International Journal of Computer Applications (0975 – 8887)
Volume 115 – No. 14, April 2015

REVEDERE – Distributed Support System for Code Review Process

Andrei Nuc
Tora Trading Services
Cluj-Napoca, Cluj County, Romania

Cosmina Ivan
Department of Computer Science
Technical University of Cluj-Napoca
Cluj County, Romania

ABSTRACT

Code review process represents a common practice in software industry for checking product quality and reducing the number of defects deployed to consumers. To integrate well with the current accelerated trend in software development, there is a need for automation and simplicity, through tools, integrated in programmer's workflows. In this paper we present the primary motivations for software inspections and we introduce a support platform for managing this activity. The system is based on core code review concepts and is presented as a middleware for human interaction. Also, a communication framework was implemented, to ease information flow and management in the code review middleware.

General Terms
Middleware, distributed systems, review

Keywords
Code review, software engineering, plug-in

1. INTRODUCTION

The code review process is a fundamental stage in the software development, used to check, increase and maintain the software quality. It happens after the code writing stage and before testing, and it is usually performed by a different person. Because this operation does not have an economic value and it is not regulated in a formal way (unlike best practices for writing code and testing), it is missing from the production chain or it is executed in aninformal way.

That is why it is necessary to build a platform to simplify this process through efficient actions, reduction of distractions for the users, and to integrate this flow in a friendly environment, easy to learn and to use.

A plug-in for the main IDEs used for Java programming is a product marketed as a solution. This plug-in can help programmers collaborate, make code review requests, inspect code and give feedback.

Some psychological aspects can be used to optimize the code review flows. Studies from [1] have shown that reviews are more efficient when the reviewer is not disturbed, the time and quantity allocated for work is well dosed and there is a direct communication link between programmer and inspector. Also, integrated tools can increase the productivity, reduce the context switch overhead and offered reachable targets.

2. MOTIVATION

During software development, one of the driving force is a solid and performant product, that works as expected in normal and critical phases. To minimize the phases when the software does not behave as expected few techniques can be used during design, implementation, testing and maintenance. K. Burke made a study related to bug finding and gathered a statistical view for some techniques, along with the found defect ratio. His results are presented in figure 1.

McConnell noticed that no technique detects more than 75% of defects [2], so no singular choice is adequate. A cheap combination of several choices can lead to a better percentage of detection. Early spots are easier and cheaper to fix. Some studies claimed that problems that reach production are 100 times more expensive to fix than before that phase. [3]

According to [4], inspections are a better approach to find bugs than testing. Unlike functional and structural testing that can detect defects but cannot localize them, at the end of the code inspection, the problems are pinpointed. If tests deal with the effects, the reviewer tries to find generic and specific causes that threaten performance, correctness and extensibility.

Main motivations according to Alberto Bacchelli and Christian Bird [5] for code review are:

- **Defect discovery** -correct logic is applied and design errors are detected
- **Code improvement** - the code is improved if a colleague suggest better algorithms or structure
- **Alternative solutions** - the reviewer can come with a better idea, but at the cost of possible disagreement
- **Knowledge transfer** -"*one of the things that should be happening with code reviews over time is a distribution of knowledge. If you do a code review and did not learn anything about the area and you still do not know anything about the area, then that was not as good code review as it could have been*" as it was in suggested in[5]
- **Team awareness and transparency** - The changes are acknowledged by the whole team and it is their duty to check that the implemented feature was the same as the one that was requested.
- **Share code ownership** - During software development a risk is born, known as "*bus factor*". This is correlated with the knowledge concentration in team members. The bus factor represents the number of developers required to be unavailable (e.g. bus accident) that will bring the project to a halt. "*In reality, many projects rely on one or more 'heroes' who are the only ones who understand certain parts of the system. When these heroes leave (and you should assume that they will), you must be prepared to recover*" [6]. Through an efficient review process, the knowledge related to a particular module or component becomes common to more developers.

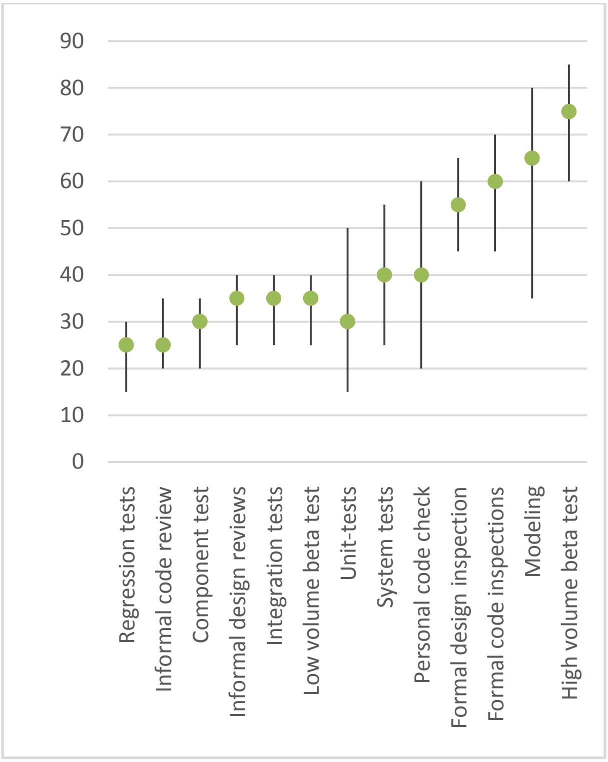

Figure 1: The number of problems founds, grouped by used techniques. Adapted from [9]

3. BACKGROUND

The software inspection process in 80' was based on a different ideology than the techniques used today. M.E. Fagan described in IBM System Journal [7], a methodology used for detecting problems, the inspection: "*Inspections are a formal, efficient and economical method of finding errors in design and code*." There is an inspection team with members playing different roles:

- **The moderator**: *"the key person in a successful inspection. He must be a competent programmer but need not be a technical expert on the program being inspected."*[7] Named as the leader of the team, his main responsibilities will be: the place and time planning for meetings, inspection result reports, and other member's motivation.
- **The designer**: *"the programmer responsible for producing the program design."*[7]
- **The coder | implementer**: *"the programmer responsible for translating the design into code."*[7]
- **The tester**: *"the programmer responsible for writing and/or executing test cases or otherwise testing the product of the designer and coder."*[7]

The diversity of the team is important in M.E. Fagan perspective. Each person from team should deal with only one aspect of the software development. The maximum recommended time for inspection is two hours, otherwise the detection efficiency shrink considerably. Also, *"the time to do inspections and resulting rework must be scheduled and managed with the same attention as other important project activities… If this is not done, the immediate work pressure has a tendency to push the inspection and / or rework into the background, postponing them or avoiding them altogether."* [7]

The process follows two directions: the design and the implementation. Both are monitored proactive so they correspond to initial requirements.

At the end of the inspection process, a resume is made, containing different kinds of detected errors, their number and implied risk. After that, all errors and problems noted in the inspection report are resolved by the designer or coder / implementer.

It is necessary to monitor the follow-up: and in [7] was said that *"It is imperative that every issue, concern, and error be entirely resolved at this level, or errors that result can be 10 to 100 times more expensive to fix if found later in the process."*

Using this scheme, the programmer can realize the frequent mistakes he makes, how often this happens, how he can detect and prevent them in the future.

But, according to [5], *"over the years, researchers provided evidence on code inspections benefits, especially in terms of defect finding, but the cumbersome, time-consuming and synchronous nature of this approach hinders its universal adoption in practice"*.

At this moment, the code review methodologies are simplified, as a response to the challenges of the 80's ideas. More tools are developed and used and the decision factors in companies consider using them based on empirical evidences to motivate this kind of process in software development.

In a world where the code must pass from idea to users devices quickly, *"as a programmer, you have little tolerance for anything that impedes your productivity"* [8], so many will consider that the code review process slows down the creative process and together with the meetings, the reports represent a waste of time. Those must be convinced to use lightweight and easy to use tools for software inspection.

According to a SmartBear document [8], the code reviews can be classified in five classes:

- **Formal review (inspection).** More people review the code, the better.
- **Over-the-shoulder review**. A formal and widespread technique that consists in one programmer presenting to another colleague the changes and the new code. When a problem is found, it can be fixed on the spot, under reviewer watch. Remote over-the-shoulder review can be accomplished using modern instruments. The main benefits are the direct communication and simplicity (more ideas can be transmitted orally rather than in a predefined, formal document. The major flaw is the absence of proof. Nobody knows that the process took place and its results. Also the reviewer depends on the presenter.
- **Sending e-mails**. Code review requests are sent via e-mail to colleagues delegated with this responsibility. The activity is scalable, because it can be used in the same room, or over an ocean. There is a decoupling between the person who write the code and the person who review it. According to [8], a 15 minutes period is required for a developer to reach the productivity "*zone*", the main disadvantage is following the changes and the conversations. A consensus cannot be achieved if more regions of code or more developers are involved.

- **Tool assisted reviews**. Specialized tools are used to realize the code review process. They automatically gather files, display differences, comments, defects, collect metrics and are integrated in developer workflow.
- **Pair programming**. This is often associated with eXtreme Programming (XP) and agile development in general. It represents the continuous code review. Two developers sit at the same workstation but only one writes code at a moment in time. The main advantage is the live feedback and multiple concentration. The main disadvantage is the fact that the reviewer can't disconnect from the context and have an objective opinion. To counter this, another pair of eyes can make a standard review.

Every class of review processes have their advantages and disadvantages. The companies prefer to settle on an approach that is the most suitable for their workflows and it is hard to change the tracks after the process is well-known. Ideas from each approach are used to design the functional requirements of the proposed system.

4. RELATED WORKS

In code review industry the main players are the ones involved with the development process as a whole. Big names like Microsoft Corporation, but also niche players like Atlassian Software Systems and SmartBear Software, try to gain the hearts and money of the users.

Two commercial products are analyzed to understand the current state of the industry. The systems from Microsoft Corporation (Visual Studio) and Atlassian Software Systems (Crucible) are used to present the opposite directions that divided the domain in two sides.

4.1 Microsoft Visual Studio

The Microsoft Corporation offers code review process integrated support in Visual Studio Suite. The visible part is hosted in IDE and the back-end functionality is managed by Team Foundation Server. The code review module is proprietary and cannot be modified. Integration with Visual Studio restrict the environments that can support this plug-in.

Administration is done inside Team Foundation Server. Asynchronous code review is supported. The involved programmers are not required to have their IDE open at the same time. This offer flexibility in scheduling priorities and allows members to work on different projects.

4.2 Atlassian Crucible

Unlike the previous system, Atlassian chose an approach integrated in company DNA: web services that can be accessed anywhere. The code is proprietary but portions can be distributed to companies that sign an agreement, to adapt it to their needs.

The licensing costs depend on the users' numbers and the type of organization. Integration is essential to capture clients, so Crucible can be used with Atlassian JIRA and Atlassian FishEye. Asynchronous code review is supported as long as access to server is available.

The current trend in review industry is to offer integrated solutions to clients, based on well-established environments (Visual Studio and Atlassian JIRA). New workflows are better integrated in familiar tools.

5. GENERAL ARCHITECTURE

5.1 Foreword

The proposed system (Revedere) takes the best parts of the both sides presented before. Integration with the IDE, available source code and remote management are used to ease the life of programmers, reviewers and administrators. The application can be modified to better suit internal requirements.

Asynchronous code review is supported inside the Eclipse IDE. The content is transferred automatically between users. Meta-information and comments can be attached to files and can be visualized in appropriate context.

A few architectures and technologies were tested and the most appropriate path was chosen. The fact that Eclipse is written in Java and there are a lot of enterprise-ready framework makes clear the language choice.

Revedere has the client-server architecture. The users interact with the system via a plug-in installed in Eclipse, to request code review, to inspect the software and to communicate with other members of the team.

The back-end is built on Distry (own platform for creating monitored, distributed systems).

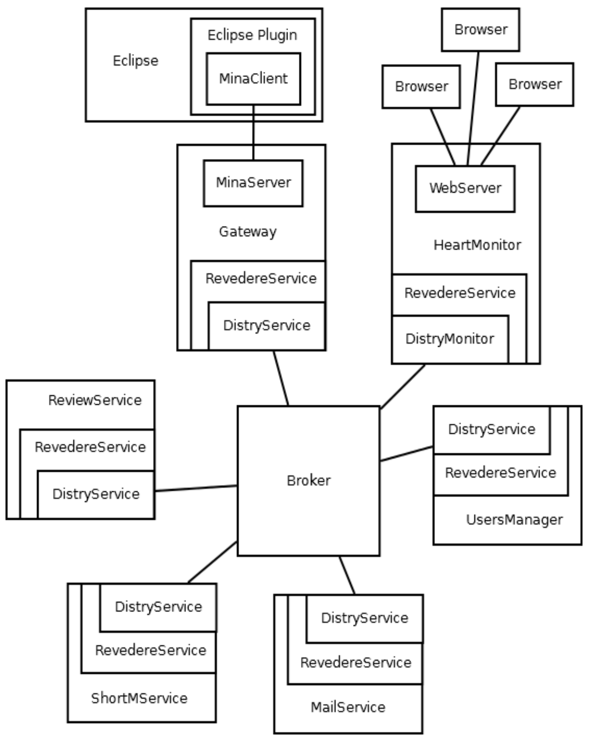

Figure 2: The Revedere system architecture

The client uses Eclipse APIs to integrate and interact with the user.

5.2 Distry

Distry platform is a middleware, which can be used on top of any distributed API, and offers transparency regarding the communication channel, monitoring and command services for systems that use its interface.

The main abstractions offered to upper layer applications are: subscription channel, send channels, remote actions, and information about participant entities.

The current design is based on JMS publish – subscribe model. The Apache ActiveMQ implementation was chosen because it is lightweight and can be wrapped in pure Java container. This allows unified and simplified deployment of services.

Two types of services are offered to consumers:

- **DistryService**: a basic API that can be used to configure connections, publish service health and accepted commands plus I/O abstraction.
- **DistryMonitor**: a specialized DistryService that monitors other DistryServices, collect health information and allow sending commands.

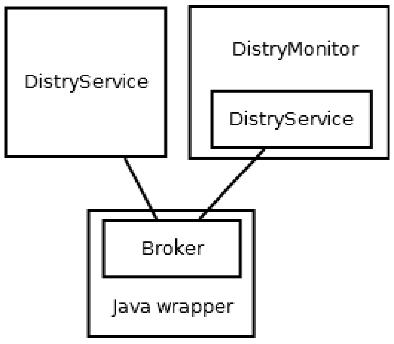

Figure 3: Distry architecture

5.3 Revedere Back-end

Every back-end service is based on DistryService, so communication and management are simplified.

- **Gateway**: the gate component. It receives the messages from outside and forward those to the qualified processors. The management and control of the users is done at this level. The component filter requests and reject illegitimate connections
- **ReviewService**: the component responsible with the management of code review operations. Every review can have one of the following states: Request, Accept, Deny, Done, Closed. A review document is attached to each request. This contains data required for understanding the context and decisions. Each team can have a customized document.

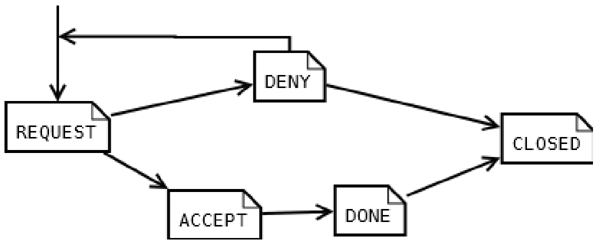

Figure 4: Review possible states and transitions

- **ShortMessageService** intermediates the team communication, or, mediate the discussion between requester and reviewer. There is an option for persisting messages so the companion can read received messages even if he was offline when they were sent.
- **MailService** offers support for sending e-mails.
- **UsersManager** deals with authentication and register. The authentication request can come from Gateway (for plug-in users) or from HeartMonitor (for web users).
- **HeartMonitor** monitors the rest of the services using DistryMonitor API. It also support a web interface where users can see the state of the system and execute maintenance commands. There are a few service statuses attached by the monitor based on aliveness: UNKNOW, OK, LATE, LOST.

5.4 Revedere Plug-in

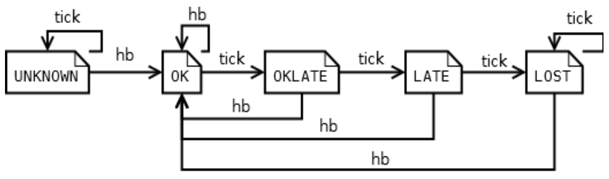

Figure 5: Service status transition

Client communicates with server via sockets, using Apache Mina library. On client side a visual wrapper is set over an API to better integrate with the IDE. Settings are available on Eclipse Preferences zone, and the plug-in is presented to the user via a view that can be hidden and restored manually.

5.5 Technology perspective

A few open source libraries are used to provide core functionality in the distributed system.

Apache ActiveMQ represents Apache implementation of JMS standard. The main benefit of using this library is its small size and the native / cross-platform approach. ActiveMQ fuels the Distry middleware.

Apache Mina (Multipurpose Infrastructure Applications) is a network framework, implemented in Java. Different transports can be used: TCP, UDP or serial communication. Mina connects the clients to the back-end gateway.

Apache Commons Email is an API built on Java Mail API, which simplifies sending mail. MailService use this functionality to distribute information outside Revedere environment.

Log4j is a logging framework used to track, persist and help debug software behavior. Different types of logging can be assigned to filter information based on relevance: ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL.

JDOM2 manipulates XML (eXtensible Markup Language) content. It offers a Java centric view of the data and operations. JDOM2 is used as a persistence API in Revedere project. The content is stored in XML format, so it can be easily parsed by other applications or read by human being.

Embedded Oracle GlassFish is a JavaEE server developed by Oracle Corporation. It is reference implementation of JavaEE. The embedded characteristic allow the encapsulation in a RevedereService. This is useful because the Web Portal is managed from a RevedereService.

Eclipse RCP (Eclipse Rich Client Platform) is a minimal set of functional plug-ins that, together with a kernel, that can be used to build a working IDE. The following components are contained in RCP: Eclipse Runtime, SWT, JFace and Workbench.

Beside integrated libraries, a few more technologies and services were used to develop the software

JUnit is a unit-test framework for Java language. It is one of the most used third-party libraries on GitHub projects. It simplifies testing classes, behavior and integration. The main benefit of tests written for JUnit framework is the readability. JUnit is used together with Mockito, a mocking library.

Jenkins is an open-source tool for continuous integration. Unit-tests are run automatically, after each integration to validate changes. This way, there is a proof that the software will behave as expected in the tested scenarios and can be released to users. The main advantages of running continuous integration software are: fast detection of defects caused by changes, avoid releasing unusable software, feedback for developers and code coverage.

Sonar is a web-based open-source platform for continuous software quality inspection. It has tools used to track some quality metrics during development: lines of code, complexity, code coverage, dependencies and to detect possible problems.

Mockito enhanced unit-tests are run by JUnit framework automatically by Jenkins continuous integration framework. The results are merged with custom metrics applied to source code and presented to developers for analysis.

6. INTEGRATION IN USERS ENVIRONMENT

On server-side, each service is a Java process than can be placed on a different machine. The only requirement is access to broker machine. Web functionality is integrated in HeartMonitor, using Oracle Embedded Glassfish.

Chat, review requests, and reviews are triggered using few commands in the right context. The user that requests the code review, chooses the desired project, select all the files to be inspected, and assign a person to take care of the review.

The reviewer is notified when a new message or review request has arrived and can accept or deny it. When the acceptance is given, the project is downloaded locally and opened in workspace. Using context menu, the inspector can mark code and add comments to specific parts of the text. When the review is finished, the comments arrive on the originator side and they can be fixed.

7. TESTING

Code was tested using unit-test and manually, running the application in expected scenarios. Different tools (Jenkins - continuous integration, Sonar – real-time feedback) were used to keep track of the quality of the source code, and manual tests were made to check successful integration.

At this stage prototype evaluation was enforced. A few predefined scenarios were considered and checked after each development iteration:

- Connection with back-end
- User authentication
- Registration
- Chat between members
- Code review request / response
- Browser authentication
- Service management in browser

Found issues are managed using an issue tracker hosted on GitHub.

8. FURTHER DEVELOPMENT

The study and application started as an idea inside a team of developers that use e-mails and Mercurial repositories to handle the code review process. Even if the plug-in is in incipient state, it shows a great potential and future features are expected.

A more transparent system is required in most software companies. Most projects are self-hosted and managed, so the direction is to bring the functionality of the server beside the client, but still offer a buffer zone for extensibility. This will allow ad-hoc reviews and a more natural deploy scheme.

The Eclipse plug-in can be rewritten to better accommodate with the final flows and improved to reduce traffic.

Intellij IDEA users expressed their desire to have a plug-in that can be used in cross-IDE teams to manage code review

For small teams an integrated version is recommended. The requirements are smaller and the ad-hoc approach is suitable for these environments. A new project was born: RevedereLite. It will try to integrate the server functionality in the plug-in code. The users will connect to each other and the excessive middleware is removed.

A lot of companies use various DVCS (distributed version control systems) to keep track of changes. Integrating this flow in Revedere could ease the transitions of these teams because the quantity of source code to be transmitted is reduced.

Multi-reviewers methodology can be applied to further improve the quality of the code. The testers can see the problems detected by the reviewers and stress test those issues.

To reduce the need of separate chat rooms, the users could communicate using an enhanced version of the plug-in. A forum window can be implemented and back-end will deal with the extra logic required for forwarding and storing messages.

According to [1] a few practices should be enforced to gain a more effective, efficient peer code review. This simple rules can be expressed as conditions imposed or checked by the software.

- Checklists or TODOs for developers, reviewers and testers
- No more than 300-500 lines of code per hour.
- Review session no longer than 90 minutes
- Self-check before handing to review.
- Review changes enforcement
- Keep comments shorter than 25 words
- Avoid interruptions

Because the Distry framework and Revedere platform are separated so the former can be used and developed independently.

The main development areas for Distry are distributed management of file system, check pointing, recovery and redundancy.

9. CONCLUSIONS

The code review field is a vast domain, and represents an important step for developing quality software.

There are a lot of ways of handling and making reviews, optimized for different scenarios. Automated tools can improve productivity and reduce downsides. Proprietary and open-source solutions already exists, but they tend to stay rigid, so companies must adapt.

With the current paper and implementation, we realized a solid backbone that can be used to build and customize a software fit for every company that use Eclipse IDEs to develop software.

Open-source libraries were used as the building block of the application, and the result is published in a permissive license.

There are a lot of further development ideas and feedback from enterprise programmers. These suggestions could represent starting points for new features or improvements.

The main focus in the future is to offer a more compelling environment for review process, to study the interaction between users and the system and implement a solution for different IDEs.

10. REFERENCES

[1] Cohen J., 2011,"11 proven practices for more effective, efficient peer code review," 25 January 2011. [Online]. Available: http://www.ibm.com/developerworks/rational/library/11-proven-practices-for-peer-review.

[2] McConnell S.2004, “Code Complete: A practical Handbook of Software construction”, Second Edition, Redmond, Washington: Microsoft Press, 2004.

[3] Shull F. e. a., "What We Have Learned about Fighting Defects".

[4] Victor R.W.,R. Basili R.1987,"Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions of Software Engineering, Vols. SF-13, no. 12, pp. 1279-1295, 1987.

[5] Bacchelli A. 2013,"Expectations, Outcomes, and Challenges of Modern Code Review," Microsoft Research, 2013.

[6] Bowler M. 2005, "Agile Advice," [Online]. Available: http://www.agileadvice.com/2005/05/15/agilemanagement/truck-factor/.

[7] Fagan M. 1976, "Design and code inspections to reduce errors in program development," IBM Systems Journal, pp. 182-211, 1976.

[8] Brown E. Jason C. and Steven T. 2013, "Best Kept Secrets of Peer Code Review," 2013. [Online]. Available: http://www.lexingtonsoft.com/assets/white/documents/best-kept-secrets-of-peer-code-review.pdf.

[9] Burke K.2011,"Why code review beats testing: evidence from decades of programming research," 3 October 2011. [Online]. Available: https://kev.inburke.com/kevin/the-best-ways-to-find-bugs-in-your-code/