

Arranging defect management in an agile development process

Matti Kokkola

Software Business and Engineering institute

Helsinki University of Technology

Matti.Kokkola@iki.fi

Abstract—Defect tracking is used to manage defects and to gather data for process improvement. Defect analysis is used to spot weaknesses of applied quality assurance methods and to make the organization learn from mistakes it has made.

In agile methodologies, the whole development team is responsible for the product quality. There are no dedicated testing team doing its own job separately from the actual development work. This fact suggests that feature management and defect management should not be separated, but they should be handled with the same process. In this research, Scrum is used as a reference model for an agile development model.

A first version of defect tracking arrangement for the case company is presented and it is evaluated by interviewing company's personnel. On basis of the evaluation, the next version of the defect tracking arrangement is suggested.

Index Terms—Defect management, defect tracking, bug tracking, Scrum, Agile processes, process improvement

1. INTRODUCTION

THE aim of this research is to describe arrangements made to establish defect management in a company applying a Scrum based software development process.

Defect tracking is used to control life-cycle of bugs and other quality issues found from a software. Its main benefit is to make sure that all reported defects are somehow handled: either fixed or explicitly ignored. Defect tracking is also used to gather data for software process improvement actions. Defect analysis is used to find weaknesses of established quality assurance. It can also be used to judge readiness of the software.

Defect management is very close to the feature and change management. A defect can become a feature request, if its resolution requires new

functionality or architectural refactoring and thus changes in the current design. From the viewpoint of an application developer, a defect and a feature are equal in most of the cases: both of them need some coding work. This analogy suggests that feature management and defect tracking can and even should be done in a centralized manner, not separately. This is more detailed described in Section 3.4.

Scrum is an iterative, incremental process for developing any product or managing any work. It produces a potentially shippable set of functionality at the end of every iteration. (Schwaber and Beedle, 2001) In Scrum there usually is no dedicated quality assurance team, but the development team itself is responsible for the quality of its own work. This fact even more encourages to handle defects and feature requests in the same manner, as the same team handles both of them.

This research concentrates on combining two worlds: agile Scrum model and traditional defect tracking. The research is conducted as a combination of literature study and action research in a case company. The case company's current situation is that they do not have any established defect tracking system. The company has applied Scrum based development process for six months now.

1.1. Objectives of the study

Objectives of this study are as follows:

- 1) To find out, what are the benefits of defect management?
- 2) To find out, how defects are handled in Scrum model?
- 3) Arranging an initial defect tracking system in the case company according to the objectives set together with the company.

- 4) Evaluate the made arrangement and give the first improvement proposal

1.2. Research Methods

The first part of the research is conducted as a literature study. The aim is to fulfill objectives 1 and 2. The results of the fulfillment of the objective 1 are also used to motivate employees of the case company and as an input to the construction phase of this study.

Scrum literature is studied to find out how the basic Scrum model takes care of defects. The presented construct should be seamlessly integrated with the Scrum model currently used by the case company and thus previous understanding about Scrum is needed.

Since the research should be conducted in less than two months and the evaluation is to be done during two weeks, we are not able to gather enough data to execute thorough evaluation of the suggested defect tracking model. Thus the empirical evaluation part of the study will consist of interviews of company personnel.

The interview aims to find out how the new defect management model fulfills objectives set for it and what steps should be taken to further improve it.

As the case product is rather mature, the amount of defect reports occurred during the evaluation part will most probably be minimal. Real life situations will be simulated by feeding the system with some old defects found from the defect tracking database of the case company.

1.3. Scope

In the scope of this study, we will not discuss about defect classification methods and agility criterias in general. Scrum model used as a base model in the context of this study is expected to be agile.

Defect analysis itself is not in the scope of this research. However, defect analysis is discussed when motivation for defect tracking is given.

The case company is a small sized software company where Scrum based development model has been used for six months. The company develops software products of its own as well as bespoke software for its customers. The company has 20 employees consisting mainly of software developers and 4 marketing/sales/admin persons.

Currently the case company does not have any structured or managed way to gather defect data. Defects are fixed as they occur and defect data is kept in developers' personal ToDo-lists.

The case product is a web-based messaging application aimed for small and medium sized enterprises. Its first version was released in summer 2002. The latest version was released in the second half of year 2003. It consists of circa 50k lines of code and has been deployed to tens of customers. So far most of the bugs have been concerning the user interface of the product.

According to the software life cycle model represented by Rajlich and Benneth (2000), the case product is slowly proceeding from the evolution phase to the servicing phase.

Improvements in the case company's development process are so far done by obeying the "evolution over revolution" approach: development evolution from the existing state of development, with small changes, implemented over time, with a specific end result in mind. The same approach is also used in the scope of this study when the defect tracking construct is presented. First step should not be too big to avoid employees get lost and frustrating themselves.

The role of the author of this research in the company has been rather active, as he has been working for it for four years, but he has not been involved in the development or design of the case product.

1.4. Structure of the Paper

In this section we set goals of this research.

Section 2 gives an overview on what benefits can be gained from a defect tracking system. Results are used to set objectives of the case company's defect tracking arrangement. The objectives are stated in Section 4.

In Section 3 we go through characteristics of Scrum development model on those part considered important in the defect tracking context. The whole Scrum process and its details are not described.

In Section 4 we describe the proposed defect tracking arrangement.

In Section 5 we evaluate the described arrangement. Based on the evaluation, we suggest the next steps to be taken to further develop defect management.

2. DEFECT TRACKING

This section explains motivation behind defect tracking and describes how defect data can be further utilized.

The term defect tracking (also known as bug management) refers to methods, processes and tools used to follow life-cycle of defects and gather and analyze information related to each defect. (Fredericks and Basili, 1998)

The term defect is a high level term including as well faults, errors (or bugs) and failures. (Fredericks and Basili, 1998) According to technical report by IEEE (1991), a failure occurs when a software does not behave as defined in its requirements specification. A fault is a mistake made by a software developer. The mistake itself is called an error.

In the following subsections, different motivation aspects of defect tracking are described.

2.1. Defect Management

The most important motivation for defect tracking is very logical and intuitive: it allows management of defects and tracking that they are eventually resolved. If defects are not resolved, it leads to customer dissatisfaction. (Black, 2004) (Kaner et al., 1999)

Each defect's life-cycle must be saved to the defect tracking system. Life-cycle is expressed with a state field indicating whether a certain defect has been handled or not.

In Black (2004) the following defect life-cycle is presented:

- Open, the defect has been registered to the tracking system, but it has not been handled yet
- In progress, the defect is currently under handling, i.e. it has been assigned to some developer or at least allocated to some forth-coming iteration
- Resolved, the defect has been solved
- Re-opened, the defect has been for some reason re-opened, usually due to improper resolution
- Closed, the defect does not need any actions, for example if the defect can not be fixed for some reason

Defect management data can be used to further organize testing activities. During the first testing iteration, all components of the system are tested with equal resources. When the iteration ends, defect data

is analyzed and more resources are assigned to test those modules from which most of the defects were found. This is due to fact stating that 80 percent of defects are caused by 20 percent of the system. (Patton, 2003) (Kaner et al., 1999)

2.2. Process improvement

In Grady (1996), defect tracking is considered to be the corner stone when improving software quality and quality management processes. Grady even states that "software defect data is the most important available management information source for software process improvement decisions".

Defect management is seen as a tool to make organizations to learn better process. From the process improvement view point, the most important defect data is the root cause of defect. Root causes can be used to focus training and process improvement. The organization eventually shifts from reactive responses to defects towards proactive responses. Failure analysis based on defect tracking data is used to evaluate defect patterns to learn product or process weaknesses. (Fredericks and Basili, 1998) (Grady, 1996)

Failure analysis sets its own requirements to defect data. The first step is to collect defect source along other defect data. Another important field is defect classification. An example classifying scheme is presented by Basili and Perricone (1987):

- Requirements incorrect or misinterpreted
- Functional specification incorrect or misinterpreted
- A Design error which spans several modules
- A Design error or an implementation error in a single module
- Misunderstanding of external environment
- Error in the use of programming language or compiler
- Clerical error
- Error due to previous miscorrection of an error

Defect classes can and should be adjusted along time as more insight concerning defects is gained.

The defect analysis is done in a separate meeting, during which a certain set of defects are analyzed and solutions to the found defect patterns are brainstormed. The meeting can be arranged for example in the end of the project before the next one is launched. Regularly arranged failure analysis meeting allows establishing of a continuous process improvement cycle. (Grady, 1996)

2.3. Quality measures

Defect tracking data allows measurement of software quality. Measurements can be used to monitor projects and to establish quality baselines. They can also be used to measure when a program is ready for release. (McConnell, 1997)

In Fenton and Pfleeger (1997) two defect-based quality measures are described.

The first one is *defect density*, which is defined as a ratio of number of known defects and product size. Of course this measure does not tell the whole truth, as it depends only on amount of known defects, but ignores understandably latent defects, that may be present in the system but of which the development team is unaware.

It should be kept in mind that defect density concerns more quality of the test process itself instead of quality of the tested product as it is totally based on amount of found defects.

The second measure is *spoilage*, which is defined as a ratio of time to fix post-release defects and total system development time. Spoilage is used especially by Japanese companies to define and measure quality.

The major problem involved in defect-based measures is that they require very carefully, consistent and thorough data collection, which usually conflicts with an agile development approach. Measuring is usually highly dependent on subjective ratings, which should be kept in mind when analyzing measurement results. To avoid this problem, measurements can be compared with data gathered from previous projects of same type implemented by the same team.

3. SCRUM

In this section, we represent Scrum development process. If not otherwise mentioned, this section is based on Schwaber and Beedle (2001) and Abrahamsson et al. (2002).

Scrum is an approach to manage the development process. It does not define any specific software development techniques for the implementation phase, but instead it concentrates on managerial and organizational aspects in a constantly changing environment. Scrum is based on a strict time paced model.

In many cases, Scrum is used together with some other agile approach, which then concentrates on engineering practices and techniques. One well known

combination is Scrum and extreme programming, known as XP@Scrum. (Vriens, 2003)

In the Scrum model, development is done in iterative and incremental cycles (also known as Sprints, see Section 3.2) lasting usually 30 days. Requirements are managed in backlogs. Usually there are different backlogs for the whole product and more detailed backlog for the current iteration. All items in the backlogs are prioritized and they are implemented in that order. These are described more detailly in the following section.

3.1. Product backlog

The product backlog contains all that anyone involved in the product or development process has thought is needed or would be a good idea in the product. I.e. the backlog contains all features, functions, technologies, and bug fixes that constitute the changes to be made to the product in future releases.

Format of the product backlog and formality of log items depends completely on the organization. In general, the product log contains items with very varying detail level. The only requirement for each product log entry is that it is detailed enough to drive the highest rhythm frame called Sprint.

Contents of the product backlog is highly dynamic, as the management of the company can repeatedly change it according to identified needs or changes in the business and customer environment. The only required fields for each log entry is a description and priority.

In addition to product features, the product backlog can also contain issues. Issues are mainly quality attribute (security, performance, modifiability etc) concerns that are not yet realized to some concrete work item. Also issues are associated with a priority. Issues are in some phase turned into concrete work items.

The product backlog is emerged from multiple sources. The most important source is management, sales and marketing persons, who have a vision concerning the product and its designated customer. Software architects and engineers are responsible for log items concerning the technology that glues the whole product together.

Customer support is responsible for feeding the product backlog with product flaws and all other

bugs found by customers. These can be either concrete features or improvements, or just issues that will finally evolve to concrete work tasks.

However, only one person is responsible for managing and controlling the product backlog. This person is usually in the role of product or project manager. This guarantees that there will be only one backlog for each product and thus development teams do not have to suffer from conflicting lists or conflicting list items as conflicts are resolved before items are added to the backlog.

3.2. Sprints

A Sprint is a 30 day long period (iteration) during which a set of items from the product log are implemented. In the beginning of a Sprint, its contents are planned. In this Sprint planning meeting, it is decided what functionality is build during the next Sprint. One of the inputs to this meeting is the Product backlog, from which log items are selected to be implemented in the priority order.

The selection of implemented features is done on basis of two things: priorities set by the product backlog owner and work estimates given by the development team. Total work estimate can not exceed time of one Sprint but still the development work should serve for implementing the most important requirements first.

After the next set of functionalities is selected, the team goes through them and on basis of that, compiles a list of tasks it has to complete to achieve the Sprint goal. This list is called the Sprint backlog. The Sprint backlog contains more detailed items than the Product backlog.

During each Sprint, a team is required to test what it builds. According to these tests, the Sprint backlog is modified: if test results bring up new tasks, these are added to the Sprint backlog. As well, if tests of a certain implementation passes, the corresponding task is marked as done. Only members of the team are allowed to make changes to the Sprint backlog.

If the team discovers that it can not complete all the selected work during the current Sprint, the goal is downgraded on basis of a discussion between the product backlog owner and the team. Also if the testing executed during the Sprint addresses quality issues that can not be resolved during the Sprint, these issues are added onto the Product backlog and prioritized by the Product backlog owner.

3.3. Scrum meetings

Sprint progress is tracked by arranging regular meetings, called Scrum meetings. Scrum meetings are kept daily and the meeting should not last more than 15 minutes. Intention of the meeting is to check what the team has achieved since the last meeting.

Agenda of each Scrum meeting consists of going through the following questions with each member of the team:

- What have you done since the last Scrum meeting?
- What will you do before the next Scrum meeting?
- Are there any impediments?

Feedback gathered during the Scrum meeting can be used to adjust sprint backlog and add items to the Product backlog. Reported impediments should not be added to the product backlog if they are not directly related to some quality issue.

3.4. Scrum and defects

In Scrum, defects are managed in the same manner as other requirements or work tasks are. They are part of backlogs as issues or as elaborated tasks. Defects are first class citizens and they are not managed independently from other tasks, as in Scrum there is no dedicated testing team, but the implementation team is responsible for reaching planned quality goals.

If during a Sprint defects are found but they can not be fixed in the scope of it, they are entered onto the Product backlog and prioritized by the Product backlog owner.

Customer reported defects are entered onto the Product backlog by the Product backlog owner on basis of the feedback he/she has received from customer support persons.

4. DEFECT TRACKING ARRANGEMENT

In this section, the case arrangement and its objectives are described. Objectives are set together with the product manager of the case product.

The elicited objectives are as follows:

- 1) Manage defects and guarantee that all of them are somehow handled
- 2) Provide a communication method inside the development team

- 3) Provide visibility how the project is progressing
- 4) Gather input for the software process improvement team on basis of the defect classification data

As Pol et al. (2002) states, the set-up of defect management depends on the type of organization and its test organization. In the case company's context, there is no distinct test organization as all code is tested – as Scrum suggests – by developers themselves. Each team is responsible for the quality of their own work.

This differs the situation from traditional organization models, where there usually exists a distinct testing teams doing their own job separately from rest of the product development team. In this model, defect tracking system is more a mean of communication between these two teams. (Pol et al., 2002)

In an agile environment, the role of defect tracking system is more a mean of internal communication of the development team and to steer direction of the project. Defect tracking system is a centralized point containing a complete list of all tasks to be done. These tasks include defects as well as feature implementation tasks. All tasks are prioritized and during a scrum meeting it is decided whether a defect fixing tasks or feature implementation tasks will be done.

The combined defect tracking and requirements management system will be the main tool to steer direction of the project, as same resources are used to fix defects and to implement new features.

4.1. Tool selection

A good defect tracking tool should enable the gathering and management of the defect data. However, the real value of a good bug tracking tool lies in its ability to gather and manage information during each bug report's life-cycle. (Black, 2004)

Support for the workflows in the defect tracking system should be integrated with release management. This allows traceability between defects, tasks done to resolve them and the release in which they are fixed. In Scrum, release management is done through product backlogs. In the product backlog, log items are road-mapped by allocating each item to a certain release. This allocation forms a rough road-map. (Black, 2004)

As Scrum recommends, defects are handled in backlogs along with other product related features

and tasks. At the moment, the case company uses Microsoft Excel based backlogs. This, however, is not the optimal situation, as most of the developers use only Linux workstation and they are thus unable to access the backlog file.

Currently the case company has a Lotus Notes based defect tracking system, but as it was difficult to use and was not integrated with any other project management tool, its use slowly decreased and finally it was dropped completely. Also the need for defect tracking tool was not clear enough, as communication between sales team and development teams moved very fast and information and defect resolution requests were shared verbally.

As sales and marketing persons use Windows instead of Linux, the selected defect tracking tool should be platform independent. This allows the whole company to use the defect data. Sales and marketing can browse the data to see whether important (from their point of view) defects has been already fixed or at least allocated to a some forthcoming release.

The tool selection was based on evaluation of defect tracking tools available in the Internet and by discussing with other software companies applying similar development process. Own tool implementation was not a choice, as it reserves resources from actual business projects. The evaluation was done mostly on basis of demonstrations and feature matrixes.

On basis of the lightweight evaluation, the choice was Jira. Jira is a commercial defect tracking tool which is implemented as a web-application. Thus it can be used through a normal browser allowing both Linux and Windows users to browse the same information. It also allows allocation of issues and defects to a certain release. Jira can also be used to manage product backlogs.

4.2. Defect Data

The next step was to define the structure of gathered defect data, which is highly dependent on the selected tracking tool. The objectives also set requirements to defect data to be gathered. For example, without decent classification data the process improvement objective could not be achieved. The defect management objective requires support for defect life-cycle (state) related fields.

To fulfill the above-mentioned objectives, the following data will be gathered for each defect:

- Unique identifier (generated automatically by the tool)
- Product / project name
- Defect description
- Priority
- Type / classification
- State
- Name of the author of this entry
- Product version

In the 1st phase, classification will be done in the following scale:

- Requirements incorrect or misinterpreted
- A Design error
- An Implementation error
- Misunderstanding of external environment
- Error due to previous miscorrection of an error

The above-mentioned classes were identified together with the product manager and with the process improvement responsible of the case company. Classes are supposed to be distinct from each others. In the beginning there should not be too many classes to keep the defect data simple. In the future, classes can be elaborated on basis of the defect analysis done in the end of each project.

Simplicity was also the key rationale while deciding the defect data structure. The structure can be elaborated and extended in the future, when more experiences concerning defect tracking are gained.

4.3. Defect management guidelines

One of the most important step to be taken before defect management is deployed into use, is to agree how it is used and to design how it will be seamlessly integrated with the software development process.

These rules should be documented as a part of the process guide. Even more important is to take these rules as a part of the daily process coaching.

Developers can directly report defects to the defect tracking system in the following cases:

- A new defect is found during another task implementation.
- Known defects were left to the product as they could not have been fixed before deadline.
- A major issue with a particular quality attribute is found. For example, the product is not scalable enough or its usability does not meet requirements. Fixing of such an issue needs major design (or re-design and re-factoring)

and thus it should be taken account at the product backlog level.

In each case the project manager is responsible for assigning priority for each defect. Developers themselves are not allowed to set priorities as they do not have the decisions responsibility.

Reported defects are handled at the sprint level, i.e. defects are kept in the sprint backlog. They are not inserted into the product backlog, as its purpose is to guide the direction of the whole project, while the sprint backlog guides direction of one sprint. If all defects could not be resolved during one iteration, these over left defects can be transferred into the product backlog. Also major quality issues should be inserted directly into the product backlog.

In the project review meeting organized in the end of each project, the defect list gathered during the project will be analyzed and quality process improvement actions will be targeted to tackle the found weaknesses and to further elaborate defect classification scheme.

As the Scrum model states, only the product backlog owner is allowed to change the product backlog. Thus all defect reports should be first reported to him/her and are then entered onto the product backlog. Also defect state changes are done by the backlog owner. Changes should be done after the daily scrum meeting to keep the defect states up-to-date.

The selected defect tracking tool also supports sending e-mail notifications when a new defect is entered into the tracking system. This was considered as a very valuable feature among software developers, as they immediately receive notification about new defects concerning products in which they are involved.

Even though developers receive e-mail notifications, they should not start to resolve defects before they have agreed about it with the product manager (who also is the owner of the product backlog). This is a consequence from responsibilities stated in Scrum model: it is on product managers responsibility to decide scope of each Sprint.

If a defect's severity requires immediate resolving actions, it can be taken into account when future actions are planned in a daily Scrum meeting. This should be also taken into account when Sprint contents and schedule is planned. In a Sprint planning meeting, a certain amount of the Sprint's total work time is reserved for defect fire-fighting. This amount

can and should vary according to the product's maturity, type and many other factors.

The fire-fighting time should be adjusted according to experiences gained from previous projects. Most probably the fire-fighting estimate will fail in the few first Sprints.

If no severe defects occur during a Sprint, the over left time can be used to develop more features to the product and thus exceed the Sprint goal.

5. EVALUATION

This section describes the evaluation method used and represents evaluation results. The evaluation was done by interviewing persons involved in the product development. One of them is product manager, two of them are members of the development team and one is a sales and marketing person. Totally four interviews were done.

The aim of the evaluation was to check whether the objectives set for the defect management arrangement are met or not. The arrangement was not evaluated against the process improvement objective due to firm schedule of this research. The evaluation was done during two weeks, which is definitely too short time to make any process improvement action – not to mention evaluating the improvements' effects.

The gathered data was analyzed manually by clustering findings under the following categories:

- 1) Project visibility
- 2) Sprint log management
- 3) Defect data sufficiency
- 4) Usability

5.1. Results

The major benefit gained from the defect tracking arrangement seems to be the visibility it provides to the whole development process. Separated defect tracking and requirements management systems could alone provide only partial visibility to the project while hiding rest of the tasks.

Combining all tasks under one system provides a complete status of all project tasks and allows project manager (scrum master, product manager) to make decisions between tasks: should we next implement that feature of fix that defect?

The major problem with the deployed system were the views it provides to the defect and task data. For example, sales person experienced the

provided data too complicated and thus it was difficult to find the information in which he was interested. A method to display only certain fields while hiding rest of them was suggested. In general, usability of the system was experienced to be good enough and thus fulfilling its requirements.

Sales persons find it easy to track status of a certain issue from the system. Product manager was not willing to allocate issues to releases before he was really sure it can be implemented during a certain iteration as he was afraid that sales persons would sell the feature beforehand.

The gathered defect data structure seemed to be sufficient for developers to fix reported defects. More guidance concerning defect description was proposed. This guidance should be part of the process documentation, but it should also be studied and understood by persons communicating with customers and registering defects reported by them to the defect tracking systems.

Most of the defect reports were not unambiguous enough to be used as alone to find the reported defect from the product. In most of the cases, the developer responsible for the fix had to contact the defect reporter and get more detailed reports.

By tutoring persons in the customer contact, they could also be able to ask correct questions and thus elaborate the defect report to be unambiguous and clear enough.

Both of the developers experienced the classification of defects rather complicated. The root-cause of a defect is not easy to discover in some cases. For example, after first look, cause for a defect could be a design error, but after careful elaboration, the root cause seemed to be a problem with initial requirements.

Classification choices should be more clearly linked to different phases of the development process to make the defect tracking provide more exact input to the process improvement team.

On basis of the analyzed interview data, the following issues were found:

- A change request will be send to author of the Jira tool to make them implement a feature allowing hiding of certain fields.
- Defect reporting guidance concerning defect description should be elaborated.
- Clarify defect classification

These issues will be taken into account when the next version of the defect tracking arrangement will

be designed.

6. DISCUSSION

This study presented motivation for the use of defect tracking. According to a literature study, the defect tracking was analyzed in the context of Scrum. Scrum is an agile development and management process used in the case company of this study.

As most of the Scrum principles are managerial and organizational aspects, it does not state anything about defect tracking. However, it suggests that both defects and other project tasks are kept in the same system.

On basis of the Scrum study, a first version of a defect tracking arrangement was described. The arrangement was done for a small case company developing software products. This arrangement was evaluated by interviewing different stakeholders of the system inside the case company.

Results of the evaluation can not be generalized, as the evaluation was done only on basis of one case company. However, general defect tracking guidelines can be applied also by other organizations using any Scrum-based agile development process. The set of guidelines was created on basis of the literature and thus they do not contain any case company specific data.

As a result of evaluation, the current set of reporting guidelines should be elaborated. Especially the root cause concept was found to be difficult. Also the current reporting tool does not support enough different views for different stakeholders. Currently all stakeholders have to browse defect data through the same detailed view.

REFERENCES

- Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods, review and analysis. Technical report, VTT, 2002.
- Victor Basili and Barry Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27:42–52, Jan 1987.
- Rex Black. *Critical testing processes – plan, prepare, perform, perfect*. Addison Wesley, Boston, USA, 2004.
- Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics – A Rigorous and Practical Approach*. PWS Publishing Company, 1997.
- Michael Fredericks and Victor Basili. Using defect tracking and analysis to improve software quality. Technical report, A DACS State-of-the-Art Report, Nov 1998.
- Robert B. Grady. Software failure analysis for high-return process improvement decisions. *Hewlett-Packard Journal*, Aug 1996.
- IEEE. IEEE standard glossary of software engineering terminology. Technical report, 1991.
- C. Kaner, J. Falk, and H. Q. Nguyen. *Testing Computer Software*. John Wiley and Sons, New York, USA, 1999.
- Steve McConnell. Gauging software readiness with defect tracking. *IEEE Software*, May/June 1997.
- R Patton. *Software Testing*. Sams Publishing, Indianapolis, USA, 2003.
- Martin Pol, Ruud Teunissen, and Erik van Veenendaal. *Software Testing – A Guide to the TMap Approach*. Addison Wesley, 2002.
- Vaclac T. Rajlich and Keith H. Benneth. A staged model for the software life cycle. *Computer*, 33 (10):66–71, 2000.
- Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, Upper Saddle River, NJ, USA, 2001.
- Christ Vriens. Certifying for cmm level 2 and iso9001 with xp@scrum. *Proceedings of the Agile Development Conference*, pages 120–124, 2003.