Self-Governance Developer Framework

Mira Kajko-Mattsson
School of ICT
KTH Royal Institute of Technology
Stockholm, Sweden
email: mekm2@kth.se

Gudrun Jeppesen
Department of Computer and Systems Sciences
Stockholm University
Stockholm, Sweden
email: gudrunjep@telia.com

Abstract—**Success of software developers should be attributed to developers' knowledge of what to do and their discipline and trust to their self-organization. To achieve this, the software community should provide appropriate process frameworks recommending developers what needs to be done, still however, allowing maximal freedom, flexibility and self-discipline. The Self-Governance Software Developer (SGD) Framework is the solution to this. In this paper, we suggest and motivate the SGD Framework. We also benchamark it against Personal Software Process (PSP). Our results show that SGD has a higher coverage of the developer activities. Still, however, it needs to be evaluated within the industrial context.**

Keywords-personal software process; self-discipline; self-organization; software development; software methods, process models, coding, unit testing.

I. INTRODUCTION

Discipline and know-how takes many forms and permeates almost every aspect of software development. Disciplined and knowledgeable developers and/or teams know what is expected from them in specific development contexts. They know best what activities to choose and how to organize their work for the success of their projects. Undisciplined and/or less experienced developers/teams, on the other hand, may not always know what to do and are not always able to deliver quality code on time and within budget.

Many sources tell software developers what to do. The most common ones are various software development methods [1]-[3][7]-[9][12][16], or guidance from managers [4], or organizational in-house methods [2]. Irrespective of whether they are waterfall, iterative or of any other nature, most methods impose sets of development activities that are not always applicable in all kinds of development contexts. Also, managers and/or organizations impose specific methods to developers which are not always explicitly stated and/or well motivated. This may limit the freedom of developers and make them into passive workers who conduct tasks to which they are not always convinced [5]-[7][13][18].

There is a big difference between developers deciding what to do and being told what to do. Making decisions on your own implies freedom. Developers become more self-driven, enthusiastic and motivated about their work [3][6][12][14]. By learning on their own mistakes, they become more experienced, and hopefully, more mature software developers. The modern methods have recognized this, and therefore, they have given more freedom to developers by eliminating the rigidity of development methods and by decreasing the authority of the managers. The modern methods give more trust and freedom to developers by allowing them to self-govern their own work, learn from their own experience and mistakes, and take their consequences [4][5][7][8][10][11].

Currently, the idea of self-governance is becoming more and more omnipresent within software development. Individual developers and/or teams are expected to exercise most of their necessary functions without intervention from others. This may work well, as long as developers and teams know what to do in order to achieve the best possible results. Unfortunately, there are not many process models providing them with this type of knowledge.

Today, there are no standard process models specifying complete lists of activities as required of software developers. Regarding the current software engineering literature, the lists of activities to be conducted by developers are scattered across various books or articles. The most relevant and all-inclusive sources are (1) Personal Software Process (PSP) as written by Watts Humphrey [7] and (2) our works on developer tests [8][9]. Usually, complete lists may be found only in the industry.

Most of the companies provide some kind of support telling developers what to do. This support is realized in form of process models or methods. The level of formality and rigidness of these models may vary from company to company. Some provide developers with strict sequences of activities which must be conducted step by step. Some others give free hands to developers in deciding what to do. Here, the choice of activities strongly depends on the developers, their knowledge of software development process, maturity, experience, and most importantly, their ability to self-govern themselves.

Even if it is highly desired, self-governance does not always function in many development contexts. There are many reasons to this. Some of them are that developers may not always be aware of what to do and when, or they may not be disciplined enough, or due to various external forces, they may be forced to choose the shortest, however, not always the most optimal way of organizing and conducting their own work.

Success of today's developers should be attributed to their knowledge, discipline and trust. To achieve it, the software community should provide process frameworks

 ISBN: 978-1-61208-553-1

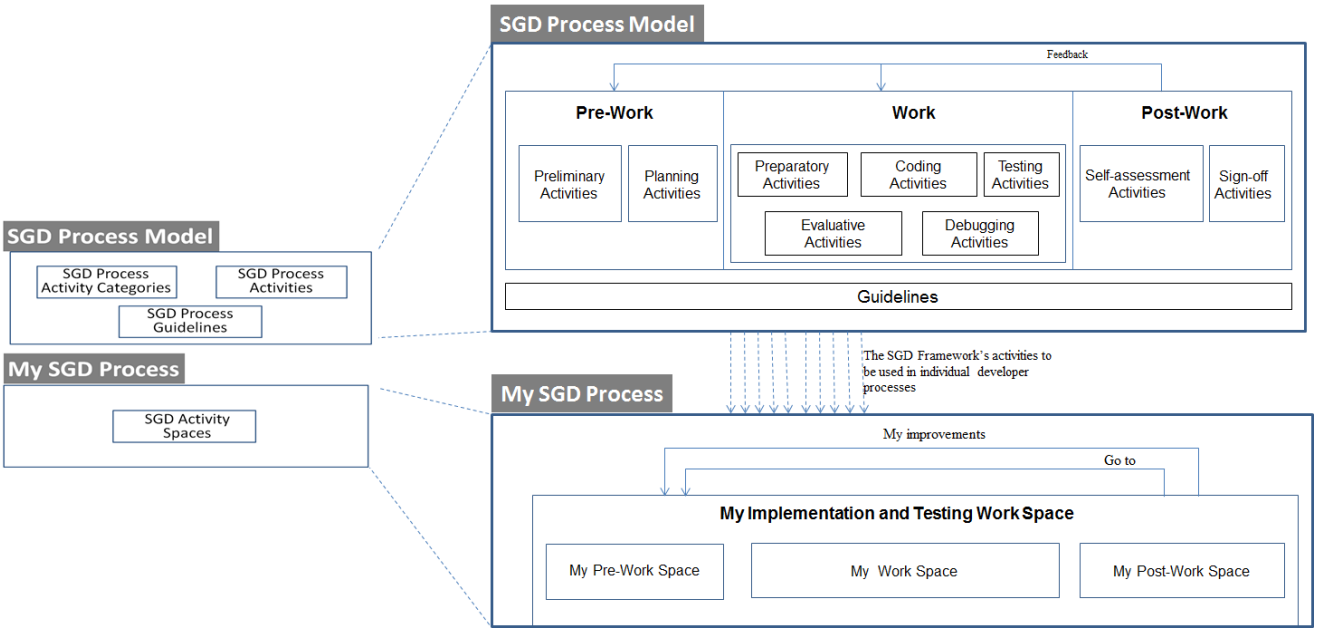

Figure 1. Structure of *the Self-Governance Developer Framework*

telling developers what needs to be done, still however, giving them maximal freedom to organize their own work [15][17][18]. The *SGD Framework* is the solution to this.

In this paper, we suggest and motivate the SGD Framework. SGD is an extension of PSP [7]. To illustrate the enhancements, we benchmark SGD against PSP [9]. The remainder of this paper is as follows. Section II presents the SGD Framework. Section III benchmarks the framework against PSP. Finally, Section IV makes concluding remarks.

II. THE SGD FRAMEWORK

The *SGD Framework* provides generic activities that can be selectively chosen by software developers or teams while implementing software code and unit (developer) testing it. The goal of SGD is to support developers in their daily work by assisting them in self-managing, monitoring and controlling their own assignments. The framework's target groups are software developers and teams whose main task is to code, compile, unit test and integrate their own code units before delivering them for integration and system testing. It is an extension of Watts Humphrey's PSP and of our former work [7]-[9].

The *SGD Framework* is structured into two parts. As shown in on the left-hand side of Fig. 1, these are (1) *SGD Process Model* and (2) *My SGD Process*. The *SGD Process Model* consists of (1) *SGD Process Activity Categories*, (2) *SGD Process Activities*, and (3) *SGD Process Guidelines*. This paper only focuses on (1) *SGD Process Categories* and (2) *SGD Process Activities*. It excludes *SGD Process Guidelines*.

A. SGD Process Model

The *SGD Process Model* consists of three main process parts. These are (1) *Pre-Work* (2) *Work*, and (3) *Post-Work*. The model's activities cover a wide and all-inclusive spectrum of activities that are relevant for conducting implementation and unit (developer) testing. In actual development endeavors, however, not all of the activities need be conducted. In some contexts and/or programming environments, only their subsets may be relevant. For this reason, the *SGD Process Model* includes the *SGD Process Guidelines* providing suggestions for what activities to conduct, when and why.

As illustrated on the right-hand side of Fig. 1, the SGD activities are grouped into nine categories that are distributed across the three above-listed *SGD* process parts. In the model, they are put in the part and category in which they are contextually relevant. They are also listed in the order that may correspond to a logical workflow. This may make the model be understood as traditional and heavyweight. However, the *SGD Framework* does not impose any specific order of activities. The activities may be conducted in any order and they may be included in any process phase that suits the developers and their environments. For simplicity reasons, however, they are mentioned in the *SGD Process Model* part only once. Developers are free to use them in the order that best suits their requirements, needs, formality levels, development approaches, contexts and specific working and/or technological environments as long as their choice contributes to product and process quality.

1) Pre-Work Activities: The SGD Process Model's first part, the *Pre-work* part, includes activities that need to be conducted before starting the implementation work. The *Pre-Work* activities are listed in Table I. They deal with

TABLE I. PRE-WORK ACTIVITIES (+ IMPLIES PRESENCE, – IMPLIES ABSENCE, P STANDS FOR PARTIALLLY)

SGD PRE-WORK ACTIVITIES	
SGD PRE-WORK: PRELIMINARY ACTIVITIES	**PSP**
PR-1: Review and agree on the overall or part of the project plan	+
PR-2: Revise and ensure that the technology to be used is tested and understood	–
PR-3: Revise and understand any appropriate internal (organizational) and external standards	I
PR-4: Learn/relearn the organization's implementation and unit (developer) testing way of work	–
PR-5: Review and revise your personal implementation and unit (developer's) testing way of working	I, P
PR-6: Sign your personal Service Level Agreement (Work Contract)	P, I
SGD PRE-WORK: PLANNING ACTIVITIES	**PSP**
PL-1: Review the requirement(s) for the unit(s) to be developed	–
PL-2: Prepare (make) and/or review the design specifications for the unit(s) to be developed	+
PL-3: Resolve unclear questions and uncertainties	P, I
PL-4: Determine your implementation and unit(developers) testing goals	+
PL-5: Determine your implementation and unit (developer's) testing strategy/ies	I
PL-6: Determine your implementation and testing practices	I
PL-7: Identify standards to be used for meeting your goals	P
PL-8: Set you own personal deadlines to be met during your implementation and unit (developer's) testing work	+
PL-9: Estimate effort and resources required for carrying out your work	+
PL-10: Schedule your work	+
PL-11: Review your implementation and unit (developer) testing plan to ensure that it is realistic and achievable	I
PL-12: Identify risks related to your plan	–
PL-13: Plan for managing any identified risks	I, P

identifying goals to be achieved, formulating strategies for achieving the goals, arranging or creating ways for reaching them, and with monitoring and controlling the implementation and unit (developer) testing steps. They aid developers in achieving an optimal balance between the development requirements and the available resources.

The *Pre-work* part consists of *Preliminary* and *Planning Activities*. They support developers in initiating their work and in creating their own implementation and unit testing personal plans. Although they are listed in the *Pre-Work* category, they may very well be conducted both before and during the actual implementation and testing work. This, of course, depends on the development context at hand and the needs that have arisen in that context.

a) Preliminary Activities: The *Preliminary Activities* are to be conducted before starting the implementation and unit (developer) testing work. They should be carried out before the actual implementation work begins. They prepare developers for performing high quality work. Here, the concerns are making sure that methodologies, technologies, standards, ways of working, commitments are understood and are in place. The SGD Framework strongly recommends that developers consider them before launching their individual development endeavors. Their non-performance may imply various risks that may jeopardize development work and results.

To carry out their work in the best possible way, developers should frequently learn or relearn the organizational ways of working, revise and ensure technologies and revise and understand standards that they are going to use (see Activities PR-2 – PR-4 in Table I). They must also pay attention to their past experiences in order to be able to improve and determine their ways of working (see Activity PR-5 in Table I). This is pivotal for sustaining quality and technologically up-to-date and standard-adhering work. If developers do not spend enough time on these activities, they may run the risk of repeating pitfalls of previous projects.

To find out about available resources and timescales for their work, developers should review and agree on the overall project plan in case of small projects or on parts of the project plan in case of large projects (see Activity PR-1 in Table I). This will enable them to plan their own work so that they can meet the stated requirements and customer expectations. Finally, the SGD Framework recommends that all developers sign their personal Service Level Agreements (SLAs) – contracts in which they commit themselves to conduct their work according to the agreed upon standards and expectations (see Activity PR-6 in Table I).

b) Planning Activities: The *Planning Activities* aid in formulating the initial and continuous development plans. They deal with (1) reviewing the necessary documents, (2) determining ways of conducting the work, and (3) planning the work.

Developers should review the documents that provide important input for understanding the scope of their work. This includes reviewing of requirements and preparing and/or reviewing of design specifications (see PL-1 – PL-2 in Table I). In many cases, requirements and design specifications may not be easy to understand. To free themselves from any misunderstanding and/or confusion, developers should resolve all kinds of unclear questions and uncertainties (see PL-3 in Table I). In this way, they make sure that they acquire a true picture of the user requirements, that the design correctly reflects the requirements, and that their plans are based on realistic premises. Having understood the requirements and design specifications aids developers in determining the limits and approaches while planning their individual work.

The SGD Framework recommends that developers determine implementation and unit (developer) testing goals and strategies and practices that will guide them in their planning (see PL-4 – PL-6 in Table I). Developers should

 ISBN: 978-1-61208-553-1

TABLE II. WORK ACTIVITIES (+ IMPLIES, – IMPLIES ABSENCE, I MEANS IMPLICITLY, P STANDS FOR PARTIALLY)

SGD WORK ACTIVITIES	
SGD WORK: PREPARATORY ACTIVITIES	**PSP**
P-1: Prepare (make) and/or review your low-level design(s) of the code to be written or changed	+
P-2: Prepare (make) an impact analysis of your low-level design(s)	–
P-3: Determine the types of unit (developer) test cases and their order	I
P-4: Create and/or revise your unit (developer) test case base	+
P-5: Revise the existing unit (developer) regression test base, if relevant	–
P-6: Create or modify stubs and drivers, if required	–
P-7: Prepare your unit (developer) testing environment and check whether it is appropriate for you work	–
SGD WORK: CODING ACTIVITIES	**PSP**
C-1: Write/rewrite your code	+
C-2: Compile/recompile your code	+
C-3: Make notes on your compilations errors, if necessary	+
C-4: Make notes your defects	I, P
SGD WORK: TESTING ACTIVITIES	**PSP**
T-1: Check whether the unit (developers) test case base meets the given requirements and design	–
T-2: Check whether the unit (developers) regression test case base meets the given requirements and design	–
T-3: Remedy requirement problems in your unit (developers) regression and/or test case base, if any	–
T-4: Perform dynamic testing by executing code	+
T-5: Perform static (human) testing by reviewing your code	+
T-6: Record/write down test results	+
SGD WORK: EVALUATIVE ACTIVITIES	**PSP**
E-1: Analyse your unit (developer) testing results	+
E-2: Depending on the unit (developers) testing results, determine your next step(s)	P
SGD WORK: DEBUGGING ACTIVITIES	**PSP**
D-1: Identify the source of error(s)	+
D-2: Determine solutions(s) for eliminating the sources of error(s)	+

then identify all kinds of standards that need be considered during implementation, set deadlines that need be met, estimate effort and resources and make their personal work schedules (see PL-7 – P-10 in Table I).

After having created their individual plans, developers evaluate them (see PL-11 in Table I), identify risks related to the plans (see PL-12 in Table I) and plan for managing the identified risks (see PL-13 in Table I). In this way, their plans will achieve the right balance of scope, approaches, resources and risks allowing developers to achieve their goals in the best possible way.

2) The Work Activities: The SGD Process Model's second part, the *Work Activities*, includes activities required for producing code and for assuring its quality. It consists of five categories of activities: (1) *Preparatory Activities*, (2) *Coding Activities*, (3) *Testing Activities*, (4) *Evaluative Activities*, and (5) *Debugging Activities*. They are all listed in Table II.

a) Preparatory Activities: The *Preparatory Activities* include the activities needed for preparing the implementation work. They help developers to become ready for writing and unit (developer) testing code. The activities deal with low-level designs, unit (developer) test case designs, stubs and drivers, and unit (developer) testing environment.

Before coding, developers should make the low-level designs of the code they are going to write or, in cases when someone else is responsible for making low-level designs, they should review them. They should also make impact analysis of the designs. The SGD Framework recommends that developers prepare and/or review several design solutions, analyze the impact of the solutions and select the most appropriate solution for the work at hand (see P-1 and P-2 in Table II). This will aid them in creating the best possible solutions for the user requirements and the given premises.

Developers should determine the types of unit (developer) test cases and the order in which they should be run. They should create or revise their own unit test case bases and regression unit test case bases (see P-3 – P-5 in Table II) and create or modify stubs and drivers, if necessary (see P-6 in Table II). Finally, developers should prepare or check their testing environments to enable continuous and smooth testing without any technical interruptions (see P-7 in Table II).

b) Coding Activities: The *Coding Activities* deal with code production including writing or rewriting code and compiling it. The SGD Framework recommends that code be produced using the chosen low-level design. If code is not based on any low-level design, then the risk is that it may not meet the stated requirements. The coding activities even include making personal notes on the compilation errors and on the detected defects (see C-1 – C-4 in Table II). This will help developers monitor their work, evaluate the quality of their work and help them learn from their own coding mistakes.

c) Unit Testing Activities: The *Unit Testing Activities* aid in assuring that the code meets the stated quality goals. They include (1) unit testing activities and (2) control of unit test cases. The unit testing activities encompass dynamic and static testing and the recording of the test results (see T-4 – T-6 in Table II). The control of the unit test cases, on the other hand, encompasses the review of the unit test case bases with the purpose of checking whether they still meet the given requirements and/or designs. Even if developers have created or revised the unit test case bases before starting coding, they should check them anew after

 ISBN: 978-1-61208-553-1

TABLE III. POST-WORK ACTIVITIES (+ IMPLIES PRESENCE, – IMPLIES ABSENCE, I MEANS IMPLICITLY, P STANDS FOR PARTIALLY)

SGD POST-WORK ACTIVITIES	
SGD POST-WORK: SELF-ASSESSMENT ACTIVITIES	**PSP**
A-1: Assess your own development work	+
A-2: Identify sources of your mistakes	+
A-3: Identify improvement areas in your own way of working	+
SGD POST-WORK: DELIVERY AND SIGN OFF ACTIVITIES	**PSP**
S-1: Check that your code fulfills the commitment(s) stated in the Service Level Agreement	P
S-2: Deliver your code	+
S-3: Sign-off your personal Service Level Agreement	–

having implemented the code. If they find any problems in them, then they should remedy them. It is only after coding that developers may clearly see what changes need to be done to the unit test case bases (see T-1 – T-3 in Table II).

d) Evaluative Activities: The *Evaluative Activities* deal with the evaluation of unit (developer) testing results and determination of the next step (see E-1 and E-2 in Table II). They should be conducted right after the unit (developer) testing activities and before starting the next series of implementation and unit (developer) testing steps. In this way, developers will make sure that they have chosen the workflow that is appropriate for their work context at hand.

e) Debugging Activities: The *Debugging Activities* aid developers in identifying the sources of the errors that have been discovered during compilation and unit testing and in suggesting solutions for eliminating them (see D-1 and D-2 in Table II). The errors are only symptoms of defects and they may not always be visible. Therefore, it is important that developers (1) debug code for the errors that are not easy to interpret and (2) confirm their underlying defects before deciding on how to attend to them. Otherwise, the defects may reappear either in the same or some other disguise.

3) The Post-Work Activities: The Process Model's third part, the *Post-Work* part, includes activities required for finalizing the implementation and unit testing. They are listed in Table III. Here, the SGD Framework suggests that developers make a self-assessment of their own development work before they deliver their code to integration and system testing and that they sign-off their personal SLAs. When assessing their development work, developers should identify causes of their mistakes and identify improvements that should help them avoid future mistakes (see A-1 – A-3 in Table III). This will help developers become more effective and efficient.

When signing off their personal SLAs, developers should first check that their code fulfills the commitments that they have agreed to before starting their work (see S-1 in Table III). They should then deliver their code (see S-2 Table III) and, finally, sign-off their assignments (see S-3 in Table III). In this way, developers will make sure that they have performed all the work stipulated in their personal SLAs.

B. *My Process Part*

My *SGD Process* corresponds to the actual developer process as planned and conducted by individual developers and/or teams. As shown in Fig. 1, it consists of three essential activity spaces. Activity spaces are empty spaces that are to be filled in by developers themselves with the activities from the *SGD Process Model.*

Not all of the SDG process model activities may be necessary to conduct in all development contexts. In some contexts, only their subsets may be relevant. For this reason, the *SGD Framework* only provides empty activity spaces that are to be filled in by the developers with the activities which they have selected by themselves. The selected activities are the reflection of developer's workflows that have been conducted or are going to be conducted. Their choice depends on the chosen strategies, methodologies and individual developer or team preferences.

As shown in Fig. 1, the SGD Framework suggests three main activity spaces. These are (1) *My Pre-Work Space* to be filled in with the start-up activities, (2) *My Work Space* to be filled in with the actual development and testing activities, and (3) *My Post-Work Space* to be filled in with concluding activities.

The *My Pre-work* activity space is to be filled in with the activities that developers need for initiating and planning their work. The activities to be used in this space are mainly the activities from the *SGD Pre-Work* part including *Preliminary* and *Planning Activities* (see Fig. 1).

The *My Work* activity space is to be filled in with the activities that developers perform when implementing and testing their code. The activities to be used in this space are mainly the activities from the *SGD Work* part including *Preparatory Activities*, *Coding, Unit Testing, Evaluation* and *Debugging* (see Fig. 1). In addition to this, the *My Work* space may include sets of activities from the *SGD Pre-work* part that developers need for conducting their continuous preparation and planning.

Finally, the *My Post-Work* activity space is to be filled in with the activities that conclude the implementation and unit testing work. The activities to be used in this space mainly come from the *SGD Post-Work* part including *Self-Assessment* and *Delivery* and *Sign-Off Activities* (see Fig. 1). However, this space may also include the activities from the *Pre-Work* and *Work* parts in cases when developers have not fulfilled their SLA commitments and, thereby, have to finalize their work before submitting their code for system integration.

III. BENCHMARKING THE SGD FRAMEWORK

The SGD Framework was benchmarked against PSP [7]. While benchmarking, we simply checked whether PSP included the SGD activities. The presence of the activities is marked with a plus (+), the absence is marked with a minus

 ISBN: 978-1-61208-553-1

(–). Unclear cases, such as implicit or partial presence of the activities, are marked with I standing for implicit and P standing for partial implementation.

The benchmarking results are presented in Tables I-III. As can be seen there, PSP does not fully cover any of the SGD Framework categories. Below, we briefly comment on the benchmarking results for each of the categories.

Regarding the *Pre-Work* activities, PSP has performed poorly. It does not encourage developers to revise and ensure that the technology to be used is tested and understood (Activity PR-2 in Table I). Neither does it suggest that developers learn or relearn the organizations' implementation way of working (PR-4 in Table I). We believe that these activities are pivotal for succeeding with the implementation work. Both technology and ways of working continuously evolve. Lack of knowledge about them may lead to substantial productivity loss. Finally, PSP is not clear about whether developers should review and revise their own implementation ways of working (Activity PR-6 in Table I). In our opinion, this is a severe omission considering the fact that this activity is driving the whole personal software process.

Regarding the *Planning* activities, PSP fails to suggest that developers review the requirements for the units to be developed (Activity PL-1 in Table I). This may lead to the fact that developers may misunderstand the requirements and develop things that have not been expected from them. PSP also fails to suggest that developers identify risks related to their own personal plans (Activity PL-12). Again, this activity is one of the driving wheels of a disciplined personal developer process.

PSP is not explicit enough about activities related to determining implementation and testing strategies and practices (Activity PL-5 and PL-6 in Table I) and in reviewing the developer plan for assuring that the work is realistic and achievable (PL-11 in Table I). We believe that this activity is very important. Not considering it may lead to failure of delivering code in time or it may result in never delivering it due to the unrealistic personal plans.

Considering the *Preparatory* activities in the *Work* part, PSP does not consider the fact that developers should make an impact analysis of their low-level designs (Activity P-2 in Table II). Neither does it consider the fact that developers should revise the existing regression test base (P-5 in Table II) and that they should prepare and check whether the testing environment is appropriate (P-7 in Table II).

PSP covers all the SGD coding activities with one exception. It does not encourage developers to make notes on their defects (Activity C-4 in Table II). We believe that this activity is important from the perspective of individual professional development. By remembering defects and analysing their root causes, that is, mainly mistakes, developers will improve their professional skills and become better at developing software.

In addition to traditional testing activities, SGD includes checks whether unit test bases and regression test bases meet the given requirements and designs (T-1 – T-2 in Table II). PSP does not consider these activities at all. Neither does it assume that there may be requirement problems in the regression test bases (T-3 in Table II). Requirements may change with time and this should be reflected in the regression test base. Lack of the activities T-1 – T-3 may lead to the omission of testing important requirements and late discovery of defects, either during integration and system testing or even during operation.

Regarding the remaining *Work* activities, such as *Evaluative* and *Debugging* activities, PSP has implemented them all. PSP also implements all but one *Post-Work* activity. The activity that it does not implement concerns signing off SLAs (see Activity S-3 in Table III).

IV. FINAL REMARKS

Self-governance should bring value in form of improved developer productivity and job satisfaction. Developers should be able to decide upon what activities to choose based on the value the activities bring. This has been recognized in PSP as suggested by Watts Humphrey [7].

In this paper, we have suggested Self-Governance Developer Framework outlining the activities aiding developers and/or teams in designing their own personal processes. SGD only provides a basic conceptual structure of the activities and provides guidelines for performing them. It does not provide any suggestion for any order among the activities. Neither does it define inputs and outputs of the activities. As a framework, it constitutes a platform for creating developer process models, which in turn, are free to define their own order, inputs and outputs, and provide guidance in decision making.

The SGD Framework is a continuation and extension of PSP [7] and of our earlier work on developer testing process [8][9]. So far, it has only been evaluated against PSP. It has not yet been evaluated against other standards and industrial or academic models. Evaluation, however, is on its way. Right now, we are conducting active research by studying activities as conducted by software engineering students at KTH Royal Institute of Technology [19][20]. We are also in the process of evaluating the SGD with the industrial software engineering professionals.

REFERENCES

[1] P. Abrahamssom and K. Kautz, "The Personal Software Process: Experiences from Denmark," Proc. Euromicro Conference. IEEE. Sept. 2002, pp. 367-374, doi: 10.1109/EURMIC.2002.1046223.

[2] F. Abdolazimian and S. Mansouri, "Business Process Reengineering by Rational Unified Process (RUP) Methodology,". *World Applied Sciencies Journal 4, (Supple 2)*, IDOSI Publications, pp. 33-42, 2008.

[3] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, F. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, and K. Schwaber, "*Manifesto for Agile Software Development*,". [Online]. Available from: http://agilemanifesto.org/, 2017.03.15.

[4] K. Culver-Lozo, "The Software Process from the Developer's Perspective: A Case Study on Improving Process Usability," Proc. Ninth International Software Process Workshop. IEEE, Oct. 1995, pp. 67-69, doi: 10.1109/ISPW.1994.512766.

 ISBN: 978-1-61208-553-1

[5] W. Hayes, "Using a Personal Software Process[SM] to improve performance," Proc. Fifth International Software Metrics Symposium. Metrics, IEEE, pp. 61-71, 1998.

[6] A. Heravi, V. Coffey, and B. Trigunarsyah, "Evaluating the level of stakeholder involvement during the project planning processesss of building projects," International Journal of Project Management, ELSEVIER, vol. 33, pp. 985-997, 2015.

[7] W. S. Humphrey, Introduction to the Personal Software Process, Addison-Wesley, 1997.

[8] G. Jeppesen, M. Kajko-Mattsson and J. Murphy, "Peeking into Developers' Testing Process," Proc. International Conference on Computational Intelligence and Software Engineering, IEEE, pp. 1-8, 2009. doi: 10.1109/CISE.2009.5366347.

[9] M. Kajko-Mattsson and T. Bjornsson, "Outlining Developers' Testing ProcessModel," Proc. 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE, pp. 263-270, 2007, doi: 10.1109/EUROMICRO.2007.45.

[10] Z. Lasio, "Project portfolio management: An integrated method for resource planning and scheduling to minimize planning/scheduling-dependent expenses," International Journal of Project Management, ELSEVIER, vol. *28*, pp. 609-618, 2010.

[11] M. Lavallé and P. N. Robillard, "The Impacts of Software Process Improvement on Developers: A Systematic Review," Proc. 34th International Conference on Software Engineering, pp. 113-122, 2012, doi: 10.1109/ICSE.2012.6227201.

[12] M. Maccoby, "Self-developers: why the new engineers work," IEEE Spectrum, IEEE, vol. 25, no. 2, pp. 50-53, 1996, doi: 10.1109/6.4511.

[13] N. H. Madhavji, X. Zhong, and E.E. Emam, "Critical Factors Affecting Personal Software Processes," IEEE Software, IEEE, vol. 17, no. 6, pp. 76-83, 2000, doi: 10.1109/52.895172.

[14] C. d. O. Melo, C. Santana, and F. Kon, "Developers Motivation in Agile Teams," Proc. 38th Euromicro Conference on Software Engineering and Advanced Applications, IEEE, pp. 376-383, 2012, doi: 10.1109/SEAA.2012.45.

[15] S. Priestley, *Scientific Management in the 21th Century*. [Online]. Available from: http://www.articlecity.com/articles/business_and_finance/article_4161.shtml, 2017.03.15.

[16] K. Schwaber and J. Sutherland, *The Scrum Guide TM*. [Online]. Available from: ttp://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf#zoom=100, 2017.03.15.

[17] C. M. Thomas, "An Overview of the Current State of the Test-First vs.Test-Last Debate," Scholarly Horizons: University of MinnesotaMorris Undergraduate Jounal, vol. 1, iss. 2. [Online]. Available from: http://digitalcommons.morris.umn.edu/cgi/viewcontent.cgi?article=1015&context=horizons, 2017.03.15.

[18] S. Wambler, *Choose the Right Software Method for the Job*. [Online]. Available from: http://www.agiledata.org/essays/differentStrategies.html, 2017.03.15.

[19] University Degree Programme in Information and Communication Technology (CINTE), KTH Royal Institute of Technology in Sweden. [Online]. Available from: https://www.kth.se/student/kurser/program/CINTE/HT13/kurslista?l=en, 2017.03.15.

[20] University Degree Program in Information and Communication Technology (TCOMK), KTH Royal Institute of Technology in Sweden. [Online]. Available from: https://www.kth.se/student/kurser/program/TCOMK/HT14/genomforande?l=en, 2017.03.15.