

Software Metrics for Agile Software Development

Reiner R. Dumke, Andreas Schmietendorf, Martin Kunz, Konstantina Gorgieva

Software Engineering Group, University of Magdeburg, Germany
dumke@ivs.cs.uni-magdeburg.de

Abstract

Agile Software Development Methods are nowadays wide spread and accepted. From the Software Measurement point-of-view not all metrics and methods from conventional lifecycle models can be used without adaptation. Distinct techniques in agile software development like effort estimation or refactoring needs new approaches and quality models in the area of software measurement.

Therefore, this paper discuss cost estimation problems and approaches for agile software development and maintenance. Furthermore this paper describes a quality model, distinct metrics and their implementation into a measurement tool for quality management in agile software development.

1. Introduction

Many technological ambitious products were designed with new complex functionality. The demand for functions establishes a need for new software requirements to deliver new functionality. Due to the fast alteration and the high cost of change in the late life cycle phases the agile software development method becomes more important in this field of application. Agile software development methods like eXtreme programming try to decrease the cost of change and therewith reduce the overall development costs. The different cost of change for agile software development in comparison with traditional software development according to the project progress is shown in Figure 1.

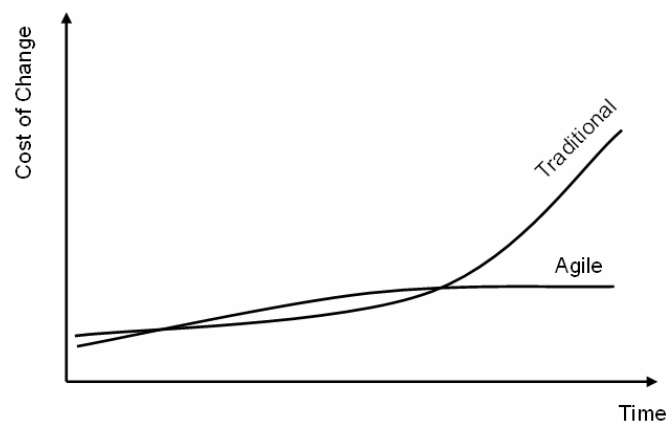


Figure 1: Cost of Change compared to development method [1]

To realize this totally different project characteristic, the agile software development has established a different type of product life cycle in comparison with traditional life cycle models like waterfall-model or V-Model.

The major element of the XP life cycle is the “Iteration”. The iteration is recurring event in which an actual version is edited for example by adding additional functionality, correcting errors or removing unnecessary functionality. The result of each iteration is validated through an acceptance test. In XP the duration of each iteration is very short (1-4 weeks) in comparison with traditional software development. So the planning phase is also very short and mainly realized by the definition of tasks. The general characteristics of the XP life cycle are shown in Figure 2.

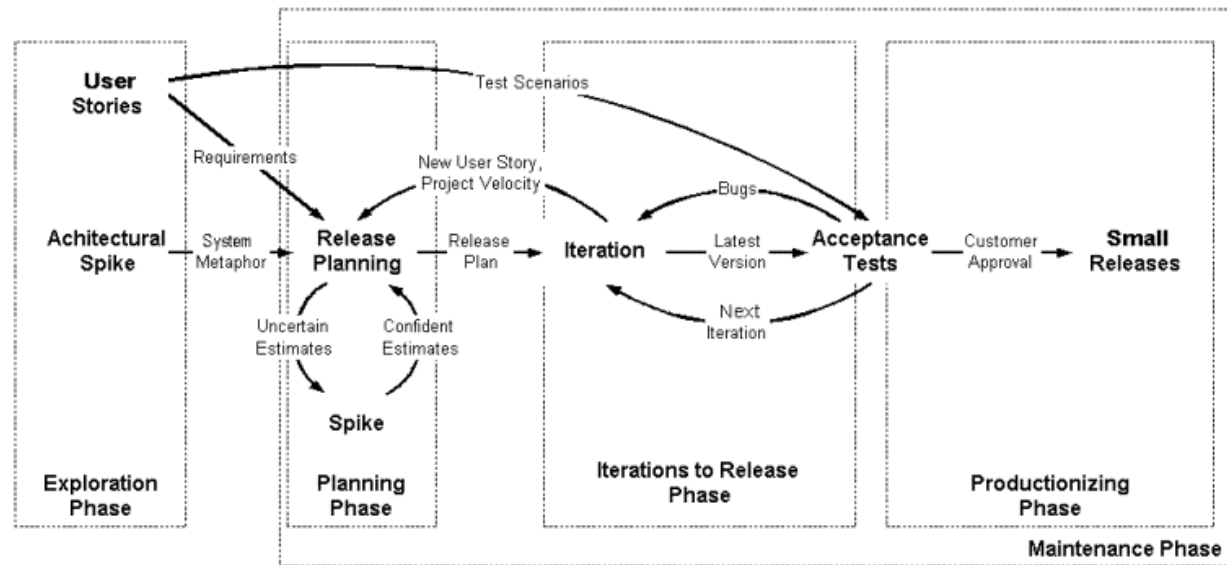


Figure 2: eXtreme Programming life cycle [Wells 2007]

The current approaches in order to analyze and measure the different aspects of product, process and resources in the agile development process could be characterized as following:

- Evaluation of new systems with agile methods for embedded systems [Cordeiro 2008] and distributed systems [Sureshchandra 2008]
- Analyzing agile development for special software methodologies like globally distributed system development (GDSD) [Lee 2006], Web site quality for e-Commerce [Rico 2008], model driven development (as AMDD [Martini 2008]), and agile requirements engineering [Cao 2008]
- Characterizing detailed aspects of agile methodology like time management [Hazzan 2007], business impact [Klein 2008], large distributed agile team [Cannizzo 2008], decision making [Moe 2008], risk management [Nyfjord 2008], offshore development [Cottmeyer 2008], pair programming [Vanhanen 2007], and communication [Korkala 2007]
- Building new quality models for agile development such as QMOOD (Quality Models Over Software Development) [Roden 2007], ROA (Real Options Analysis) measurements [Racheva 2008], XPMM (XP maturity model) [Altaraweh 2008], and SWOT (Strengths-Weaknesses-Opportunities-Threats) [Shahir 2008]
- Analyzing quality aspects of agile development considering ISO 9001 [McMichael 2007], defining SQA [Mnkandla 2006], and considering performance [Shu 2007]

In the preliminary roadmap for empirical research on agile software development [Dingsoyr 2008] that there are any weaknesses in the maturity, coverage, understanding and impact of measurement of the agile paradigm. This paper is addressed to the two essential management impacts: the system quality assurance and the effort estimation.

2. Cost estimation for Agile Software Development

The consideration of the effort estimation activity within agile software development methodologies was also investigated by [Buglione 2007]. He shortly describes the estimation approaches used in various agile software development methods. In the following representations, we want to concentrate on the eXtreme Programming (in short XP) approach. For this, available approaches for the effort estimation will be shown. In addition, further influence criterions will be analyzed.

As mentioned earlier within this paper, feedback and change activities are central principles of the XP-methodology. Feedback and changes are important for effort estimation activities too, as Gilb* noticed. He considers the budget-related requirements of the software development with the following statement: "Accurate estimation is impossible for complex technical projects, but keeping to agreed budgets is still possible using feedback and change."¹

The activities of feedback and change are key drivers to fulfill budget restrictions of a software development project. An exact estimation of the corresponding effort at beginning of a complex development project is not reachable as [Gilb 2006] mentioned. He sees the reasons for this in:

- Inaccurately defined requirements,
- Missing experience background with the effort of completed projects,
- New projects differ often from realized projects.

In order to reach a successful realization of technical requirements, he proposes to combine the effort estimation with the following "principles of resource control" [Gilb 2006]:

- Risk principles (under consideration of: DRIVERS, EXPERIENCE, ARCHITECTURE, STAFF, SENSITIVITY)
- Control principles (under consideration of: LEARN SMALL, LEARN ROOT, PRIORITIZE CRITICAL, RISK FAST, APPLY NOW)

While the risk principles show the potential causes of inadequate effort estimation, the control principles accept possible solution alternatives. Similar to the agile methodologies in the context of the software development [Gilb 2006] puts the person-referential aspects (acceptance of the learning and profits of experiences) as well as the guarantee of feedback (short iterations and small increments) in the foreground of a successfully executed software development. For the effort estimation of agile executed software development projects [Steindl 2005] gives the following recommendation: "Don't estimate too far into the future, if the future is unclear!"

If effort estimations are difficult, projects should work more with feedback and change. Future methods of the effort estimation must take this challenge into account. That means effort estimations must be executed frequently, but within a very short time. The established methods of the effort estimation mostly do not possess this quality.

For a first evaluation we established a questionnaire with 17 questions. The questions consider aspects of agile methodologies and the effort estimation, too. The complete questionnaire is available inside the appendix of this contribution. The questionnaire was sent to approximately 40 known experts in Germany [Schmietendorf 2008].

The following section contains some selected results from a first analysis. However they show a first trend, but a statistical security can not be guaranteed yet.

¹ Source: Gilb, T.: Estimation or Control? – Thesis paper, URL: <http://www.dasma.org>, December 2007

Which agile methodologies do you know?

5 participants of the survey didn't know any methods to the agile software development. After all, 12 participants of the survey knew agile methodologies. The knowledge of concrete methods can be taken from the figure 5.

For how many years are projects executed under consideration of an agile procedure inside your company?

- No experience → 29%
- Less than 2 years → 18%
- Less than 5 years → 18%
- More than 5 years → 35%

The participants with less than 2 years experience came all from academic facilities. All other participants came from industrial businesses. The average sizes of software development projects inside companies with more than 5 years experience was 5 to 30 developers.

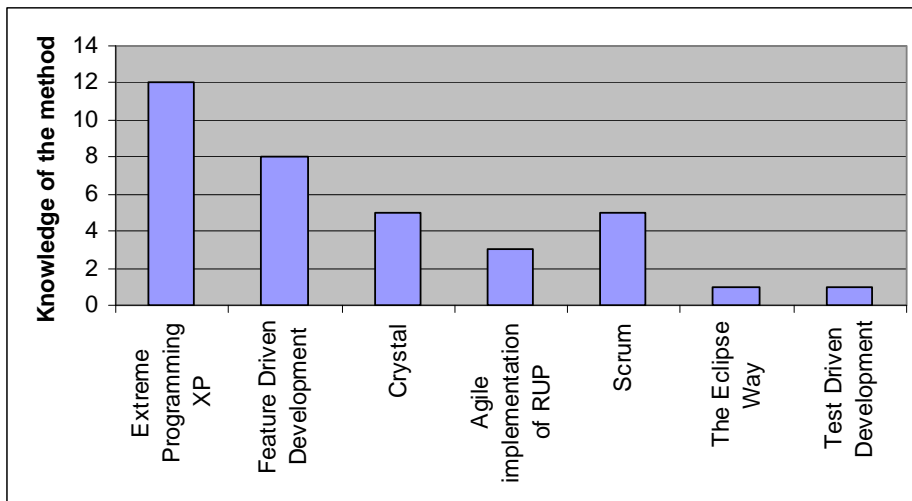


Figure. 3: Knowledge of agile methods

How important is the effort estimation in agile operated software projects?

- Highly important → 43%
- Important → 36%
- Necessary → 21%

How important is it to carry out a risk evaluation during the planning game?

- Highly important → 43%
- Important → 29%
- Necessary → 7%
- Less necessary → 21%

Which temporal and personal effort should be caused by an effort estimation method in agile accomplished projects?

The question was answered by 11 experts. However, 6 experts could not make any statements about the necessary effort. The replies of this question brought the following results:

Effort for the estimation: min.1 person day, max. 2 person day,

Effort estimation is a cyclic activity (weekly task),

The effort should be fixed project-dependent,

The effort should be proportionally towards the total expenditure,

5 % to 10 % of the whole development effort.

In your mind, is there a demand on adaptation or new development for an effort estimation method which is applicable in agile accomplished projects?

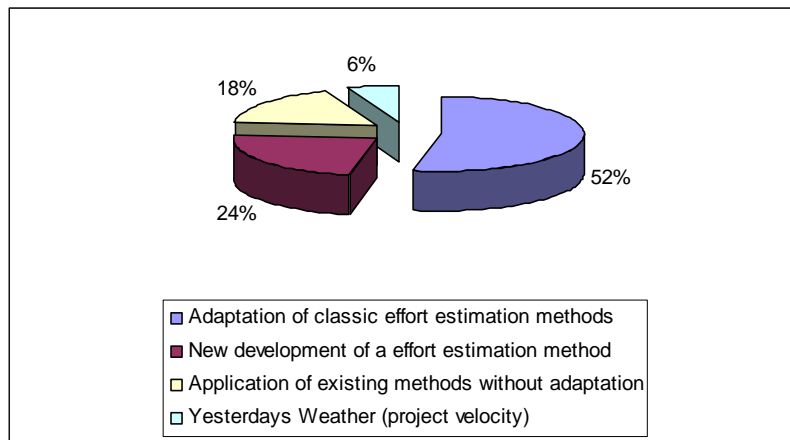


Figure 4: Effort estimation methods in agile projects

This question was difficult to answer for most of the participants. The validations of the answers require the consideration of another question about the known effort estimation methods. For an elaborate analysis, this question must be formulated more precisely. The experienced developers of agile software projects consider the activities inside the planning game as effort estimation method too. The concept of “Yesterdays Weather” use the number of story points that are completed in the last iteration. Under consideration of this experience, this number will be used as predictor for the next iteration.

3. Product Measurement for Agile Software Development

Besides the different life cycle, agile software development is distinguished by one major element: the refactoring. Refactoring means the change of source-code and software-design without altering the functionality or external behavior of software [Beck 1999]. In XP, a refactoring cycle is proposed after each iteration to adjust the software product. The main goal is to ensure product quality attributes [Fowler 2000].

Therefore the refactoring cycle has essential importance for agile software development, so it is a fundamental toehold to support agile software development. Since traditional standards for software product quality like ISO/IEC 91216 [ISO9126] are not designed for agile software development, a new set of metrics, thresholds and

measurement artifacts which are suited for agile software development and especially for the refactoring, have to be identified. After describing the quality model consisting of selected metrics and thresholds in the following chapter an implementation of this model into an Eclipse Plug-In [Eclipse] to support the refactoring cycle in agile software development is presented.

To support agile software development and especially refactoring, mainly source-code based product metrics are beneficial to increase quality and productivity [Goodman 2004]. Primarily internal quality attributes have to be ensured to control the source-code quality and to evaluate refactoring steps [McConnell 2004]. The goal of our approach is to combine refactoring with software measurement to give advice about the following aspects:

- Appropriate point in time for necessary refactoring steps
- Significance of a refactoring step and the apparent quality effect
- Side effects of refactoring steps

With these three aspects one can ensure quality along refactoring steps. The metrics should deliver indices for distinct refactoring steps and they should be easily interpretable. The measurement results should be a trigger or activator for useful refactoring steps and they should avoid quality loss through refactoring steps.

There are various possibilities to define a set of metrics for a distinct information need or for specific conclusions. As a traditional top-down approach the Goal Question Metric-Method [Basili 1984] can be used. Additionally, metrics and calculation methods known from agile environment enhance these possibilities [McConnell 2004] [Henderson 1996] [Fowler 2000]. One major base was the metrics suite from Chidamber & Kemerer [Chidamber 1993]. Source code metrics among others are: Number of Name-Parts of a method (NNP), Number of Characters (NC), Number of Comment-Lines (CL), Number of Local Variables (NLV), Number of Created Objects (NCO), and Number of Referring Objects (NRO). [Henderson 1996][Bloch 2001]

For the interpretation and analysis the value according to different thresholds are normalized. The mapping between the normalized values and the linguistic quality terms are shown in the following table:

Linguistic term	Normalized value
Best/Excellent	1.00
Very Good	0.82
Good	0.66
Average	0.50
Poor	0.32
Very Poor	0.16
Worst	0.00

Table 1: Mapping between values and linguistic terms

The thresholds for the normalization are defined on the basis of empirical data but they can easily be modified according to distinct projects specifications. This mapping defines the general quality model. The same mapping for all metrics is used, but this is just the standard setting and editable.

Beside the described adoption of well known metrics, the analysis of additional features in every executable product after each iteration is necessary and therefore the Running Tested Features Metric (RTF) can be used. The RTF Metric was introduced in [Jeffries 2004] and it is defined as the number of features which passed their acceptance test. In this way RTF delivers a possibility to quantitatively evaluate different iteration according to their implemented functionality.

Subsuming the metrics are classified in three categories:

- Method-level (e.g. Number of Parameters, Lines of Code)
- Class-level (e.g. Coupling between Objects, Number of Children, Depth of Inheritance Tree)
- Package-level (e.g. Cycle Count of Dependency Graph)

In agile software development and especially in the refactoring the early observation of quality is essential to keep the software stable throughout the evolutionary development process. Due to the specific characteristic of each distinct software development project it is not useful to define strict thresholds. Instead it is important to keep the limits and the interpretation of the measurement values as simple and flexible as possible.

It is very important to accommodate this requirement in the realization of metrics for agile software development.

4. “UnitMetrics” – Measurement Tool

To support agile software development at the origin a tool was implemented as a Plug-In for an Integrated Development Environment (IDE). Eclipse, because of the preconditions and its open source character [Eclipse], was chosen. Especially because of the many iterations’ of a product until it reaches final status the integration in a development environment instead of a stand-alone realization is recommendable.

The project was realized as an open source project and published under sourceforge.net [UnitMetrics]. Since August 2007 the plug-in was downloaded over 300 times but empirical information about the usage of the tool is not yet available [Kunz 2008].

The general goal was to create a measurement tool which expands the Eclipse-Development-Environment to provide source-code analysis by the use of appropriate metrics.

Four major features should be supported:

- Continuous analysis
- Fundamental interpretation
- Interactive visualization
- Extensibility

Another important implemented feature is the snapshot concept. A snapshot stores the measured attributes of source code and allows the comparison of actual values with previous ones. And in this way it allows a trend analysis and estimation about future changes.

Especially XML-based import and export of snapshots provides a lot of additional possibilities for analyzing measurement data. For example one can use the sequence of snapshots for empirical analysis of the development process.

Besides the described features and characteristics the UnitMetrics Plug-In provides four major functionalities: presentation of metric values, normalized metric values and mapping according to threshold of distinct quality models, visualization of package and type dependencies, and package distance diagram.

In Figure 3 one can see the measured values for a chosen snapshot. The metrics are calculated at method level and without normalization. The measurement value table presents the core measurement results without any interpretation or normalization.

One has to recognize that without normalization the interpretation is very difficult due to less clarity. To give the user a more graphical representation of the normalized values are established in a “star” concept where three stars means excellent and zero stars means worst (in half star steps according to the mapping in table 1).

Figure 4 shows the same results as shown in Figure 3 in a normalized way by using the “star” concept. This normalization and mapping using the thresholds of a distinct quality model for the specific project provides more clarity and a better presentation of measurement results.

Type	Package	Project	NNP	LOC	CBO	NRO
BasicGraphUI	org.jgraph.plaf.basic	org.jgraph	3.0	809.0		5.0
GraphLayoutCache	org.jgraph.graph	org.jgraph	3.0	667.0	45.0	18.0
JGraph	org.jgraph	org.jgraph	2.0	511.0	71.0	28.0
EdgeView	org.jgraph.graph	org.jgraph	2.0	436.0	29.0	12.0
DefaultGraphModel	org.jgraph.graph	org.jgraph	3.0	435.0	47.0	8.0
EdgeRenderer	org.jgraph.graph	org.jgraph	2.0	313.0	46.0	2.0
GraphConstants	org.jgraph.graph	org.jgraph	2.0	294.0	35.0	24.0
GraphEd	org.jgraph.example	org.jgraph	2.0	270.0	58.0	4.0
EdgeHandle	org.jgraph.graph	org.jgraph	2.0	252.0		1.0
RootHandle	org.jgraph.plaf.basic	org.jgraph	2.0	235.0	44.0	1.0
VertexView	org.jgraph.graph	org.jgraph	2.0	184.0	17.0	7.0
DefaultGraphSelection...	org.jgraph.graph	org.jgraph	4.0	160.0	32.0	3.0
JGraphIconView	org.jgraph.example	org.jgraph	4.0	157.0	17.0	3.0
SizeHandle	org.jgraph.graph	org.jgraph	2.0	157.0	40.0	2.0
AbstractCellView	org.jgraph.graph	org.jgraph	3.0	133.0	32.0	15.0
AttributeMap	org.jgraph.graph	org.jgraph	2.0	130.0	21.0	29.0
DefaultGraphCellEditor	org.jgraph.graph	org.jgraph	4.0	113.0	34.0	4.0
GraphTransferHandler	org.jgraph.graph	org.jgraph	3.0	94.0	35.0	1.0
GraphModelEdit	org.jgraph.graph	org.jgraph	3.0	94.0		1.0
VertexRenderer	org.jgraph.graph	org.jgraph	2.0	91.0	35.0	2.0
ConnectionSet	org.jgraph.graph	org.jgraph	2.0	77.0	12.0	10.0

Figure 3: Measurement values for a snapshot [14]

Type	Package	Project	NNP	LOC	CBO	NRO
BasicGraphUI	org.jgraph.plaf.basic	org.jgraph	☆☆☆	☆☆☆		
GraphLayoutCache	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆		☆☆☆
JGraph	org.jgraph	org.jgraph	☆☆☆	☆☆☆		☆☆☆
EdgeView	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆		☆☆☆
DefaultGraphModel	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆		☆☆☆
EdgeRenderer	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆		☆☆☆
GraphConstants	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆		☆☆☆
GraphEd	org.jgraph.example	org.jgraph	☆☆☆	☆☆☆		☆☆☆
EdgeHandle	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆		
RootHandle	org.jgraph.plaf.basic	org.jgraph	☆☆☆	☆☆☆		☆☆☆
VertexView	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆		☆☆☆
DefaultGraphSelection...	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆	☆☆☆	☆☆☆
JGraphIconView	org.jgraph.example	org.jgraph	☆☆☆	☆☆☆	☆☆☆	☆☆☆
SizeHandle	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆	☆☆☆	☆☆☆
AbstractCellView	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆	☆☆☆	☆☆☆
AttributeMap	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆	☆☆☆	☆☆☆
DefaultGraphCellEditor	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆	☆☆☆	☆☆☆
GraphTransferHandler	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆	☆☆☆	☆☆☆
GraphModelEdit	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆	☆☆☆	☆☆☆
VertexRenderer	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆	☆☆☆	☆☆☆
ConnectionSet	org.jgraph.graph	org.jgraph	☆☆☆	☆☆☆	☆☆☆	☆☆☆

Figure 4: Normalized measurement values [14]

Another important set of views is represented by dependence graphs [Henderson 1996] [McConnell 2004]. A dependence graph displays the coupling between different elements. The UnitMetrics tool provides the visualization on type-(class) - and package-level.

Dependence graphs allow to display the netting of coupling and to identify strong components. A strong component contains all elements which are connected into a loop. Because of the fact that a goal of agile software development is to avoid loops or circles, identification is very useful. Figure 5 shows an example of a dependence graph on package level. As one can see nearly every package is connected to every other package which manifests a loop-problem. Loop-problem means that a connection from an origin to every other node by using just two paths exists. In this case the chosen architecture seems to contain substantial problems.

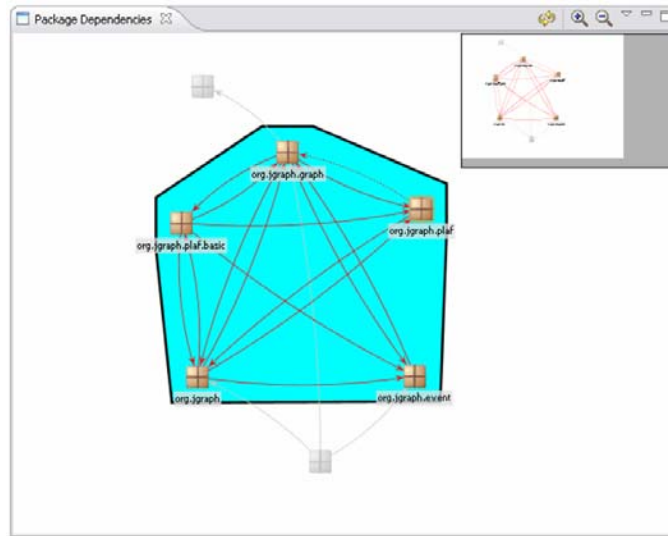


Figure 5: Dependence graph on Package-Level

Figure 6 shows the dependences on type-level and allows the further analysis of the discovered loops. At this point it stands out that the loops are not limited to interfaces (nodes marked with an “I”) but also obtains other components.

In both views strong components are illustrated by black bordered colored planes. For a more clear view the strong components were highlighted. Both views possess manifold possibilities to analyze the netting of dependencies (e.g. fade in and fade out of distinct node, selection of nodes according to the distance of a specific node).

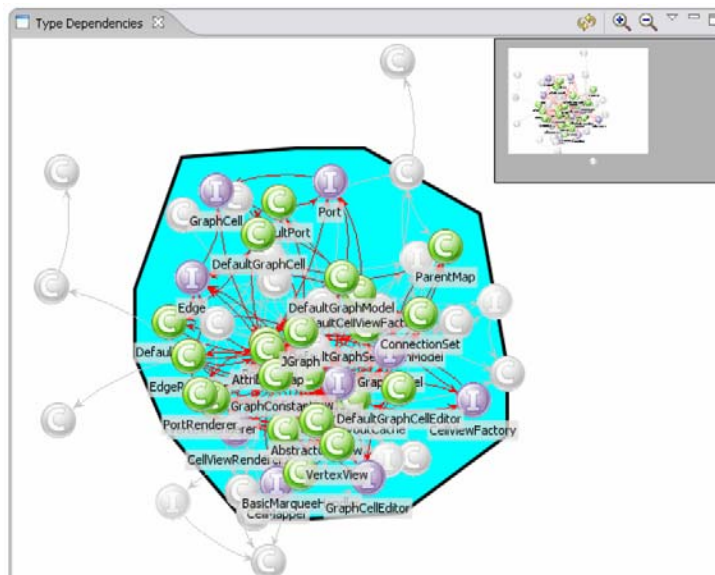


Figure 6: Dependence graph on Type-Level

Another important visualization is the distance to the main sequence of specific packages in the measurement values [Martin 203]. This Package Distance Diagram is shown in Figure 7. To increase the interpretability standard deviation and the double standard deviation is marked with dashed lines.

The distance of a package represents the balance between instability and abstractness. Packages approaching one are both abstract and stable (Abstractness=1; Instability=0) or concrete and instable (Abstractness=0; Instability=1). While the first case indicates a lack of design integrity, the second case represents useless design [Martin 2003].

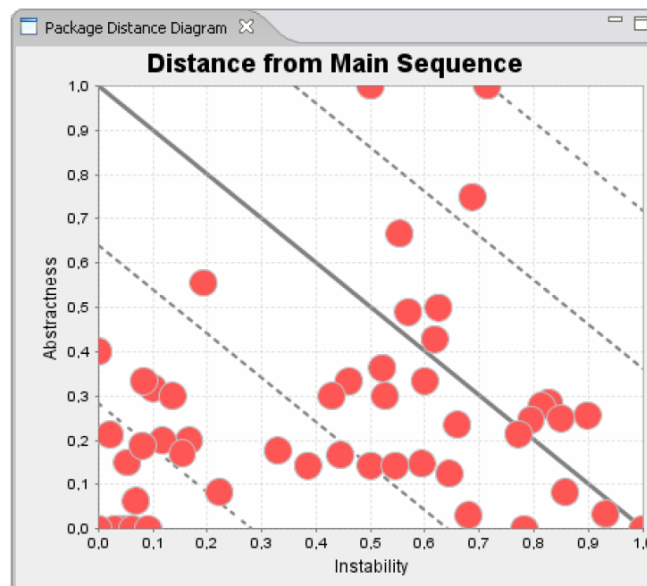


Figure 7: Package Distance Diagram

5 Conclusions and Future Work

This paper presents an approach to support agile software development and especially the refactoring by the use of software measurement. The selection of the metric is also important to the outcome of the refactoring process. Future implementations may also include different types of metrics, e.g. a resource utilization metric, to improve the stability and speed of the program.

An empirical analysis about the usage and impacts of the tool can be useful to identify the influence of measurement values for the agile software development process. The existing Implementation enhanced the Eclipse IDE to support agile software development. Besides Eclipse, there are some other widespread IDE's. On the basis of our reference implementation one can simply adapt the tool to other IDE's. At the moment the tool supports only Java as a programming language. The enhancement to other programming languages can increase practical benefit.

References

- [Altarawneh 2008] Altarawneh, H.; Shiekh, A. E.: *A Theoretical Agile Process Framework for Web Applications Development in Small Software Firms*, Proc. of the 6th SERA Conference 2008, pp. 125-132
- [Basili 1984] Basili, V.; Weiss, D.: *A Methodology for Collecting Valid Software Engineering Data*, IEEE Transaction on Software Engineering, 10, pp. 728-738., 1984
- [Beck 1999] Beck, Kent: *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999
- [Bloch 2001] Bloch, Joshua: *Effective Java: Programming Language Guide*, Addison-Wesley, 2001
- [Buglione 2007] Buglione, L.; Abran, A.: *Improving Estimations in Agile Projects: some thoughts and suggestions*, in Proc. Software Measurement European Forum - SMEF07, Rom/Italy 2007

- [Cannizzo 2008] Cannizzo, F.; Marcionetti, G.; Moser, P.: *Evolution of the Tools and Practices of a Large Distributed Agile Team*, Proc. of the AGILE 2008 Conference, pp 513-518
- [Cao 2008] Cao, L.; Ramesh, B. *Agile Requirements Engineering Practices: A Empirical Study*, IEEE Software, Jan/Feb 2008, pp. 60-67
- [Chidamber 1993] Chidamber, Shyam; Kemerer, Chris: *A Metrics Suite for Object Oriented Design*, M.I.T. Sloan School of Management, 1993
- [Cordeiro 2008] Cordeiro, L. et al.: *A Platform-Based Software Design Methodology for Embedded Control Systems: An Agile Toolkit*, Proc. of the ICECS 2008, pp. 408-417
- [Cottmeyer 2008] Cottmeyer, M.: *The God and Bad of Agile Offshore Development*. Proc. of the AGILE 2008 Conference, p. 362-367
- [Eclipse] Eclipse Foundation (<http://www.eclipse.org>), March 2007
- [Fowler 2000] Fowler, Martin: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 2000
- [Gilb 2006] Gilb, T.: Estimation or Control? – Thesenpapier, URL: <http://www.dasma.org/>, abgerufen im Juni 2007
- [Goodman 2004] Goodman, Paul: *Software Metrics: Best Practices for Successful IT Management*, Rothstein Associates, 2004
- [Hazzan 2007] Hazzan, O.; Dubinsky, Y.: *The Software Engineering Timeline: A Time Management Perspective*, Proc. of the ICSTE 2007, pp. 95-103
- [Henderson 1996] Henderson-Sellers, Brian: *Object-oriented Metrics: Measures of Complexity*, Prentice Hall. 1996
- [ISO 9126] ISO 9126-1: 2001, *Software engineering –Product quality -- Part 1: Quality model*, International Organization for Standardization, Geneva, 2001
- [Jeffries 2004] Jeffries, Ron: *A Metric Leading to Agility*, 2004
- [Kersten 2007] Kersten, Martin, *Unterstützung der agilen Softwareentwicklung durch geeignete Softwaremetriken* Masterthesis, University of Magdeburg, 2007
- [Klein 2008] Klein, H.; Canditt, S.: *Using Opinion Polls to Help Measure Business Impact in Agile Development*, Proc. of the BIPI 2008, Leipzig, pp. 25-30
- [Korkala 2007] Korkala, M.; Abrahamsson, P.: *Communication in Distributed Agile Development: A Case Study*, Prof. of the Euromicro SEAA Conference 2007, pp. 1-8
- [Kunz 2008] Kunz, M.; Zenker, N; Menck, S.; Dumke, R.: *UniMetrics – A Tool support Refactoring in Agile Software Development*, Proc. of the WORLDCOMP 2008, Las Vegas, pp. 208-214
- [Lee 2006] Lee, O. et al.: *Aligning IT Components to Achieve Agility in Globally Distributed System Development*, Comm. of the ACM, 49(206)10, pp. 49-54
- [Martin 2003] Martin, Robert C.: *Agile Software Development: Principles, Patterns, and Practices*, Prentice Hall, 2003
- [Martini 2008] Martini, P.; Kaiser K.; Miksch, S. : *Easing the Formalization of Clinical Guidelines with a User-tailored, Extensible Agile Model Driven Development (AMDD)*, Proc. of the ISCMS 2008, pp.120-126
- [McConnel 2004] McConnell, Steve: *Code Complete: Second Edition*, Microsoft Press, 2004
- [McMichael 200] McMichael, B.; Lombardi, M.: *ISO 9001 and Agile Development*. Prof. of the AGILE 2007, pp. 1-4
- [Mnkandla 2006] Mnkandla, E.: *Defining Agile Software Quality Assurance*, Proc. of the ICSEA 2006, pp. 72-78
- [Moe 2008] Moe, N. B.; Aurum, A.: *Understanding Decision-Making in Agile Software Development: a Case Study*, Proc. of the Euromicro SEAA 208, pp. 216 – 223

- [Nyfjord 2008] Nyfjord, J.; Kajko-Mattson, M.: *Outlining a Model Integrating Risk Management and Agile Software Development*, Proc. of the 34th Euromicro SEAA 2008, pp. 476-483
- [Rachea 2008] Racheva, Z.; Daneva, M.: *Using Measurement to Support Real-Option Thinking in Agile Software Development*, Proc. of the APSO 2008, Leipzig, pp. 15-18
- [Rico 2008] Rico, D. F.: *Effects of Agile Methods on Website Quality for Electronic Commerce*, Proc. of the 14th HICSS 2008, pp. 1-9
- [Roden 2007] Roden, P.; Virani, S.; Eitzkorn, L. H.; Messimer, S.: *An Empirical Study of the Relationship of Stability Metrics and the QMOOD Quality Models Over Software Development Using Iterative or Agile Software Processes*, 7th IWCAM 2007, pp. 171-179
- [Schmietendorf 2008] Schmietendorf, A.; Kunz, M.; Dumke, R.: *Effort Estimation for Agile Software Development Projects*, Proc. of the SMEF 2008, Milan, pp. 46-51
- [Shahir 2008] Shahir, H. Y.; Daneshpajouh, S.; Ramsin, R.: *Improvement Strategies or Agile Processes: A SWOT Analysis Approach*, Proc. of the SERA Conference 2008, pp. 221-227
- [Shu 2007] Shu, X.; Maurer, F.: *A Tool for Automated Performance Testing of Java3D Applications in Agile Environments*, Proc. of the ICSEA 2007, pp. 1-6
- [Steindl 2005] Steindl, C.; Krogdahl, P.: *Estimation in Agile Projects*, IBM Academy of Technology Best Practices in Project Estimation Conference, IBM Corporation 2005
- [Sureshchandra 2008] Sureshchandra, K.; Shrinivasaadhani, J.: *Adopting Agile in Distributed Development*, Proc. of the ICGSE 2008, pp. 217-224
- [UnitMetrics] <http://sourceforge.net/projects/unitmetrics/>, July 2007
- [Vanhanen 2007] Vanhanen, J.; Korpi, H.: *Experience of Using Pair Programming in an Agile Project*, Proc. of the 40th HICSS 2007, pp. 1-10
- [Wells 2007] Wells, Donn: *Extreme Programming: A gentle introduction*, <http://www.extremeprogramming.org>, March 2007