Developer Perceptions of Process Desirability: Test Driven Development and Cleanroom Compared

Jonathan Percival  
Utah Valley University  
jonathan@databaseconsultinggroup.com

Neil Harrison  
Utah Valley University  
neil.harrison@uvu.edu

Abstract  
Design and coding processes are an important part of an overall project methodology. However, perceptions of these processes influence how developers use them. A process perceived as desirable is more likely to be sustainable than one that is not. Test Driven Development has gained popularity in recent decades and we compared its perceived desirability with that of Cleanroom. Developers were given projects to complete using the processes and reported on their experience via a survey. Survey results were analyzed statistically to see if perceptions varied. No significant difference was discovered between the two processes in any of 5 subjective categories rated and developer comments indicated they would prefer a hybrid process. We recommend developers integrate ideas from both TDD and Cleanroom into their development processes.

1. Introduction

It is of great interest to determine which processes are considered to be the most desirable by developers as their perception of the process will dictate how they utilize it. If they find it to be undesirable they may abandon its use and lose any benefits it provides. Conversely, a developer that likes a process is more likely to apply it fully and correctly. Rainer et al. have shown that developers will sometimes favor local opinion over empirical evidence when adopting new processes which makes perception even more important [29]. An undesirable process may fall out of use even if it does provide empirical benefits. Additionally, developers are often skeptical about the benefits new processes provide and do not want to invest resources into a process that they don’t perceive as desirable [30]. The abandonment of a process could result in the loss of productivity, with corresponding long-term implications for organizations adopting them.

In this study we investigate two development processes that are used at the code design, implementation, and testing phases of a development project and compare how developers perceive their desirability. We are not studying end-to-end development methodologies. In particular, this is not a study of high-level methodologies such as Agile vs. Waterfall but rather of development processes that could be incorporated into either system at the design, implementation, and testing stages.

1.1 Test Driven Development

Test Driven Development (TDD) is the first process that we studied. Beck describes it as “…a set of techniques that any software engineer can follow, which encourages simple designs and test suites that inspire confidence…The goal is clean code that works…Write a test. Make it run. Make it right [1].”  

Martin offers three laws [17] which should be followed while executing test driven development, which are writing the test first, writing only enough code to fail the test, and finally writing no more code than is sufficient to pass the test. In practice this means that there is a profound emphasis on writing unit tests before writing code. A developer writes a set of test cases before writing any code and ensures they all fail. He or she then writes code so that each test case passes, refactoring code as they go, until all the unit tests pass and a reasonable design emerges. Following TDD fully requires automated regression tests to be run after every code change. Since there are so many unit tests available problems with the code get detected and corrected quickly. This approach claims to guarantee that the software is testable and produces the correct output as long as the test set was complete.

Research into the architecture produced by TDD suggests that it has a lower cyclomatic complexity [3] and higher initial quality [14] which presumably can lead to lower maintenance costs. The tests produced by TDD can be used as documentation [11] which reduces the time spent writing and maintaining it as well. Developers using TDD spend less time debugging and have more confidence in the code they write [13]. It has also been shown that TDD is easily taught as the precepts are readily picked up by students [12]. Anecdotally, there many developers in forums online who claim to use TDD successfully on their projects [26,27].
There are some trade-offs to using TDD. The nature of TDD means that it requires considerable discipline to execute properly. Strict TDD dictates that at least one test be written for every requirement that a developer wants his code to fulfill (use cases, test cases, etc.). In some projects where TDD is implemented faithfully every line of code has a unit test associated with it [1,14] and occasionally even more. Obviously, this requires considerable effort. It has also been reported that the length of the development cycle increases by 14%-16% [2,11,14]. Pančur reports marginal increases in productivity [16] (so marginal in fact that the author finds the difference between test-first and test-last unimportant). Additionally, one study indicates that developing with TDD leads to code with higher coupling [3].

1.2 Cleanroom Development

Cleanroom was first developed in the late 1980’s by Harlan Mills and his colleagues, and in recent years has been championed by Allan Stavely. He describes some of the goals and methods of Cleanroom as “…to avoid or eliminate as many defects as possible before the software is ever executed. Elements of the method are: specification of all components of the software at all levels, stepwise refinement…verification of all components by development team, no unit testing, no execution at all prior to certification testing [4].”

In contrast to TDD’s highly iterative approach, Cleanroom follows the design mantra of “Do it right the first time [7].” The core of the Cleanroom process lies in the specification and verification of the code. A developer specifies the code by writing out its intended function in pseudo-code, English, mathematical equations, or in any other form that is appropriate [4]. This is then broken down into smaller parts using box structures (essentially pieces of code that can be treated as a complete unit and whose behavior is well-defined [20] which can then be broken down into smaller box structures) with each part getting its own specification down to the lowest level. This forces the programmer to think carefully about what each part is supposed to do and also acts as documentation for the code. Once the specification is complete it is implemented in code. After implementation the code is verified by a group of three to eight people. Each line of code is checked against its previously described intended function [6] to see if does what it is supposed to do. Mathematical proofs are only used when necessary but the concepts of functional decomposition and substitution are widely applied. The code is considered verified when all members of the group agree that the code matches its intended function.

The primary benefit of Cleanroom is regarded as being extremely low error rates in code. One study covering 15 different projects at IBM found that the error rate never exceeded 5.1 errors per thousand lines of code, that several projects had no defects at all, and that the average defect rate was extremely low [10]. Stavely reports similar results when teaching students about Cleanroom [4]. Early studies into Cleanroom [19] suggest the quality of the code improved. One of the authors’ colleagues mention they prefer Cleanroom because it allows them to get the “big picture” before beginning to code which in turn allows them to see potential use and test cases that may be missed when using a bottom up approach such as TDD. Additionally, there can be a significant amount of knowledge transfer about the code and project during peer reviews which increases the robustness of the development team.

As with TDD, Cleanroom requires quite a bit of discipline in order to implement fully. One criticism of Cleanroom is that it doesn’t fit well with modern object oriented programming since it treats code very procedurally although suggestions for improving that aspect of Cleanroom have already been made [9]. Another shortcoming is that the original methodology doesn’t handle project specifications well and that formal (logical and mathematical formalizations) methods for defining requirements improve both software quality and cost [8]. One study suggests that developers miss being able to execute code while developing using Cleanroom [19].

1.3 Similarities and Differences

In spite of the differences between the two processes both claim low error rates per line of code [2,4,8,14] which suggests that using either process will produce similar quality in the resulting software. Several of the studies cited compared TDD processes to “legacy” or “Waterfall-like” development processes [13,14] but none mention Cleanroom specifically so it cannot be inferred that using TDD improves error rates per line of code better than Cleanroom only that TDD was better than the (unspecified) legacy processes.

One paper suggests a closer relationship between TDD and Cleanroom saying that Cleanroom was an early forerunner to test-first methodologies since the design of the code is verified early in the process [28]. Errors are hypothetically caught very early in the development process; either as soon as the code is written (in TDD) or soon after the design for it is conceived (in Cleanroom).

There are two things about these development processes which make them particularly interesting for comparison. First, they are nearly opposite approaches. TDD is a bottom-up approach while Cleanroom is a top-down one. The ideologies behind them are different. Cleanroom assumes that good design and verification will produce clean, defect-free code while TDD postulates that good design can emerge organically from clean code. TDD forces a developer to test their code while Cleanroom validates design by
code inspections and walkthroughs. Second, there is nothing stopping a developer from using Cleanroom Development on an Agile project at the implementation level (e.g. he or she is assigned a task, designs the components, implements and verifies them before ever running the code) and vice-versa even though this may not have been done traditionally.

There are varying perceptions of the processes in general among developers. It’s conceivable that developers stick with whatever development process they first encounter in the workplace or in school or is preferred by their colleagues. Indeed, many who prefer TDD see those that practice Cleanroom as dinosaurs who simply refuse to give up their outdated ways. In spite of this we don’t actually know how people feel when they try them both. In particular, both processes require a significant amount of discipline to execute. This is an important consideration because it may mean that developers find them both a burden rather than an aid.

2. Prior Research

Previous research has focused on using objective measurements of code quantity and quality rather than a subjective measurement of how developers perceived it. These studies were done under different conditions (case studies and controlled experiments), with various subjects (professional developers and students), and measured the results for various metrics (code quality, development time, etc.).

The bulk of the current research on TDD was performed during the 2000s. Kaufmann and Janzen [13][3] compared two groups of test-first and test-last developers and found the test-first group was more productive in terms of lines of code and quality of code produced than their test-last counter-parts.

Williams, Maximilien, and Vouk performed a case study at IBM [14] where they found the application of TDD greatly reduced the defect rate. Williams and George [2] analyzed the use of TDD at three different companies. They concluded that TDD improved code quality, but noted that it lengthened development time. Bhat and Nagappan [11] performed case studies on two projects at Microsoft, and found similar results; code quality improved dramatically and development time increased by about 15%.

A recent study, which consisted of controlled experiments done by Pančur and Ciglarič [16] over a period of two years, reached similar but much less dramatic results. They concluded that a test-first process did improve code quality but the difference was insignificant. They note that it’s possible that the benefits of TDD are the result of a highly iterative development cycle rather than writing the tests first.

This is one possibility for why TDD has performed better in case studies than in controlled experiments.

Bowyer and Hughes did an assessment of undergraduate experiences with TDD [12] and found that students participated well in the agile process, understood the ideas underpinning it well, and claimed to have a good experience, but they did not specifically study the desirability of TDD.

Cleanroom research is scarcer than TDD but has produced broadly similar results. Shelby, Basili, and Baker did a wide reaching study of Cleanroom [19] and found that teams using it were more likely to deliver on time while meeting project specifications and producing code with lower complexity. Linger found similar results at IBM [10].

Langari and Pidduck compared faults per line of code for several development processes, including two versions of Cleanroom (with and without using formal methods for function specification) and found that Cleanroom using formal methods for function specification produced the lowest number of errors [8].

As with TDD there has been some research on teaching Cleanroom to students. Stavely reports on his experiences teaching a college course about Cleanroom [4] saying that in general the students were able to successfully understand and apply the principles of Cleanroom.

As far as we are aware, there have been no studies directly comparing TDD and Cleanroom or comparing developers’ perceptions of the two. Furthermore, there are very few studies reporting how developers actually perceive any development process. The Agile Manifesto [22] states that people should be favored over processes and this study provides some research in this area.

3. Research Questions

The principal question of this study is which of the two development processes do developers perceive as the most desirable. We divided the idea of the “desirability” of a development process into smaller pieces which together represent the overall value of the process as perceived by the developer. These are:

- The effectiveness of the process at producing high-quality software.
- The ability to accelerate software development.
- How difficult the process is to use.
- How enjoyable developers find a process to use.
- The likelihood that developers will continue to use the process in the future.
By gathering data on each of the items in the list above we determined which process was perceived as being the most desirable. Our hypothesis was that developers would prefer TDD over Cleanroom. This hypothesis was influenced by the current popularity of TDD with mainstream developers.

4. Study Design

Experimentation in Software Engineering [5] suggests three main strategies for empirical studies: Case Studies, Controlled Experiments, and Surveys. A controlled experiment is one in which separate groups are studied under similar conditions where the factor under study is varied to see if its influence can be identified. We chose to do a controlled experiment in order to compare separate treatments (developers using different processes and all other factors being equal) which are described below that allowed us to see which development process was the most desirable according to our subjects.

The controlled experiment consisted of a number of subjects completing two small programming projects. One project was completed using the Cleanroom process and the other was completed using TDD. At the completion of each project the subjects were given a survey about their experience which asked them about their perceptions of each development process. The survey results were then analyzed statistically.

The subjects of the study were 25 software engineering students at Utah Valley University. We selected these students because they already had some development experience (all had academic experience and some were professional developers) which would mean that the difficulty of the projects would have a minimal impact on whether they liked the project or not by causing them excessive frustration and also because they were all in the same software engineering course. Selecting the students from the same class ensured there was some uniformity in their academic experience and ensured they all received the same instruction regarding the development processes being used. All students were instructed on how to use each process and told to follow it as closely as possible while completing the programming assignments.

The two programming projects selected were designed to be nearly equivalent in difficulty. The first was to find the longest palindrome within a sequence of characters and the second was to implement a priority queue that would sort inputs based on their importance. The authors implemented the solutions themselves and judged the difficulty to be about the same. Both projects were checked against the rest of the curriculum to make sure that the students had not completed them already.

The students were randomly divided into four groups, with each group having to complete both projects using a different process for each. The order in which each project was completed and which process was used was varied in order to eliminate any potential biasing due to the ordering of the processes or projects or any difference in difficulty between the projects.

The survey given at the end of each project was the same and consisted of six questions designed to make the students think about their experience with the processes and in particular address the research questions presented. The first five questions were rated using a Likert scale [25] ranging from 1-5, with 5 being the highest (Strongly Agree), 1 the lowest (Strongly Disagree) and 3 being neutral. The last question required the subjects to respond with the number of minutes it took to complete the project.

- How effective was the software development method at producing high quality software?
- Did the method help you develop faster?
- How easy was it to follow the method?
- How well did you like following the method?
- How likely are you to use this method again?
- From start to finish, how much time did you spend on your project using the method?

Perceptions are made up of opinions about many things and we tried to capture the subjects’ opinions about various aspects of the processes with these questions. Once both projects were complete the students were asked which method they preferred and to (optionally) write any comments they had down. They were not solicited to write down anything specifically, but rather record their general impressions about the processes.

For the TDD project the students were instructed to write test cases first and then implement their code so that the test cases passed. For Cleanroom, the students were instructed to do design work first and then test their design by manually stepping through it, and once they felt it was correct to code, compile, and run it to verify it worked.

For the purposes of this study we restricted the use of TDD to a very small scope due to resource constraints. The subjects were students so there was a limited amount of time to teach them about the development processes and for them to execute the projects and respond with the surveys. The projects did not generate a large number of unit tests. For our small tasks the developers created a set of tests that had to be passed for the task to be considered complete, and then wrote code so that each test passed one by one, refactoring and simplifying the code as they went until
all the tests passed and the developers found the software design satisfactory.

Similarly, we also restricted the use of Cleanroom to a very small scope. We completely excluded the processes of requirements gathering and certification testing and focused on the specification and verification processes of development.

In both the case of TDD and Cleanroom these limitations don’t prevent the study from being relevant; it simply narrows the scope from the overall project methodology to the actions of a single developer performing a small design, implementation, and testing task in a larger project. The actions performed by the test subjects mirror the actions of a developer using these processes in the real-world on a small task.

5. Results

The survey data collected can be divided into three primary groups. The first group is the data collected from the Likert scale survey questions. The second is the numerical data collected determining how long each project took. The third are the comments provided by the developers on their perceptions of the development processes.

5.1. Survey Data

The first set of data was analyzed using a Chi-Square test, which is considered an appropriate analysis for determining statistical difference for this type of categorical data [21]. The numerical data was analyzed using a simple T-Stat. The comments were read and categorized according to their content, and were used to aid in the interpretation of the other results.

The Likert scale data collected is shown in Figure 1, allowing us to characterize the results before a more detailed analysis. Broadly speaking, the results for both processes are similar. For nearly every category the range of responses varies from Strongly Disagree to Strongly Agree. Furthermore, the proportions of each category are similar for both processes and the modes for each category are also similar. The greatest discrepancy between the two processes occurs in the Faster category with Cleanroom receiving more negative responses (but not fewer positive ones). There is no indication of a “landslide” victory for either development process. Interestingly, nearly an identical number of developers agreed that TDD and Cleanroom were effective processes.

We tested for any significant difference between the developer responses to the survey questions for the two processes. For each category we calculated the expected responses from the observed responses, and then used the Chi-Square test to determine the significance of the differences. The results are found in Table 1 for a 95% confidence level. There was no significant difference between the two processes in any of the categories on the survey.

![Figure 1. Survey results](image)

Table 1. Chi-square test results for survey data

<table>
<thead>
<tr>
<th>Category</th>
<th>Chi-Square</th>
<th>Significant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effective</td>
<td>3.115</td>
<td>No</td>
</tr>
<tr>
<td>Faster</td>
<td>6.090</td>
<td>No</td>
</tr>
<tr>
<td>Easy</td>
<td>4.955</td>
<td>No</td>
</tr>
<tr>
<td>Like</td>
<td>2.602</td>
<td>No</td>
</tr>
<tr>
<td>Use Again</td>
<td>3.010</td>
<td>No</td>
</tr>
</tbody>
</table>

a. Critical Value = 7.815

5.2. Time to Completion

We also tested for a significant difference between the processes in the reported time to completion at a 95% confidence level. The results of this analysis are in Table 2.

Again, there was no significant difference between the two processes. The mean time to completion for the TDD project was just over 2 ½ hours while the Cleanroom project took 3.

Table 2. T-stat results for time to completion

<table>
<thead>
<tr>
<th>Category</th>
<th>TDD</th>
<th>Cleanroom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Minutes</td>
<td>157.12</td>
<td>180.36</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>81.45</td>
<td>96.92</td>
</tr>
<tr>
<td>Variance</td>
<td>6634.11</td>
<td>9393.82</td>
</tr>
<tr>
<td>Range</td>
<td>40-360</td>
<td>20-380</td>
</tr>
<tr>
<td>T-Statistic</td>
<td>0.93</td>
<td>2.01</td>
</tr>
<tr>
<td>Significant</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
5.3. Developer Comments

We analyzed the comments to see if we could gain any insight into the subjects’ perceptions of the processes or find any common themes relating to the results of the data analysis. We received 90 comments overall (3.6 per subject). 36 of the comments referred to the developers’ experience with Cleanroom, and 26 were specifically directed at TDD. The remaining comments considered both processes.

We grouped the comments the developers made according to the content. The most common comment made by developers about TDD was that they liked the immediate feedback it provided on the code that they wrote. This indicates a large percentage (six developers) of the test group found this aspect of TDD appealing. Four developers also commented that they used TDD at work (indicating use of TDD is common among professional developers). Four more developers said they found it difficult to know what to test while using TDD.

In a related vein, three developers mentioned that it was difficult to sit down and code with zero upfront design since they didn’t know where to start on the projects. This suggests that while the developers enjoyed the immediate feedback on the work they had done they did not like the lack of direction that came with using TDD. The rest of the comments for TDD only occurred once and ranged from calling TDD tedious to praising it as the main development process in the real world.

The comments regarding Cleanroom were a little more numerous and varied. Nine subjects commented that the use of Cleanroom improved design decisions, primarily by making a developer more aware of design trade-offs. Six of the subjects found it difficult to not use the IDE while writing code. The reasons given ranged from not having full access to the language’s library, to not having access to refactoring tools, to simply not having the language’s syntax committed to memory.

Surprise about the effectiveness of Cleanroom was also common among the developers, with five mentioning they found that Cleanroom exceeded their expectations in some way (fewer bugs, faster, etc.) or that they had changed their previous opinion about it. Several developers (four) felt that Cleanroom was more suitable for experienced developers who could evaluate that the code met its intended function more accurately or who understood their languages libraries thoroughly and understood which pieces would need to be implemented and which were already available. Finally, two subjects mentioned they found Cleanroom restrictive (they wanted to test as they went), and two felt it took longer than TDD. The remaining Cleanroom comments also only occurred once and ranged from calling Cleanroom “archaic,” to noting that they had to code twice (once on paper and once in the IDE).

In general, the comments on Cleanroom were more varied than those about TDD. Additionally, it’s clear the developers had less exposure to Cleanroom before the study since they found it surprising and none noted using it at work specifically.

Nearly half of the developers (eleven) mentioned that both systems had their strengths and that they would use either as required. Several said that the size of the project determined which process was most effective although they were split about which was better for large projects and which was better on small ones. There was quite a bit of overlap (six developers) between those developers who said they would either development process as required and the nine who said they would prefer a hybrid approach incorporating elements from both processes in the study or other (unspecified) processes. Three developers commented that they felt that either process was superior to no process.

In general, the comments on Cleanroom were more varied than those about TDD. Additionally, it’s clear the developers had less exposure to Cleanroom before the study since they found it surprising and none noted using it at work specifically.

Nearly half of the developers (eleven) mentioned that both systems had their strengths and that they would use either as required. Several said that the size of the project determined which process was most effective although they were split about which was better for large projects and which was better on small ones. There was quite a bit of overlap (six developers) between those developers who said they would either development process as required and the nine who said they would prefer a hybrid approach incorporating elements from both processes in the study or other (unspecified) processes. Three developers commented that they felt that either process was superior to no process.

The remaining comments evaluating both processes occurred no more than once, and ranged from not liking either to both slowing down development. Figure 2 shows a summary of the preceding paragraphs, including all groups that occurred more than once.

6. Interpretation

Analyzing the data yields some interesting insights. The variance for the data with Cleanroom is higher, both numerically and with regards to the comments received which ranged from deriding it as “archaic” to expressing surprise about how effective it was. This suggests that Cleanroom was more polarizing than
TDD or that developers had a harder time adapting to Cleanroom.

The subjects rated the processes as being nearly equivalent for creating high-quality software and both above the neutral point of three. This suggests that the developers’ perception is that either process is capable of producing satisfactory results.

Individually, personal preferences varied greatly with the ranges running from the bottom to the top of the scale. We can infer that each developer’s affinity for each process varies greatly. In light of the comments we can safely say that developers believe that neither system is perfect and that both can be improved by either hybridizing them or incorporating features from other processes.

There is no significant difference at 95% confidence for how fast the developers perceive each methodology to be. This has implications for the long-term use of the development process because the perception of time passed could also be an indicator of how tedious a developer feels a methodology is to follow. This suggests that either process would be equally fatiguing as the other.

Additionally, the mode for TDD with regards to being faster was three which indicates that the developers felt like they developed just as quickly with or without it. This suggests that developers don’t find following TDD to be more tiring in spite of the higher discipline and more time required writing large numbers of tests. Since the data shows no significant difference between developer perceptions of TDD or Cleanroom with regards to speeding up development it’s probable that developers do not perceive either process as taking more time than the other. However, the difference is close enough to the critical value (favoring TDD as being faster) that future studies may refute this conclusion.

There is no significant difference between the time taken to complete the projects using the different processes. This suggests that the processes are closely matched in terms of raw output. The data shows that the developers feel that the processes are of similar difficulty to follow as there is no significant difference between the two and also that they probably require similar amounts of discipline.

The number one complaint against Cleanroom in the comments was that it forbade the use of an IDE to write code which precluded the use of code-completion and refactoring tools. This is actually not the case with Cleanroom. The subjects were instructed to use pseudo-code (outside of an IDE) while designing the components and then allowed to use an IDE once the design was complete and validated to flesh out the code. Many of the subjects simply wrote all of the code outside of an IDE which caused them significant annoyance if they had not committed a language’s syntax and libraries to memory. It’s possible that the developers would have liked Cleanroom more if they had used an IDE instead of doing the whole project without its assistance.

In the case of TDD it’s interesting to note that it was perceived by the developers as being of similar difficulty to follow as Cleanroom but there were several comments (six) saying that TDD is hard without a design phase or that the developers did not know what to test while using TDD. In fact, the projects given to the students represent what could be considered a “best-case” scenario for TDD since the specifications for the project were already defined, in contrast to many projects that may lack project specifications completely leaving a developer with even less direction and making it more difficult for them to find a starting point for implementation.

The data for how likely the developers were to use each process again showed a trend similar to those of the other results. In this case the rating for Cleanroom is somewhat lower than TDD but not significantly so at a 95% confidence level. Some of the developers mentioned using TDD in the workplace so it’s possible that TDD’s higher rating for being used again is simply a reflection of this. No developers mentioned using Cleanroom in their workplace.

Developers also rated the two methodologies the same for how well they liked following them. Additionally, they both hovered around the neutral point (i.e. they neither like nor disliked it).

The comments once again shed some light on why this is. The most common comment of all (in nearly half the surveys) was that the developers thought the methodologies were too extreme and they would prefer a hybrid approach between the two which allowed for some upfront design and for some coding to be done via tests. Developers said that they would prefer incorporating ideas from other processes suggesting that neither TDD nor Cleanroom is optimal and that developers prefer a more flexible approach to software development.

6.1 A Hybrid Approach

As a partial answer to what a hybrid approach would look like we make several suggestions based on comments given. This is an important consideration based on the fact that the two most common comments were the developers wanted to use both systems where appropriate or hybridize them.

Developers mentioned that they felt doing design work upfront improves the overall design of the software and the developer’s awareness of how it’s supposed to work. Doing this also simultaneously eliminates the two biggest criticisms of TDD as well. These were that writing tests without a design was difficult and knowing what to test was tricky. One subject mentioned they “liked being able to see what I was coding,” referring to the fact the Cleanroom gave them a big-picture view of the code they were going to
write. Taken together, this suggests at least some upfront design.

Judging from the comments many developers don’t like doing design work outside of an IDE because they aren’t familiar enough with a language’s libraries to know which parts of the design need to be implemented and which parts are already present in the library. As an example Microsoft’s .Net 4.0 framework reportedly has over 10,000 public classes available to the developer [15]. Using an IDE makes navigating the library easier and helps developers avoid reinventing the wheel.

Combining these features would suggest a development process with the following characteristics: High level design would be accomplished outside of the IDE, via class diagrams, box structures, etc. in a peer reviewed (or even collaborative) environment. More detailed design would then be accomplished in the IDE by writing out intended functions as is typically done in Cleanroom. Actual implementation of each function would take place in the same way as TDD. A set of tests would be written for each intended function and each test made to pass accordingly. The interface for the function would already be determined by the upfront design so less refactoring would be necessary but the code would still benefit from the same thorough validation and immediate feedback while coding.

Such an approach could have several advantages and disadvantages when compared to either TDD or Cleanroom which would need to be empirically verified. It could produce a cleaner architecture than TDD, allow developers to better understand the system they are working on beforehand, potentially produce less refactoring than TDD, allow the development team to be more robust since the peer-review process would allow knowledge transfer, and boost developer confidence in the code. Some of the potential disadvantages include a lengthier development process than TDD (due to planning and peer-reviews stages, although this may be offset by less refactoring), and simply being rejected by both the Agile and Waterfall communities since it follow neither strictly.

It’s worth considering that this could simply be a formalization of development processes already in use in the real world since often times corners are cut with regards to rigor in implementing a process or processes are simply blended based on developers’ prior experiences (or at least the desire to blend them exists as evidenced by our subjects comments) so it may a relatively simple thing for a development shop to adopt some of the suggestions. All that it would require is formalizing the process that is already in place.

7. Limitations and Threats to Validity

Several limitations apply to this study, due to the modifications we made to the development processes, the small scale of the projects used in the experiment, and the subjects being students at a university rather than professional developers. Additionally, there are some potential threats to the validity of our study’s methodology that need to be addressed, including the processes used in the study, the selection of the subjects, and the execution of the development processes by the subjects.

7.1. Limitations

A limitation is that the study was done at a small level, e.g. one developer working on a small task. While every large project is eventually broken down into micro-sized tasks, and for those small tasks the results are valid, it isn’t clear how it scales up to large projects. Further research will be required to confirm this and is suggested at the end of this paper.

The subjects themselves create another limitation to this study. Highly experienced developers may have different opinions about TDD and Cleanroom. However, many of the subjects in the study do work as developers professionally (even mentioning that they used TDD in their workplace) so the results do reflect the perceptions of real world developers to some degree.

A single small project may not provide enough experience with either process in order to create an informed opinion. We agree that the results represent only an early indicator, and that further study is in order.

7.2 Threats to Validity

One threat to the validity of the study is that the processes used did not represent true TDD or Cleanroom processes. With the limited timeframe the students had in mind we picked subsets of each process that were easier to follow but still included the core ideas of the process. In the case of Cleanroom we eliminated the use of box structures as well as group code review, and for TDD we eliminated the use of automated tests. We specifically included design time validation for Cleanroom and writing the tests first for TDD. This allowed early validation in both instances, which is the essence of either system. This means the study may represent a best case for each process in terms of developer perception since they were less strict. It’s significant that the students wanted to incorporate parts from both systems which suggest that they experienced each enough to see their respective limitations.
A related threat is that the students’ instruction on how to use the processes was quite limited. We cut down the processes to include only the essentials to understanding and applying them which is a common technique when teaching [4]. The students received the same amount of instruction in the processes as they would have in most Software Engineering courses; generally considered sufficient for students to be able to apply the principles of those processes.

The subjects were students in one Software Engineering course which limited the population from which subjects were drawn. However, this allowed more consistency in the teaching of the development processes and explanation of the projects to be completed and it allowed us to control for external variables better (we could ensure their previous academic education did not include the projects in the study and that they all had sufficient experience to complete the study among other variables). Additionally, resource constraints did not permit a large random sampling of either professional developers or students. However, the students had considerable variation in their work experience which helped approximate a random sampling. We also had nearly 100% participation (only one student refused) and it was a required course, so this allowed us to be as close to a random sample as possible while guaranteeing a minimum level of development competency sufficient to be able to complete the tasks.

The subjects likely did not execute the processes exactly as directed. Indeed, in the case of Cleanroom it’s clear that they did not follow the directions regarding the use of the IDE while implementing the code. Often times developers in the real world will stop using a process they’ve previously committed to [24] if they don’t perceive it as desirable (e.g. it doesn’t produce the results expected, it takes too much effort, it’s tedious, etc.). The focus of our study was to determine the subjects’ perceptions of the development processes, not to ensure they followed them strictly. In fact, if we had done so it would have altered the subjects’ perceptions of the processes (moving them away from the real world experience) since they would have needed to be closely monitored. Outside of pair-programming, developers do not typically have an observer critiquing their application of a process at every moment.

8. Conclusion and Recommendations

The principal research question of the study was whether or not developers’ perceptions of desirability differed between Cleanroom and TDD. We found that in every area we measured, there was no significant difference between the two, which cause us to reject the hypothesis that TDD would be favored.

On the whole, developers do not perceive either process as being superior to the other, but that both have their strengths and weaknesses. Based on comments made by the subjects developers appear to prefer a hybrid approach to development.

Coplien and Harrison [23] recommend that organizations should allow developers to control the development process used and we have outlined a potential development process that incorporates feedback from the subjects that could be considered a hybrid between TDD and Cleanroom. We recommend that developers consider a using a development process incorporating elements from both Cleanroom and TDD. Specifically, developers should ensure their process includes both some design so that they have a sense of direction (and an end goal) while coding and early feedback about the code a developer is writing in order to provide high confidence in its quality.

9. Further Research

The findings presented in this paper open up several possibilities for further research which may confirm or refute the results found herein.

The development process suggested that incorporates features from both Cleanroom and TDD could be tested. This would show whether developers truly prefer a hybrid approach or if that response was simply the result of dealing with two extreme processes. This would also give some indication on whether the processes could be strengthened by incorporating elements of other systems or if their relative strengths lie in their purity and by adding elements they would be weakened instead. We have suggested what one such hybrid approach would look like. Alastair Cockburn has suggested development processes incorporate a means for modifying themselves [18] and it would be interesting to see any similarities that have developed in shops that have been executing this idea.

It would also be interesting to do a similar study on a larger project and see if and how developers’ perceptions change over time with experience. It’s also possible that a larger project would open a gap between Cleanroom and TDD or that the size of the project would determine which process is preferred. For example, TDD has been criticized for generating an excessive amount of tests on large projects.

There was some indication that Cleanroom may have scored higher if it were altered slightly such as by having developers use an IDE for the entire process rather than only at the implementation stage. Another study could be done using slight variations on both TDD and Cleanroom to see if there are any other specific aspects of the processes that developers either enjoy or dislike. This would give greater insight on how developers perceive both processes as well as
enhancing the formulation of a hybrid process that uses aspects of both.

Replicating the study with a group of professional developers would help eliminate many of the constraints placed on the study. More time could be taken to teach them about the development processes (and they may have more real world experience using them) and the projects used could be larger.

Additionally, it would informative to see if each process has an affinity for a specific overall development methodology. We specifically excluded end-to-end methodologies in this study, but any project of non-trivial size has some methodology in place. For example, TDD might be better suited to an Agile environment.

10. Acknowledgments

The authors would like to thank Uwe van Heesch for his help in preparing this paper, and the students who participated in the study as subjects.

11. References


