Code Reviews, An improved, light weight version over Fagan Inspections

Introduction:

Code review is an important step in software development. It is well known that it is cost effective to fix bugs, defects in software early on in the product development cycle. In addition, writing code to enable maintainability is important. Generally, more time is spent in reading, maintaining a code base than in writing the original code base. Fagan inspection is a common, popular method for code inspection. A variety of different flavors of this methodology is used in the industry. Fagan inspections, while efficient if fully implemented, has some process overhead that makes its adoption in practice difficult. In this paper, a lighter process is discussed that helps reduce the process burden with Fagan inspections without losing its positive points.

Fagan Inspection:

As discussed in [1],[2],[3] Fagan inspection involves the following processes.

a. Planning – This involves finding the inspectors, determining the work to be inspected, finding, arranging a common meeting place for the author, inspectors and the moderators.
b. Overview – This is where the author explains the work to the inspectors.
c. Preparation – This involves the author providing printed copies of the review material to the inspectors and the time that is needed for the inspectors to go over the material to be ready for the inspection meeting. This time is not used to find defects.
d. Inspection Meeting – In this meeting, the author typically goes through the source code line by line. This is a meeting where the defects are found, noted.
e. Process Improvement – After the meeting, the areas where the process should be improved in subsequent meetings is noted.
f. Rework – All the defects noted in the inspection meeting are now fixed.
g. Follow-up – This where the verification is made that fixes actually work.

In addition to the above mentioned processes, Fagan inspection implementation needs the author, inspectors to take on the following roles:

a. Moderator – the person that arranges the meeting, moderates between the author and the inspectors on areas of contention and does the Follow-up following the inspection meeting.
b. Author – This is the person that has authored source code and the person that will provide the overview, be involved in preparation, meeting and rework.

c. Reader – This is the person that reads source code and finds defects, problems with the code base.
d. Tester – This person verifies that the defects have been fixed either by verifying the code or by writing test code that determines the fix. The tester may also read the code from the testability stand point. This role can be assumed by the author or one of the readers.

Fagan Inspection in practice:

Usually, the role of the moderator is assumed by the team lead or by a senior developer in the team. The author or the moderator sets up the meeting time, prepares the printed copy of the materials to go through. The inspectors are suggested either by the author or by the moderator. The involvement of testers is not very common – usually the test feedback comes in at the time of automated tool development where lack of a specific hook or API hinders automation.

Fagan inspection process imposes some people and process overhead. The people overhead involves having a moderator, reader, author and tester roles defined and involving suitable individuals. The process is heavy weight on the authors and the readers in several respects –

a. The author needs to do some work to schedule the meetings, distribute the material under review, do overviews etc. As a result, the author will delay reviewing code till the code base is fully developed. This usually causes a large body of code to be reviewed at once. As a result, usually the entire code base doesn't get reviewed with the thoroughness that is necessary making the process somewhat ineffective. Sometimes, code reviews will cause design changes for maintainability or for compliance with a known check list. If the code is reviewed at the completion of a project, the likelihood to change code for maintainability, compliance is less likely as one risks destabilizing the code base.
b. The moderator is usually a person other than the author. This usually is a team lead or a senior developer in the team. When there are several projects in the team, the burden is on the moderator to make sure that all the reviews are scheduled and will happen. It is also difficult for a moderator to ensure that all the fixes suggested in the review meeting are made in the code base. In order for the Fagan inspections to succeed, it usually requires a very committed moderator. In practice, the code reviews seem to stop or not happen after a while as it is difficult to police the code reviews over a long period of time.
c. The readers/inspectors may not be fully engaged in a meeting. This could happen for many reasons
    a. If the reader/inspector is busy with his/her work, (s)he might not have the time to be prepared for the review. This is the case in most of the reviews.

b. The time of the review meeting may not be convenient to the reader.
c. Some readers/inspectors may not be comfortable reading printed code in a meeting. Usually a reader is more effective if he is able to choose a time of his choice in his preferred environment, using his favorite tools, editors for review.

While Fagan inspections will produce the results if the process is fully implemented, in reality, the process is hardly implemented over a long period of time. The process suffers from the inability to scale to large teams and can function very well in small well knit teams. Even in a small team, it takes a lot of effort to implement, maintain the Fagan inspection process. There are other studies [4] that note some of the problems in maintaining the Fagan Inspection process.

Alternative to Fagan Inspection:

Fagan Inspection can be reduced in complexity to be more light weight and effective.[5] In the light weight process, there are three parties that have responsibilities.

a) The developer – This is the person that wants his code to be reviewed. He is responsible to make sure that his code is reviewed; the issues raised are resolved before it is checked in to the source tree.
b) The team (reviewers) – This is the group of people that work with the developer on a project or people knowledgeable in the area the developer is working in.
c) The lead or the person tracking code reviews on a periodic basis.

Setup:

It is necessary to setup a file server to hold the code review files. This is a common location where all the team members' code review files will be stored. A share can be setup such as \\nisdev\codereviews and each of the developers can have an individual project based on e-mail id. For example, a person with e-mail id johndoe will create a share \\nisdev\codereviews\johndoe.

For project1, John Doe will create a share such as \\nisdev\codereviews\johndoe\project1 and drop the code in that share. A date can be attached to the project if the review is incremental to enable the reviewers to distinguish the changes easily.

e.g.,

\\nisdev\codereviews\johndoe\project1\07102003\ and
\\nisdev\codereviews\johndoe\project2\07202003

or
\\nisdev\codereviews\johndoe\project1\drop1\ and
\\nisdev\codereviews\johndoe\project1\drop2.

Reviewers can easily diff between these two by using standard tools such as windiff. This enables the reviewers to figure out the changes in the code base without having to spend a lot of time.

It is important to keep the files around till after the review is completed with all the reviewers having signed off on the code.

The author's responsibilities:

The author doesn't check-in the code till it is reviewed and signed off by the reviewers. This applies to any code that is developed including C, C++, SQL, C#, VBS/WSF, ASP, VB.Net etc. It is the developer's responsibility to make sure that the code is reviewed prior to check-in. For large projects, the developer should do incremental code reviews. Incremental reviews help both the developer and the reviewers. For the reviewers, small chunks of code (e.g., a class with unit tests) are easier to review. In some situations, design changes may also be suggested. In this situation, it will be easier for the developer to incorporate the changes in to the code that is being developed.

When submitting the code for review, the developer includes the title of the project with a brief overview as appropriate. If it is a bug fix, usually the bug title is included. A check list is included that has the following check points.

1. built on all relevant platforms – yes/no
2. unit tested – yes/no
3. static code analyzer output – attached or n/a
4. Formal self review completed – yes/no

After this is done, the files necessary for review are packaged and put in a share. The share location is pre-determined for the team. For bug fixes, a windiff package may be included for easy review of the changes. This will save the reviewers time and effort.

The location of the project files and the directory should be included in the e-mail sent to the reviewers and the team. For new code, the reviewer should be able to compile, build or possibly run the unit tests from the share where the code is dropped.

After this is done, an e-mail is sent to the team including the developers and testers of the project. The author specifies the estimated time to check-in the code and nominates two reviewers to review the code.

Sending out e-mail to the team allows the team members to know the progress being made with different projects. Also, some team members might be interested in participating in the code review of a specific project. This process allows them to voluntarily participate in the review of code that is of interest.

After the team has reviewed the code, it is the developer's responsibility to resolve the outstanding issues with the reviewers. In case of disagreements, the developers, reviewers can consult with senior developers in the team or the team lead to resolve issues. The developer follows the same process outlined above for resubmitting the code till the code is signed off by the reviewers.

A sample e-mail that describes the process followed by the developer is attached below.

From: Sandeepan Sanyal (SUMAN)
Sent: Tuesday, July 29, 2003 11:52 AM
To: Mark Schurman
Cc: SNS Developers
Subject: Code review: PEAP inner identity test for PEAP-EAP MSCHAPV2

Spec: n/a
self review: yes
x86/ia64/amd64 build: n/a *(trivial fix; see below)*
x86 unit test: yes
prefast: n/a
typo: n/a
W4: n/a
Lint: n/a

Suggested reviewer: mschur

Sources: \\NISTest\CodeReviews\ssanyal\identityreq\07-28-03\protocol\peap

ETA for checkin: 7/30

Details:

Script tests server if inner identity varies from outer identity. This case is important for security.

Thanks

Sandeepan

This information is subsequently entered and tracked in a spread sheet or website.

Once the team or reviewers have signed off on the code, the code is checked into the source tree. The developer should update the tracking spread sheet or the website to indicate that the code is checked in. The developer is encouraged to notify the team via e-mail that the code is checked in. This tells the team that all work related to the review is completed. Once the check-in is complete, the author can cleanup the code review directory or move it to another place for archival purposes.

Reviewer's responsibility:

The reviewers should allocate time as necessary to review the code assigned to them. In case the reviewer isn't able to review the code in a timely manner, he should notify the author with a substitute reviewer or indicate the inability to review code in a timely manner.

The comments should be saved to a file in the review directory. The directory should include the authors e-mail address as a suffix when all the comments are included in one file. Review comments should be interspersed with code to allow the dev to figure out the code in question easily. It is possible to use document editors such as MS Word to contain the source file where multiple authors can comment on the same code. As an example a file foo.cpp can be stored as foo.cpp.doc and web view enabled. The reviewers can insert comments into this doc. If the reviewer finds all the code good, a comment such as 'pass' should be included to indicate that the code is good as is.

Once the review is complete, the reviewer should update the date the review is completed in the tracking spread sheet or web page. The reviewers should provide suitable comments for the developer in e-mail. This allows the developer to update his code or check-in the code if no issues are present in the code.

Code review tracking:

The lead or the person tracking code reviews should go through the outstanding code reviews on a periodic basis and make sure that the reviews are happening as they should. In case the reviews aren't happening as they should, the lead should take the appropriate corrective action with a specific individual or the team as a whole to make sure that the reviews continue to happen.

An example of a spread sheet that is used for tracking code reviews is noted below:

Title	Developer	Created	Status	Reviewer 1 status	Reviewer 2 status
Traffic.exe	Jersmi	7/28/2003	Checked in on 8/4/2003	JaySri completed on 7/30/2003	ThiruB completed on 7/29/2003
Traffic library	JerSmi	8/1/2003	Checked in on 8/8/2003	PhilipW completed on 8/4/2003	ThiruB completed on 8/5/2003
RACOM	MarkWo	8/7/2003	Initiated	Jaysri completed on 8/7/2003	ThiruB – in progress, ETA 8/8/2003

Results:

The alternative to Fagan inspections was tried in my team. It took some time for the team members to get used to the code review process. Over a period of two years, the team did around five hundred code reviews. Every bug fix and code change was reviewed prior to check-in. This is a significant improvement over the Fagan inspection method which was used in the team previously. Using Fagan inspection, in five years the team did around twenty code reviews. It has been our finding that the light weight alternative to Fagan inspection is highly effective in a dynamic software development environment. It is also been observed through this study that continuous participation in code reviews over a period of time improves the skill level of both the reviewer and the author. The reviewer is able to find defects more effectively and the number of mistakes made by the author also decreases.

Summary:

Code review is static code analysis done by human beings of varying skill levels and quality. The variation in the skill set of different individuals is going to impact the quality of the code reviews positively or negatively.

However, it is well known that code reviews are effective in finding many defects in the code base. Fagan inspections while extremely popular are not generally effective in practice. The light weight alternative presented in this paper obviates the need to train the inspectors, trackers and the author. It also doesn't have the

overhead of setting up meetings and printing material ahead of the review. As a whole, this process much more light weight and it makes it effective.

It is also proven that over a period of time, code review participation is an effective way to train new comers to the team on common coding standards and how to write good code.

Bibliography

[1] Inspections – Evolution and History 1972-2001 – Michael Fagan.
http://www.sdm.de/download/sdm-konf2001/f_7_fagan.pdf
[2] Michael Fagan - **Advances in Software Inspections**
IEEE Transactions On Software Engineering, July 1986
[3] **"Design and code inspections to reduce errors in program development," by Michael Fagan**
IBM Systems Journal, Vol. 15, No. 3, pp. 182-211
[4] Fagan inspections. http://www.cs.virginia.edu/~cs201/Lectures/inspection.pdf
[5] Code review process explained – Mark Schurman, Thirumalesh Bhat; Microsoft Corporation.