A Tool Evaluation Framework based on Fitness to Process and Practice

A usability driven approach

Diego Fontdevila

Departamento de Ingeniería
Universidad Nacional de La Matanza
San Justo, Buenos Aires, Argentina
dfontdevila@ing.unlam.edu.ar

Departamento de Ingeniería
Universidad Nacional de Tres de Febrero
Caseros, Buenos Aires, Argentina
dfontdevila@untref.edu.ar

Abstract — **Most current and traditional research on software development tool evaluation focuses on tool capabilities and features following the traditional approach for generic software evaluation. Existing evaluation frameworks and methods address functional and non-functional requirements, constraints, technology, knowledge domain, costs and other acquisition aspects, but such approaches do not account for the context in which work is done. We propose a usability-based framework for tool evaluation in terms of fitness to the development process and practice of their users. Our contribution is a framework for relating ways of working to tool evaluation, and a concrete checklist for performing that evaluation. We present this paper as proof-of-concept of our framework and validate its applicability (but not the evaluation results) by using it to evaluate tools with which we have hands-on experience.**

Keywords-tool evaluation; usability-based framework; process and practice.

I. Introduction

Most current and traditional research on software development tool evaluation focuses on tool capabilities and features [1][2][3][4][5][6][7][8] following the traditional approach for generic software evaluation [9][10]. Existing evaluation frameworks and methods address functional and non-functional requirements, constraints, technology, knowledge domain, costs and other acquisition aspects, but such approaches do not account for the context in which work is done [11]. For example, tools with the required features might rate well in a feature-based evaluation, but support users poorly by implementing the workflows in a way that does not match the users'. Jadhav and Sonar [12] state that none of the primary studies reviewed address the final step of the selection process: "Purchasing and implementing most appropriate software package". The authors also state "good evaluation practice suggests that some action should be taken to ensure that the selected package performs as well as expected after implementation". The problem with such after-the-fact check of successful evaluation and selection is that mistakes can be very costly; that is why we propose an earlier focus on evaluating the final effectiveness of the implementation beyond traditional tool requirements.

Although research has been conducted on evaluation of technology fitness to context, including software development tools [11], the proposed method is limited to technical issues and maintains a requirements-based approach (the case study is for web services technology). Storey et al. [13] propose collaborative demonstration based tool evaluations, focusing on interoperability and tool integration, not on end user support (the target users are themselves researchers).

We propose a usability-based framework for the evaluation of tools in terms of fitness to the development process and practice of their users. Our contribution lies in providing a framework for relating ways of working to tool evaluation. We present this paper as a proof-of-concept of our framework and validate its applicability by using it to evaluate tools with which we have hands-on experience (this is considered good practice in tool evaluation [6][11]).

The capability of a tool to support the software development process and practices of its users might very well be described in terms of usability, based on the idea that any significant divergence between the tool's model of the work and the actual way the work is performed would make the tool difficult to use. A common scenario for inappropriate process implementation might be having a tool that forces a process so heavy on its users that they abandon it partially or completely. Same with practice, a practice might not fit the process, or a tool might not support the practice appropriately. For example, inconsistencies in code review practice between different teams might turn up in system testing. We consider fitness for use, a key quality notion in any product o process, and extend it to fitness to context, where context is defined in terms of software development process and practice. This work's key contribution is a checklist of specific criteria for evaluating fitness to process and practice, inspired by usability terminology.

First, we present the framework and then apply it to the evaluation of two different tools, one related to Configuration Management practices (Jenkins Continuous Integration Server [23]) and the other to Requirements and Project Management (Pivotal Tracker [24]). These tools have been chosen by theoretical sampling to provide very different process and practice coverage. Jenkins is a tool that supports a single practice, while Pivotal Tracker covers multiple processes and practices. Our evaluation method assumes the evaluators are familiar with the tool's capabilities and can focus on evaluating their fitness to process and practice. Other methods might be used as a first approach for tool evaluation, to validate basic conformance, followed by applying our approach to the top ranking alternatives.

In Section II, we define several usability principles and propose applications of those principles in the context of software development process and practice. In Section III, we use those relationships to establish tool evaluation criteria based on how well the tool supports process and practice according to those principles. Finally, in Section IV, we apply those principles and criteria to the evaluation of the two tools, and in Section V, we present our conclusions and perspectives.

II. Applying Usability Principles to Process and Practice

At this point, we need to state our working definitions of practice and process (see [14], chapter 4, for a description of the interconnection between process and practice):

Process: It is the flow of work, products and information across the organization that produces value and coordinates the activities of groups with different practices.

Practice: The term practice describes the everyday activities and experience of work. Practices comprise a process, but they can exist without a defined process. If a practice is imposed that is not viable for the people doing the work, that same people will usually redefine the practice.

The reason we choose to focus on practice and process is that process focused perspectives often ignore how the work is actually performed by teams and individuals, and thus loose information that is critical to any improvement effort. In our case, choosing the right tool for the job cannot ignore "the way we do things here to succeed" (to paraphrase the title of [15]).

Usability principles are guidelines for the design of things that are meant to be appropriate for use. They provide guidance for creating usable designs and for evaluating those designs.

Processes and practices are tools that humans use to define, coordinate and execute their activities, and provide a harness for sustainable high quality work (in [16], Alistair Cockburn presents a view of practices as one kind of tool of agile teams). As tools, their success is sensitive to the capacity of people to make use of them. This leads to the following definition:

Process/Practice Usability: A measure of how easy it is to follow a process or practice, including the effort needed to learn, the probability of making mistakes, the cost of such mistakes and the overall satisfaction and motivation promoted by following the practice or process.

In Section III, we present a detailed criteria checklist organized by usability principles to evaluate how well a tool suits the process and practice of its users. The main contribution of this work is the criteria checklist we have created inspired by those usability principles. This checklist is not a usability checklist, for it does not evaluate tool usability, it extends usability terminology to define criteria for fitness to process and practice.

In Section II.A, we offer our own working definition of several usability principles (or heuristics, as they are referred to in [17]), an example of their application to everyday things (the standard view of usability) and a description of how each principle can be applied to processes and practices.

A. Usability Principles

We define usability principles for process and practices. We then apply them to the evaluation of tool fitness to process and practice. In this section, we extend these principles described in [18] (Chapter 1) and [17] (Chapter 5) Heuristics" to define a framework for the software development domain. Here, we define the following principles:

1. Feedback
2. Affordance/natural mapping
3. Matching conceptual models
4. Tolerate mistakes
5. Force function

We have chosen these principles because of the way they resonate with software development process and practice concepts. The initial inspiration for this work came to us with the realization of the importance of the term feedback in the context of both usability and software process improvement. As we explored this idea, we found that other usability principles appeared in both contexts, for example, creating safe work environments by tolerating mistakes is a key agile tenet.

An example of usability heuristic that we have not applied here, because no specific criteria related to it seem applicable to process and practice, is avoid modes [17].

1) Feedback

When we act upon the world, there is a reaction from the world that we can perceive (based on [18], page 27).

In everyday life: When we press a floor button in an elevator, we expect it to light up to confirm that the elevator has been programmed to go to that floor, otherwise we press the button again and again.

In Practice/Process: This principle is key to Shewhart's continuous improvement cycle Plan-Do-Check-Act. The process must be such that it offers continuous feedback so that we can appreciate (and check) the effect of the improvement efforts. Idem for Practice, we need to see the effect of a practice to motivate us to maintain it.

2) Affordance/Natural Mapping

Things should by their outward nature expose what they are for, what their purpose is (based on [18], page 9, in this context affordance means "to be for" something).

In everyday things: A small red iron hammer hung in a red container next to a glass window hardly requires an "Emergency" sign to express that it is there to help us break the window (see [18], page 9, there actually is a psychology of materials such that glass by itself affords the idea of shattering).

In Practice/Process: Process activities should have obvious effect in the production of quality work by the people involved. In other words, the purpose of all activities should be so clear as to not require explanation beyond the initial adoption phase. As a corollary, process activities should then match exactly the Practice of the people doing the work (i.e., should not make them work in a way they do not believe in).

3) *Matching Conceptual Models*

Every artifact has an implicit mental model that should match that of the people doing the work (based loosely on [18], page 12).

In Everyday things: People tend to believe that if a coin is bigger, it should be worth more, but that is not always the case.

In Practice/Process: A process should match the view that people participating in it have of their work. A particularly important aspect of this is the coordination of teams with very different practices, like software development and marketing. Each team must have an enabling out-model that allows them to integrate their work (an out-model is our model of something we are mostly ignorant about; in this case, the other team and how they work [15]).

4) *Tolerate mistakes*

Since mistakes are typical of humans, things should allow us to make mistakes without incurring much rework or frustration.

In everyday things: Pushing one wrong button should not wipe out hours worth of a document we are working in. Systems should recover from mistakes easily and gracefully. When recovery is not possible, or too costly, a force function (the next principle in this Section) might be used to prevent people from making that mistake.

In Practice/Process: Activities should be designed in such a way that we do not have to do them all over again if we make a mistake. Iterative and incremental processes are good examples of this. Practices such as Collective Product Ownership, Collaborative Design, Self-Organized Teams and fluid communication channels around the people working on the product provide excellent means of reducing the impact of mistakes. A culture that fosters exploration and innovation must also “applaud” mistakes as the acceptable cost of trying out new things.

5) *Force Function*

Things should not allow us to make use of them if there is danger of grave consequences of that use.

In everyday things: Door finger protection for babies are examples of force functions put in place to avoid painful finger injuries.

In Practice/Process: Processes and practices should establish hard boundaries on activities that run the risk of breaking up the team or seriously compromising product quality. For example, the practice of working long hours can drive a developer to burnout, and is typical of processes driven by unrealistic scheduling. The force function might then the opposite practice, disciplined 40hs a week work; it is called Energetic Work and included as one of the core Extreme Programming practices by Kent Beck [19]. Another example is when a person is empowered to break a tie in an argument.

III. A Framework for Tool Evaluation

Software development tools are meant to help to work more efficiently, or to reduce the probability of mistakes, or to record information. The way the tool supports the process and practice of its users (the ones doing the work), its alignment with that process and practice, can determine the appropriateness of the tool and its overall usefulness.

In this Section, we outline a simple framework for tool evaluation based on the usability principles described. First, we describe how tool fitness to process and practice can be evaluated through the usability principles presented. We offer a set of criteria for tool evaluation for each principle, and present an example for each criterion. Finally, we present the concrete steps to be performed for tool evaluation.

A. *Tool Evaluation Criteria Checklist by Usability Principle*

1) *Feedback*

A tool should be evaluated according to its capacity to provide feedback on its successful use to support a given practice or process. Possible criteria are:

a) Calculation and display of metrics that reflect the performance of practices or process activities.
b) Validates activity results (e.g., automated test execution, static analysis, and model checkers).
c) Supports collaboration and interaction between individuals that provide the actual feedback. For example, centralized code versioning tools use two styles for coordinating modifications, copy-merge-commit (as in CVS, Concurrent Versioning System, and SVN, Subversion) and lock-modify-commit (as in Microsoft's old Source Safe). The copy-merge-commit style favors parallel modification and fast code integration; thus, providing timely feedback, whereas lock-modify-commit code versioning tools tend to delay integration and thus.

2) *Affordance/Natural Mapping*

A tool should be evaluated according to how its external appearance suggests its purpose and meaning. Possible criteria are:

a) Uses the user's language to describe practices and process activities.
b) Workflow steps in the tool match the practices and process activities (tools developed in-house tend to work much better in this respect). As an example, Defect Lifecycle Tracking tools need to have a defect lifecycle that matches the one in use by the organization.

c) Is accessible to the people doing the job and they have the appropriate privileges. As a counterexample, only a manager might be allowed to create tasks on which developers must book hours.
d) Supports recording rationale and contextual information to further under-standing between teams, especially in activities that coordinate work between different teams. For example, an Architecture Modeling Tool should record design decision rationale (see a practical application to documentation in [20], "Seven Rules for Sound Documentation", page 24).

3) Matching Conceptual Models

A tool should be evaluated according to the match (or lack thereof) between the tool's model of the work and the actual way the work is performed. Possible criteria are:

a) Supports specific practices and activities that are necessary for the process, practice or methodology: For example, a Scrum planning tool over a general purpose issue tracker, requirements management tool over a document editing requirements plugin, as opposed to end-to-end, generic software engineering tools.
b) Specificity: Tools built for a specific practice tend to match that practice very well and avoid cluttering the interface with low value features (like Jenkins for Continuous integration, described in Example 1 in Section IV.A, or most versioning systems, or UML modeling tools). In practice, such specificity needs to be balanced with good integration with tools that support related practices.
c) Cohesion: The tool supports multiple activities but they are deeply interwoven (e.g., versioning systems and requirements tracking systems, when a developer commits a change to implement a requirement or bug fix, the tool records the relationship between the two, providing traceability).
d) Flexible customization usually allows users to bend the tool to better align it with their own process and practices.

4) Tolerate Mistakes

A tool should be evaluated according to how well it reacts to problems and how helpful it its in guiding or supporting recovery of users towards more effective behavior. Possible criteria are:

a) Does not make judgmental assertions about the meaning of a practice or process activity. For example, in the case of metrics (that provide feedback), a tool should not establish fixed criteria for determining success. In the words of Tobias Mayer "metrics should be used to measure truth — not to measure success or failure. Only measures of truth can be trusted not to incite quick-fix behavior in a team" [21]. This might mean tools driving teams to react to the judgment of the tool by "pushing the dirt under the rug". As a concrete example, a while ago we helped one team to handle a problem in their automated tests. It only took a little time to isolate, but it had driven them weeks ago to disable all tests because they were failing – They had reacted inappropriately to the feedback of their tool and abandoned the good practice of automated testing.
b) Provides means to establish flexible thresholds for status, alarms and notifications, so that teams can configure them according to their context. As an example, tools that generate many e-mails a day with false positive results for a check (e.g., server monitor reporting incorrectly that a server is down) tend to drive teams to ignore any of those e-mails.

5) Force Function

A tool should be evaluated according to the force functions it provides to avoid potentially grave consequences of inappropriate use. Possible criteria are:

a) Supports rules for automatic recognition of inconsistencies. For example, does not allow improper use of a modeling language construct (In the case of UML, a semi-formal language, this can easily become a nuisance).
b) Warns or sets hard restrictions when practices reach unhealthy limits. For example, for a project management tool, a force function might be forbidding team overload.
c) Does not support poor practices because they tend to establish the inappropriate behavior into the team or organization and make it harder to fix in the future. As an example, consider tools that create an economy of compensation (points, money, etc.) for specific activities (e.g., bug fixing). Such practices tend to promote the unthinking pursuit of the compensated activities without regard to the value they provide [22]. Putting a tool in place for that will only make the practice harder to change.

IV. TOOL EVALUATION

In this section, we propose a method for applying the usability principles and criteria to tool evaluation. Evaluation is done for all practices and process activities at the same time to avoid multiple iterations that might make the framework cumbersome.

To evaluate each tool:

1. Identify practices and process activities supported by the tool.
2. For each usability principle
    a. Qualitatively evaluate the tool on each criteria related to the principle.
    b. Rate the tool on Process and Practice support.

The rating provides a simple transformation from the qualitative evaluation of the criteria above into a quantitative rating describing how well the tool follows the principle for the selected practices and process activities. Ratings can be assigned according to the following guidelines:

Low: if the tool fulfills none of the criteria.

Medium: if the tool fulfills one of the criteria.

High: if the tool fulfills two or more of the criteria.

In the following checklists, the evaluation notation is:

√ Complies with criteria (comments for specific practices or activities)

~ Partial compliance or biased interpretation (explanation)

X No compliance or particularly negative design regarding the principle.

A. Example 1: Jenkins Continuous Integration Server

Name: Jenkins

Type: Free Software

Workflow/Phase: Configuration Management

Area of focus: Practice

Main Practices/Activities: Continuous Integration

Description: "Jenkins monitors executions of repeated jobs, such as building a soft-ware project or jobs run by *cron*. Among those things, current Jenkins focuses on the following two jobs:

- Building/testing software projects continuously [...]
- Monitoring executions of externally-run jobs" [23]

Jenkins Evaluation Checklist

1) Feedback

a) Calculates and displays metrics. √ (product build and test status)

b) Automatically validates activity results. √ (automated build and tests)

c) Supports collaboration and interactions that provide feedback. √ (sends e-mails to the whole team when a build fails)

2) Affordance/Natural Mapping

a) Uses the user's language to describe practices and process activities. √ (Main entities are builds, dependencies, jobs).

b) Its workflow steps match the practices and process activities.~ (is centered on one practice, has few process issues).

c) Is accessible to the people doing the job and they have the appropriate privileges. √ (simple authorization scheme, usually developers install and manage it).

d) Provides support to record rationale and other contextual information.~ (allows users to comment almost all entities, but has no focus in rationale).

3) Matching Conceptual Models

a) Supports only practices and activities that are necessary for the process, practice or methodology. √ (Supports only the Continuous Integration practice).

b) Is designed for one specific practice. √ (See previous)

c) Supports cohesively multiple activities when they are deeply interwoven. √ (See previous)

d) Provides flexible customization for better alignment to process and practices. √ (Provides extensive customization and extensions through third-party plugins of which it has a built in market with an many options, besides its own API and being Free Software)

4) Tolerate Mistakes

a) Does not make judgmental assertions about the meaning of a practice or process activity. ~ (a broken build is considered negatively by the tool, but that is defined at the core of the practice, not the tool)

b) Provides means to establish flexible thresholds for status, alarms and notifications. √ (Allows to set custom thresholds on test code coverage, failed build mails can be sent to the author of the change or to the whole team).

5) Force Function

a) Supports rules for automatic recognition of inconsistencies. √ (Checks input values by attempting to use them proactively and offers clear error messages to advice on correcting errors).

b) Warns or sets hard restrictions when practices reach unhealthy limits. X (It does not limit too long builds).

c) Does not promote poor practice. √ (It is a lean tool focused in a single practice without unnecessary or counterproductive features).

TABLE I. JENKINS EVALUATION MATRIX

Process Activity/ Practice	***Feedback***	***Affordance/ Natural Mapping***	***Matching Conceptual Models***	***Tolerate Mistakes***	***Force Function***
Continuous Integration	High	High	High	Medium	High

The results in Table I show overall high scores for Jenkins Continuous Integration server. This fits the fact that it is a tool targeted to a single practice. In other words, if the users follow the practice of Continuous Integration, it is reasonable to expect Jenkins to evaluate as a good candidate for successful implementation.

B. Example 2: Pivotal Tracker Project Management

Name: Pivotal Tracker

Type: Application as a Service

Workflow/Phase: Requirements Management/ Project Management (Scrum)

Area of focus: Process

Main Practices/Activities: Requirements Management/ Project Planning/Project Tracking

Description: "Simple, collaborative project management." [24].

Pivotal Tracker Evaluation Checklist

1) *Feedback*
   a) Calculates and displays metrics. √ (e.g., release burn-down, expected velocity)
   b) Automatically validates activity results. √ (if expected velocity does not match actual velocity, it modifies the plan expected end date accordingly; orders requirements by priority automatically)
   c) Supports collaboration and interactions that provide feedback. √ (integrates tracking information on completed items from the whole team)

2) *Affordance/Natural Mapping*
   a) Uses the user's language to describe practices and process activities. √ (assuming users are familiar with Scrum)
   b) Its workflow steps match the practices and process activities. √ (Planning and Tracking are supported naturally, if a Release plan is in place for the Release Burn-down to work)
   c) Is accessible to the people doing the job and they have the appropriate privileges. √ (simple authorization scheme, owner, member, viewer roles).
   d) Provides support to record rationale and other contextual information. X (Very little in the way or rationale or contextual information beyond a general description of each user story)

3) *Matching Conceptual Models*
   a) Supports only practices and activities that are necessary for the process, practice or methodology. √ (Supports counting story points for bugs, but strongly discourages it).
   b) Is designed for one specific practice. √ (Generally well aligned with Scrum)
   c) Supports cohesively multiple activities when they are deeply interwoven. √ (Planning and Tracking)
   d) Provides flexible customization for better alignment to process and practices.~ (Very limited, charts in particular)

4) *Tolerate Mistakes*
   a) Does not make judgmental assertions about the meaning of a practice or process activity.~ (Velocity changes in recent iterations affect heavily and automatically the planned outcome of the project, but this is usually good practice after the first few iterations)
   b) Provides means to establish flexible thresholds for status, alarms and notifications. X (None)

5) *Force Function*
   a) Supports rules for automatic recognition of inconsistencies. √ (Plans and predicts schedule automatically based on simple velocity metric)
   b) Warns or even sets hard restrictions when practices reach unhealthy limits. X
   c) Does not promote poor practice. X (Charts and reports are unwieldy)

TABLE II. PIVOTAL TRACKER EVALUATION MATRIX

Process Activity/ Practice	***Feedback***	***Affordance/ Natural Mapping***	***Matching Conceptual Models***	***Tolerate Mistakes***	***Force Function***
Requirements Management	High	Medium	High	Medium	Low
Project Planning	High	High	High	Medium	Low
Project Tracking	HIgh	High	Medium	Medium	Medium

The results in Table II show overall medium-high scores for Pivotal Tracker. This fits the fact that it is a tool targeted to several processes. In other words, fitting multiple user´s processes is more challenging for the tool since it spans a wider range of activities and practices. It still evaluates as a good candidate for successful implementation, but the insights provided by the checklist should be taken into account to reduce risks during tool implementation.

V. CONCLUSION AND FUTURE WORK

The purpose of this paper was to validate the applicability of our usability-based framework for analyzing tool fitness to the user's process and practice (not to validate its results). The principles selected and the criteria proposed to evaluate their concrete application allowed us to conduct the evaluations without obstacle, and the framework did not turn up any inconsistencies during the process. Nonetheless, there is significant overlap between some of them. For example, a tool that has a Matching Conceptual Model will usually have Natural Mapping, and both Feedback and Tolerate Mistakes are related to metrics, although with different perspectives. Overall, the criteria and usability terminology have been effective in supporting the discussion and description of tool fitness to process and practice. One valuable output of the evaluation that complements other evaluation methods based on tool requirements is the qualitative comments produced for each checklist item, which might help implementors to assess areas of risk during the implementation process (e.g., for Pivotal Tracker, item 2.b highlights the need to define release items in the tool if we need to use the release burn-down chart).

Future work includes formal experimentation with tools to validate evaluation results, refinement of principles and criteria, peer feedback and expert validation of the framework, refinement of the evaluation template structure, an in-depth study of the conceptual issues explored in this paper, and the application of the framework to the evaluation of fitness between organizations and practices and processes.

We have learned that the framework is coherent and a viable subject of research, and that the resonance in terminology between usability and process and practice that inspired this work holds in the practical application to the real tools evaluated.

Acknowledgment

Universidad Nacional de La Matanza and Universidad Nacional de Tres de Febrero fund this research.

References

[1] A. Guha Biswas, R. Tandon, and A. Vaish, "A case tool evaluation and selection methodology," International Journal of Strategic Information Technology and Applications (IJSITA), 4(2), 2013, pp. 48-60, doi:10.4018/jsita.2013040104.

[2] E. Miranda, M. Berón, G. Montejano, M. J. Pereira, and P. Henriques, "NESSy: a new evaluator for software development tools," In 2nd Symposium on Languages, Applications and Technologies (SLATe'13), Faculdade de Ciências da Universidade do Porto, 2013, pp. 21-38, ISBN 978-3-939897-52-1

[3] E. Anjos and M. Zenha-Rela, "A framework for classifying and comparing software architecture tools for quality evaluation." In: B. Murgante, O. Gervasi, A. Iglesias, D. Taniar, B.O. Apduhan, Eds. ICCSA 2011, Part V. LNCS, vol. 6786, pp. 270–282, Springer, Heidelberg, 2011.

[4] I. Dalmasso, S.K. Datta, C. Bonnet, and N. Nikaein, "Survey, comparison and evaluation of cross platform mobile application development tools," Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International, 1-5 July 2013, pp. 323-328, doi: 10.1109/IWCMC.2013.6583580.

[5] Center for Assured Software, National Security Agency, US, "CAS Static Analysis Tool Study – Methodology", December 2012 [Online]. Available from: http://samate.nist.gov/ 2014.05.11.

[6] J. F. Cochran and H.N. Chen, "Fuzzy multi-criteria selection of object-oriented simulation software for production system analysis," Computers & Operations Research, vol. 32, issue 1, January 2005, pp. 153-168, ISSN 0305-0548, doi:10.1016/S0305-0548(03)00209-0.

[7] X. Franch and J.P. Carvallo, "Using quality models in software package selection," IEEE Software, January-February 2003, pp. 34–41.

[8] R. Firth, V. Mosley, R. Pethia, L. Roberts Gold, and W. Wood. "A Guide to the Classification and Assessment of Software Engineering Tools" (CMU/SEI-87-TR-010). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1987. [Online]. Available from: http://resources.sei.cmu.edu/ 2014.05.11.

[9] S. Comella-Dorda, J.C. Dean, E. Morris, and P. Oberndorf, "A Process for COTS Software Product Evaluation," Springer-Verlag, ICCBSS 2002, LNCS 2255, pp. 86–96.

[10] M. Morisio and A. Tsoukias, "IusWare: a methodology for the evaluation and selection of software products," IEEE Proceedings Software Engineering, 144 (3), 1997, pp. 162-174.

[11] G.A. Lewis and L. Wrage, "A process for context-based technology evaluation: examples for the evaluation of Web services technology," Commercial-off-the-Shelf (COTS)-Based Software Systems, 2006. Fifth International Conference on, Feb. 2006, pp. 13-16, doi: 10.1109/ICCBSS.2006.2.

[12] A.S. Jadhav and R.M. Sonar, "Evaluating and selecting software packages: A review," Information and Software Technology, vol 51.3, March 2009, pp. 555-563, ISSN 0950-5849, doi:10.1016/j.infsof.2008.09.003.

[13] M.A.D. Storey, S. E. Sim, and K. Wong, "A collaborative demonstration of reverse engineering tools," ACM SIGAPP Applied Computing Review, vol. 10 issue 1, Spring 2002, pp. 18-25.

[14] J.S. Brown and P. Duguid, The Social Life of Information, Harvard Business School Press, 2000.

[15] I. Gat, "How we do things around here in order to succeed", Workshop, Agile 2010 Conference, Orlando, August 2010.

[16] A. Cockburn, "What the Agile Toolbox Contains", Crosstalk Magazine, November 2004.

[17] J. Nielsen, Usability Engineering, Morgan Kauffman Press, 1993.

[18] D. Norman, The Design of Everyday Things, Basic Books, 1988.

[19] K. Beck, Extreme Programming Explained, Embrace Change, Addison-Wesley Professional, 1999.

[20] P. Clements et al, Documenting Software Architecture, Views and Beyond, SEI Series in Software Engineering, Addison-Wesley Professional, 2003 (second edition 2010).

[21] T. Mayer, "Simple Scrum", Agile Anarchy Blog, [Online]. Available from: http://agileanarchy.wordpress.com/2009/09/20/simple-scrum/, 2014.05.11.

[22] M. Poppendieck, "Team Compensation", Better Software, July/August 2004, [Online]. Available from: http://www.poppendieck.com/pdfs/Compensation.pdf 2014.05.11.

[23] K. Kawaguchi, et al, "Meet Jenkins", Jenkins Web Site, [Online]. Available from: https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins, 2014.05.11.

[24] Pivotal Labs, "Pivotal Tracker Features", Pivotal Tracker Web Site, [Online]. Available from: http://www.pivotaltracker.com, 2014.05.11.