Visualizing the TDD Cycle

Michael Hilton, Mihai Codoban, and Caius Brindescu

1 INTRODUCTION

Software processes have had little or no visualization aids. Without this kind of information, it is hard to grasp whether these processes are followed correctly. It is hard to find anomalies, and identify potential areas for improvement. Without visualizations that show the big picture, one has to go through tedious, time consuming and incomplete data to learn how a processes was actually implemented. One has to go through reports, interviews and mine software artefacts in order to gain a big picture understanding on how a particular team implemented a software process.

We intend to build visualizations that aid in analyzing developers TDD practice. Such visualizations would visually expose, for each developer, the main stages of the practice and how they evolve over time.

This kind of information benefits developers, managers and researchers. Developers can better improve their practice by analyzing their past behaviour. Managers can get a sense of magnitude of how well TDD is implemented in the team. Researchers can gain insights and better describe the theoretical foundations of TDD by finding patterns and exploring the divide between practical and theoretical TDD.

2 TARGET AUDIENCE/USER

The target audience for this visualization is anyone who wants to know about how the software development process happened.

Developers who want to analyze their own TDD cycle can use this visualization to examine their performance and look for ways to improve their practice.

Researchers analyzing the practice of TDD can use this to look for patterns in how TDD is actually used. Without this visualization, the data would be a series of JSON files that would be almost impossible to get meaning from just by looking at them.

Finally, managers analyzing the performance of those that they are managing can use this visualization to have an accurate picture of their teams performance.

3 HIGH LEVEL DOMAIN QUESTION/QUESTIONS OF INTEREST

Developers:
Q1: What does my TDD workflow look like?
Q2: How does my TDD workflow compare to others?
Q3: How does my TDD workflow change over time?

Managers:
Q1: How consistent are my developers in using TDD?
Q2: Who are the best TDD performers?
Q3: How much time do all developers spend in each TDD cycle on average?
Q4: How much code do all developers write in each TDD cycle on average?
Q5: What percentage of the code my developers write conforms to the TDD model?

Researchers:
Q1: What does a practical TDD workflow look like?
Q2: How does practical TDD differ from theoretical TDD?
Q3: How much time do TDD developers spend in each TDD cycle on average?
Q4: How much code do TDD developers write in each TDD cycle on average?
Q5: What percentage of code that developers write follows the ideal TDD model?

4 DETAILED QUESTIONS THAT USERS MAY ASK OF THE DATA

Developers:
Q1: What does my TDD workflow look like?
• What part of the TDD cycle to I spend the most time in?
• What part of the TDD cycle do I write the most code in?
• How close does my workflow compare with ideal TDD?
• What code did I write for a specific TDD stage?
Q2: How does my TDD workflow compare to others?
• How does the time I spend in each TDD cycle compare to others?
• How does the code I write in each TDD cycle compare to others?
• How close is my workflow to the ideal TDD in comparison with others?
Q3: How does my TDD workflow change over time?
• Is my TDD workflow consistent?
• How much time do all developers spend in each TDD cycle on average?
• How much code do all developers write in each TDD cycle on average?
• What percentage of the code my developers write conforms to the TDD model?

Managers:
Q1: How consistent are my developers in using TDD?
• How much time does each developer spend in each TDD cycle?
• How much code does each developer write in each TDD cycle?
• How much time do all developers spend in each TDD cycle on average?
• How much code do all developers write in each TDD cycle on average?
• What percentage of the code my developers write conforms to the TDD model?
Q2: Who are the best TDD performers?
• Which developers have a process that is closest to the ideal TDD process?
• Which developers spend the least/most time in each TDD cycle?
• Which developers write the least/most code in each TDD cycle?

Researchers:
Q1: What does a practical TDD workflow look like?
• How much time do TDD developers spend in each TDD cycle?
• How much code do TDD developers write in each TDD cycle?
• What percentage of code that developers write follows the ideal TDD model?
Q2: How does practical TDD differ from theoretical TDD?
• What are patterns that we can identify in how developers write using the TDD process?

• How does the time TDD developers spend in each cycle compare with our reference implementation of the TDD process?

• How does the code TDD developers write in each cycle compare with our reference implementation of the TDD process?

5 ABSTRACT OPERATIONS/TASKS THAT MUST BE SUPPORTED IN ORDER TO ANSWER THE DETAILED QUESTIONS

Compute TDD cycles: starting from low level text edits, cluster the edits according to the TDD cycle stages: red, green, blue.

DQ1a: Compute TDD cycles. Compare cycle stage times
DQ1b: Compute TDD cycles. Compare cycle stage size
DQ1c: Compute TDD cycles. Retrieve ideal TDD cycles. Compare user cycles and style with theoretic TDD variants
DQ1d: Compute TDD cycles. Retrieve code edits for specific cycle or stage.

DQ2a: Compute TDD cycles. Retrieve cycles for specific developers. Compare developer cycle times with other cycles.

DQ3a: Compute TDD cycles. Derive variance of TDD cycle metrics. Characterize distribution of variance
DQ3b: Compute TDD cycles. Derive variance of TDD cycle times. Characterize distribution of time variance
DQ3c: Compute TDD cycles. Derive variance of TDD cycle sizes. Characterize distribution of cycle size variance

MQ1c: Compute TDD cycles. Retrieve cycles for specific developers. Derive average TDD cycle times of developers.
MQ1e: Compute TDD cycles. Retrieve cycles for specific developers. Retrieve ideal TDD cycles. Compare with ideal TDD

MQ2b: Compute TDD cycles. Retrieve cycles for specific developers. Retrieve ideal TDD cycles. Retrieve cycle times. Compare with ideal TDD. Find range of developers with times closest and farthest away from ideal TDD.
MQ2c: Compute TDD cycles. Retrieve cycles for specific developers. Retrieve ideal TDD cycles. Retrieve cycle sizes. Compare with ideal TDD. Find range of developers with cycle sizes closest and farthest away from ideal TDD.

RQ1a: Compute TDD cycles. Retrieve cycles for specific developers. Derive average TDD cycle stage times of developers.
RQ1b: Compute TDD cycles. Retrieve cycles for specific developers. Derive average TDD cycle stage size of developers.
RQ1c: Compute TDD cycles. Retrieve cycles for specific developers. Retrieve ideal TDD cycles. Compare with ideal TDD

RQ2c: Compute TDD cycles. Retrieve cycles for all developers. Retrieve ideal TDD cycles. Retrieve cycle sizes. Compare with ideal TDD.

6 DATA TYPE ABSTRACTIONS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Expected value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event type</td>
<td>Nominal</td>
<td>textChange, testRun, etc</td>
</tr>
<tr>
<td>Event timestamp</td>
<td>Quantitative</td>
<td>UNIX timestamp</td>
</tr>
<tr>
<td>Event Source</td>
<td>Nominal</td>
<td>user, cut/copy/paste, etc</td>
</tr>
<tr>
<td>Text Added</td>
<td>Nominal</td>
<td>string</td>
</tr>
<tr>
<td>Offset</td>
<td>Quantitative</td>
<td>location from file start</td>
</tr>
<tr>
<td>Length</td>
<td>Quantitative</td>
<td>size of text edit</td>
</tr>
<tr>
<td>File Changed</td>
<td>Nominal</td>
<td>The path of the file being changed</td>
</tr>
<tr>
<td>Test Outcome</td>
<td>Nominal</td>
<td>Number of seconds the test took</td>
</tr>
<tr>
<td>Test Duration</td>
<td>Quantitative</td>
<td>Application start and stop</td>
</tr>
<tr>
<td>Launch</td>
<td>Nominal</td>
<td>Refactorings done</td>
</tr>
</tbody>
</table>

Data is gathered by a plugin that is installed by the user in her IDE. The plugin records all the events from the event table and sends them to a server.

7 ENCODINGS

7.1 Overview

For this design iteration we only focus on the single developer scenario. Thus we do not tackle design solutions for user questions that involve managers or researchers.

Our single user solution proposes the visualization of data at several abstraction levels, as [2] suggests. Thus users have three scopes at which they can analyze their TDD practice, varying from concrete to abstract:

Text change level. The user can browse through individual text changes to see how they relate to TDD stages. We call this visualization the TDD Atomic Chart

TDD cycle stage level. The user can browse through the stages of their TDD cycles. We call this visualization the TDD Spectrum Chart and is inspired by [1].

TDD cycle level. The user can browse through high level TDD cycles. We call this visualization the TDD Sun Chart and is reminiscent of radar charts.

Figure 1 shows a sketch of the single developer TDD workbench. The top bar contains the high level, abstract visualizations of the TDD cycles. The upper part represents the TDD Spectrum Chart while the bottom part represents the TDD Sun Chart.

The bottom bar shows the TDD Atomic chart. It is mainly a change navigation tool that users use to browse through granular code changes. The mapping from code changes to TDD stages is present via the top band.

In the middle there are three editors that show the code changes between two arbitrary points in the TDD Atomic Chart. The first two editors show the first and the last variant of the code while the third shows the diff between the two: green highlights for added text and red highlight for deleted text.

8 TDD ATOMIC CHART

8.1 Justification

This visualization allows the developer to navigate through the code events. Each coding event is abstracted away via a rectangle. Moreover, the events are grouped by the files in which they are performed. The event rectangles are sorted via their timestamps. Temporal data abstraction and data clustering techniques from [1] inspired this visualization: complex coding events are abstracted via rectangles and
Event types are clustered: event types are encoded via square color. A similar visualization is the Cluster Calendar View presented in [1].

The temporal succession of rectangles also provides the affordance of time flow. It resembles soundwave amplitude visualization from sound editors and music players. However, one key difference from a sound editor is that while the sound editor shows continuous data, we show cycle snapshots, so our visualization resembles more closely small multiples.

Since developers main concern with the tool is the study of TDD, we provide a mapping to TDD cycles at text change level as well thus presenting TDD relevant data at various abstraction levels. Since TDD stages are aligned with coding events, the users would tend to instinctively visually group code changes by TDD stage (gestalt principle: instinctively continuing an interrupted interval). The TDD stages are differentiated by their color. TDD literature uses red, green and blue for TDD stages.

Encoding (using Mackinlays Ranking):
1. Temporal succession of coding events (qualitative): position
2. Temporal succession of TDD stages (qualitative): position. Since coding events and TDD stages are on separate axis, both can use the position encoding.
3. Event type (qualitative): color
4. TDD stage (qualitative): color. Since coding events and TDD stages are on separate axis, both can use the position encoding.

9 TDD SPECTRUM CHART
9.1 Justification
The visualization in Figure 2 enables developers, managers and researchers to answer many of their questions. It shows the succession of TDD cycle stages in time. Each TDD stage appears as a rectangle. We chose rectangles because we can encode two TDD metrics with it: cycle code change size and cycle duration.

We ordered our encodings using Mackinlays ranking. Since the TDD cycle is defined by a temporal repetition of stages, the sequence of stages (categorical data) is encoded via position on the Ox axis. TDD stage size metrics are encoded using the rectangle width (time) and height (amount of code). Position being taken, they take the second encoding for quantitative data, length. The TDD stage type (categorical) is encoded via color: red for tests, green for development and blue for refactoring. Color occupies the second ranking for categorical data.

With this information the users can answer questions about what part of the TDD cycle they spend the most time on and how big their cycles are. They can also analyze how their cycles vary over time and whether any patterns occur.

Moreover, users can compare the TDD cycles for several users by showing several TDD time series side by side. This allows comparison of how much time different developers spend in stages, how big their stages are and whether they follow TDD at all.

10 TDD SUN CHART
10.1 Justification
The purpose of this visualization is to represent an entire TDD cycle by just one shape. See figure 3 By doing so, users obtain a singular mental image of an entire TDD cycle. [1] presents this technique as temporal data abstraction, in which complex, large temporal data is abstracted and mapped to qualitative entities, in our case, shape.

The visualization is based on a radar chart with three axis. The axis represent the size of each of three TDD cycle stages. These three axis generate the shape of a particular TDD cycle.
Instead of using straight lines to connect the radar chart points, we use curved lines. Together they create a deformed circle. Thus the unique size characteristics of each TDD cycle are encoded into the deformation of the circle. We used a circle and not a square due to gestalt principles: the vertices of the square introduce discontinuity in perception, and biases the viewer to the number 4. A circle on the hand has no vertices (points of discontinuity) and therefore does not introduce deformation bias. Thus, we hypothesize that circles show deformation in a more unbiased way than squares.

As with the timeseries chart, the sequence of TDD eggs are encoded via position, due to the temporal cyclic property of TDD cycles. We thus obtain small multiples snapshots of how the TDD cycle progresses in time. TDD cycle metrics are encoded via the next available encoding, length (the radar chart axis).

Encoding (using Mackinlays Ranking):

1. Succession of TDD stages (quantitative): position
2. TDD stage size metrics (quantitative): length

REFERENCES
