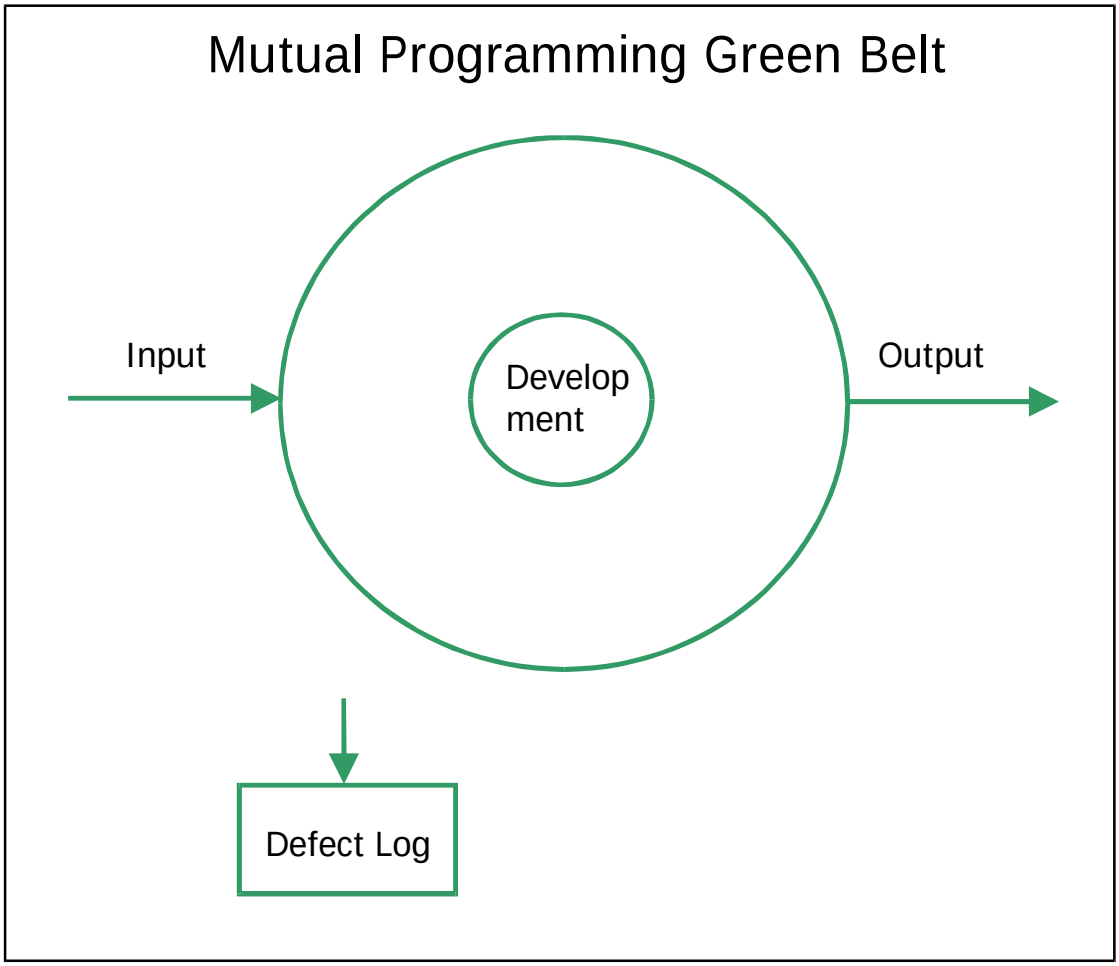
- 1 developer, doing both development and QA for assigned work products.

Required Elements

- Defect logging and defect data analysis (DLDA) of personal defects in accordance with the standard to be found at http://www.ipd.uka.de/PSP/Dokumente/DefTyp/DefTyp.html and [13].
- Ask the three questions to be found at http://www.multicians.org/thvv/threeq.html .

Recommended Elements

- Statistical analysis of defect log every three months.



6.2 Blue Belt

Team Setting

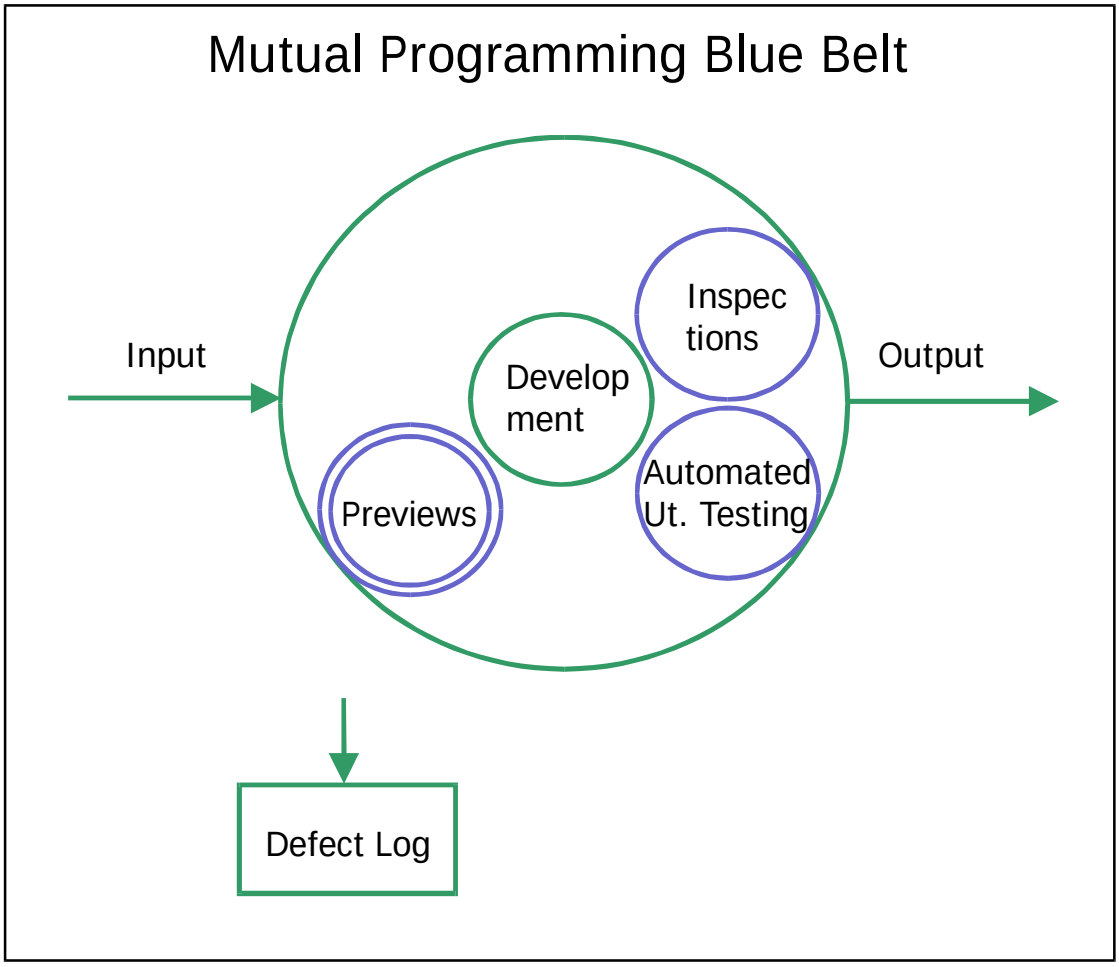
- 2 developers, one doing actual development and the other doing QA and vice versa.
- Optional Moderator for inspections.

Required Elements

- All required and recommended elements at the white belt level.
- Work product previews: Before a work product is created a formal meeting is held, similar to what is described as “Overview” in [3]. The exit criterion of that meeting is a document stating the purpose, scope, and constraints of the work product and any concerns identified.
- Explicit QA of the work product: In the case of an executable work product this is an automated unit test suite implemented by the QA responsible. In the case of a design document this is a formal inspection conducted by the QA responsible.
- The test cases of the unit test suite have to be specified before the implementation of the executable work product begins.
- The work product is released to the project as soon as either all specified test cases are passed or all issues found during the inspection are resolved. Release must be approved by the QA responsible.

Recommended Elements

- Test coverage target and test coverage measurement.
- Effort/time tracking, for example generated document pages or LOCs.



6.3 Red Belt

Team Setting

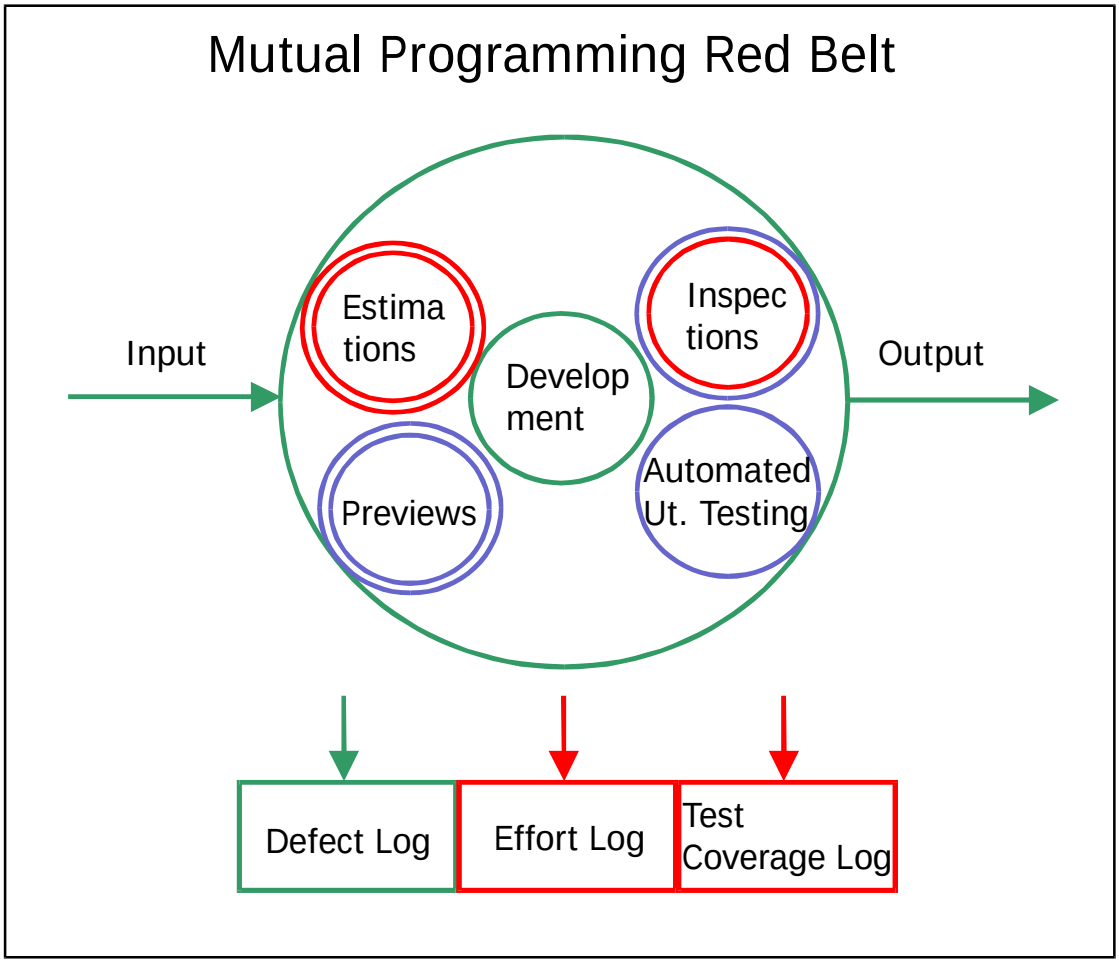
- 2 developers, one doing actual development and the other doing QA.
- Moderator for inspections.

Required Elements

- All required and recommended elements at the blue belt level.
- The two developers know about the defect profiles and the defect logs of each other and mutually ask the three questions to be found at http://www.multicians.org/thvv/threeq.html .
- Fagan inspections for all work products.
- Application of formal effort estimation such as the 3-point effort estimation method independently done by both developers [4].

Recommended Elements

- Program trace files, if applicable.
- Tracking of defect density.



6.4 Black Belt

Team Setting

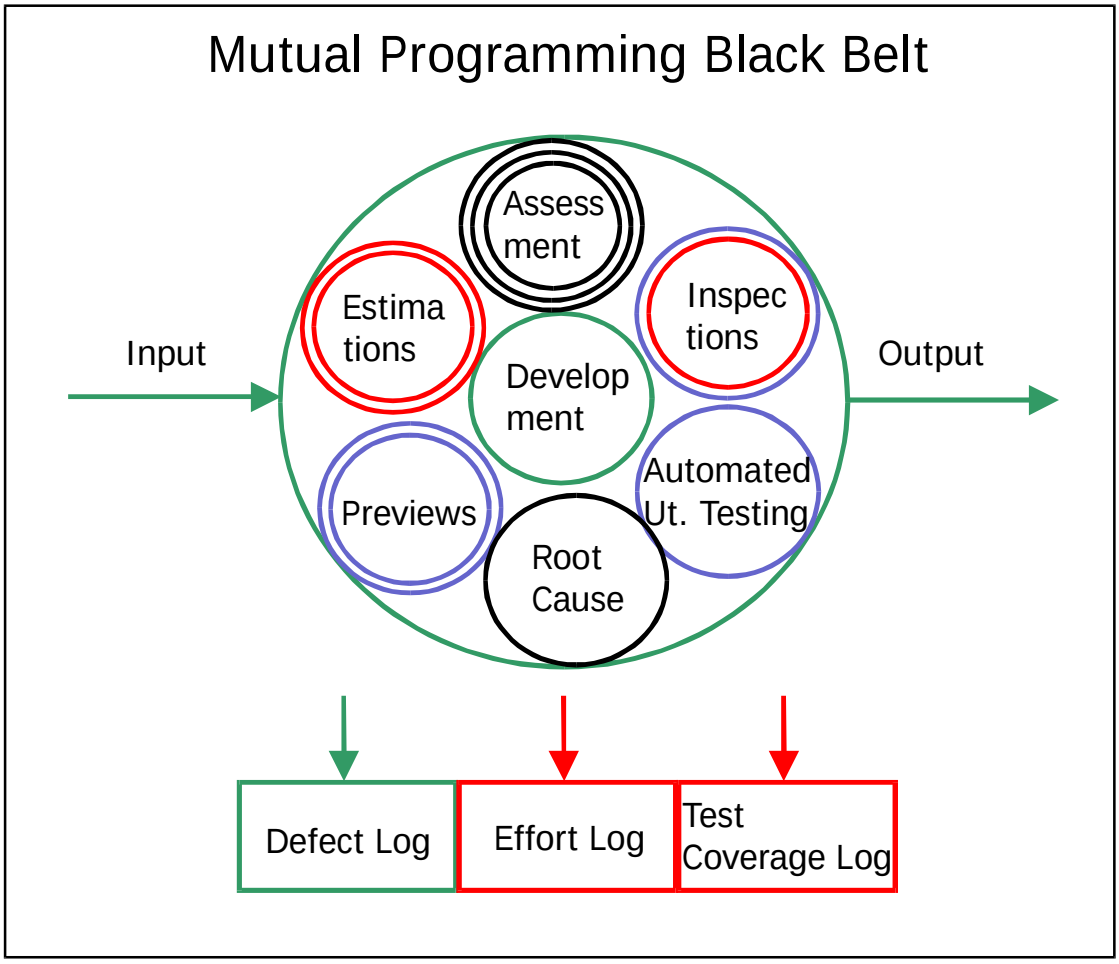
- 2 developers, one doing actual development and the other doing QA.
- MP coach.

Required Elements

- All required and recommended elements at the red belt level.
- Developers should have a complementary skill profile.
- Formal work product risk and opportunity assessment conducted by the MP coach prior to every major work product (more than 10 days effort) task as an enhancement of the work product preview.
- Root cause analysis of each detected defects by the MP coach.
- The QA responsible and the MP coach must approve release.

Recommended Elements

- Statistical analysis of process metrics by the MP coach.
- ATA blocks as described in [9] in program code.



7 Element Explanations

7.1 DLDA

Defect logging and defect analysis (DLDA) is a technique that the individual developer employs during daily work. A set of mnemonics is used to precisely document what has happened and when with identified defects during inspections, testing, debugging, and integration or other software related work.
Each defect is assigned an identification number and defect the defect lifetime stages are recorded in a DLDA logfile with consecutive log statements.

The following status mnemonics exist:

- bd: “begin defect”.
- bs: “begin search”.
- ss: “search successful”.
- su: “search unsuccessful”.
- bf: “begin fixing”.
- ef: “end fixing”.
- bv: “begin fix verification”.
- ed: “end of defect”.

#	Current state	Event	Next state	Action
1	Undefined	Possible defect encountered.	bd	Generate ID and write log entry with timestamp.
2	bd	Begin search after some time.	bs	Write log entry with timestamp.
3	bd	Begin search immediately, problem obvious, fixed, and verified immediately.	ed	Write log entry with timestamp, defect type, defect phase, and defect reason.
4	bd	Begin search immediately, problem obvious, fixed, and verified immediately, but not successful	bd	Write log entry with timestamp.
5	bd	Begin search immediately, problem obvious, and fixed, but not verified.	ef	Write log entry with timestamp.
6	bd	Begin search immediately and search is successful (even if there is no defect).	ss	Write log entry with timestamp.
7	bd	Begin search immediately and search is not successful.	bd	Write log entry with timestamp.
8	bs	Search successful.	ss	Write log entry with timestamp.
9	bs	Search successful, fix defect immediately, and verified.	ed	Write log entry with timestamp.
10	bs	Search successful and fix defect immediately, but not verified.	ef	Write log entry with timestamp.
11	bs	Search not successful.	bd	Write log entry with timestamp.
12	ss	Begin fixing defect after some time.	bf	Write log entry with timestamp.
13	bf	Fixing finished.	ef	Write log entry with timestamp.
14	ef	Begin fix verification.	bv	Write log entry with timestamp.
15	bv	Verification successful.	ed	Write log entry with timestamp, defect type, defect phase, and defect reason.
16	bv	Verification not successful.	bd	Write log entry with timestamp.

Initially only the bd and ed entries might be used in order to increase developer acceptance. Later on the stage “search”, “fix”, and “verification” should also be separately logged.

7.2 Three Questions

This is a technique suggested by Tom Van Vleck [15]. For each detected defect the developer has to give answers to the following three questions:

1. Is this defect somewhere else also?

2. What follow-on defect might be caused by a fix of this defect?

3. What should I do to prevent defects like this?

At the blue, red, and black belt level the developer will be asked those questions by the QA, who in addition can give further advice.

7.3 Automated Unit Testing

Unit testing is the classical arena of white box testing techniques. Following the nomenclature of Boris Beizer, data flow testing, control flow testing, finite state machine testing, loop testing, or domain testing are all techniques that can be applied for unit testing. For the purpose of automation xUnit testing tools are recommended that are available for close to all c programming languages currently in widespread use.

7.4 Inspections

Most of what has to be said about inspections can be found in Michael Fagan's paper [3]. In situations where no moderator is available, a modified variant of inspections is possible, whereby the QA reviews the work product and the findings are discussed in a meeting together with the developer. However, this practice is only allowed at the blue belt level and with participants that have substantial experience with inspections. At the red and black belt level a trained moderator is required.

7.5 Work Product Previews

A work product preview is a formal meeting with the intention to align the stakeholders about the purpose, scope, and constraints of the work product to be built. The specification of

- use,
- change, and
- test cases

is a major goal of this meeting.

The following participants roles are recommended and you can certainly assign more than one role to a person:

- Domain expert.
- Architect.
- Developer.
- Tester/QA.
- Product manager.
- User.
- Integrator.
- Program manager.

7.6 Coverage Measurement

There are basically three coverage metrics in use:

1. C0 (all statements),
2. C1 (all statements and conditions), and
3. C3 (all paths).

To achieve the red belt level, it is required that the C0 statement coverage criteria is reached and measured with a code coverage tool. Alternatively a well thought out program tracing or a debugger can be helpful in the absence of such a tool.

7.7 Effort Tracking

This practice requires the developer and the QA to track the effort they spent on certain tasks, such as

- requirements,
- specification,
- design,
- coding,
- test,
- reviews.

In addition the tracking of tasks that are not directy related to development, like communications, meetings, trainings, etc. could also be useful for the reason that in some environments the time spent on actual development tasks is as low as 18 hours per week.

7.8 Formal Effort Estimation

As an alternative to simple guessing, formal effort estimation is required starting from the red belt level. This involves the developer and the QA, who both have to provide three estimates:

- best case.
- most likely case.
- worst case.

Those estimates are then put together for an overall estimate that should be compared to actual effort later on. More information can be found in the article [4].

7.9 Program Trace Files

Program trace files should track all the function or method call sequences. In addition, they should cover at least the values of important variables at checkpoints. If more information is needed, the information content of program traces should be adjustable. Program trace files are very helpful for debugging purposes and for desk-checking of program behaviour.

7.10 Formal Work Product Risk and Opportunity Assessment

A work product risk and opportunity assessment is basically a risk management process such as the one described in [8] tailored and applied to major work products.

7.11 Root Cause Analysis

The practice of root cause analysis has been described in several articles, for example in [2].

7.12 Defect Density Tracking

Defect density is the number of defects found per work unit. Work unit could be for example KLOC or a certain number of written pages. To calculate the defect density, defect log entries have to be related to work products and the work product size has to be measured.

8 Tools

8.1 DLDA

We set up a tool that let developer enter their defect information with an HTML-form.

8.2 Effort Tracking

There are several free effort tracking tools around. See for example http://www.allnetic.com .

8.3 Automated Unit Testing

Please have a look at the various programming languages with xUnit support at the www.junit.org web site. There is also a cUnit test framework set up by Chuck Allison available for ANSI C.

9 Conclusions

We described a levelled approach to introduce a disciplined method for the development of software and software related work products. Close to all of the described practices have been around for quite some time and we don't claim that those practices are new. However, we are convinced that a major hurdle for software development productivity increase is not the absence of practices, but the lack of their application to daily work. The method therefore balances between simplicity and formal rigour in a way that makes adoption easy, but does not compromise on necessary formal aspects, such as explicit, measurable quality assurance, and clear entry and exit criteria.

We think that the mutual programming approach will make the introduction of formal rigour easier, because at each level the feasibility of more formalism can be actually measured. The black belt level should sufficiently cover major CMM/CMMI level 5 aspects for implementation tasks.

10 References

1. D. R. McAndrews, 2001/2000, *The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices,* SEI Technical Report CMU/SEI-2000-TR-015.

2. D. N. Card, *Learning from Our Mistakes with Defect Causal Analysis*, IEEE Software, Vol. 15, No. 1 (January/February 1998), pp. 56-63.

3. M.E. Fagan*, Design and code inspections to reduce errors in program development***,** IBM Systems Journal 15(1976) pp 182-211.

4. P. Gartner, *Die Drei-Punkt-Schätzmethode zur Kalkulation des Projektaufwands*, Projektmanagement 4/99.

5. W. Humphrey*, A Discipline for Software Engineering***,** Addison-Wesley, 1995.

6. W.S. Humphrey, *The Team Software Process (TSP)*, SEI Technical Report CMU/SEI-2000-TR-023, 2000.

7. G. Keefer, *Extreme Programming Considered Harmful for Reliable Software development*, AVOCA Technical Report, 2002.

8. G. Keefer, *A CMMI Compatible Risk Management Process*, PSQT South 2002 Presentation.

9. K. Miller, *A modest proposal for software testing*, IEEE Software, Vol. 18, No. 2 (March/April 2001), pp. 96-98.

10. J. Nawrocki, A. Wojciechowski, *Experimental Evaluation of Pair Programming,* ESCOM 2001 Paper.

11. L. Prechelt et al., 1997, *Experience Report:Teaching and Using the Personal Software Process (PSP)*, University of Karlsruhe, Submission to the ESEC.

12. L. Prechelt et al., 1999, *A Controlled Experiment on the Effects of PSP Training: Detailed Description and Evaluation*, University of Karlsruhe Technical Report 1/1999

13. L. Prechelt, *Accelerating Learning: from Experience: Avoiding Defects Faster*, IEEE Software, pp. 56 - 61, November/December 2001.

14. R. Smith, J. Hale and A. Parrish, *An Empirical Study Using Task Assignment Patterns to Improve the Accuracy of Software Effort Estimation*, IEEE Transactions on Software Engineering, Vol. 27, No. 3, March 2001.

15. Van Vleck, Tom, *Three Questions About Each Bug You Find*, Tom Van Vleck Web site.

16. Webb, D. and Humphrey, W., 1999, *Using the TSP on the TaskView Project*, Crosstalk 12, 2 February 1999.

11 Tools

12 Contact Data

Gerold Keefer / Hanna Lubecka AVOCA GmbH
Kronenstr. 19,
D-70173 Stuttgart
Germany
Phone: +49 711 22 71 374
Fax: +49 711 22 71 375
E-mail: info@avoca-vsm.com
http://www.avoca-vsm.com

