CITS5502: Assignment 2 – Simulating a Process

Ash Tyndall, 20915779

September 4, 2014

Contents

1 Introduction & Objectives

One of the great features of software processes is that to greater or less extents they enable predictability in the whole cycle of software development. Predictability is important, as it is required for all estimates, plans and deadlines generated throughout a project.

One key way in which software processes can create or increase predictability is through the integration of metrics into the test and repair process of software development. These metrics can form objective means by which progress can be measured on time scales, allowing comparison to similar projects, and affording a greater sense of visibility on the software processes progress, and possibly how long it has remaining.

However, it is important to realise that metrics are not infallible, and while they do provide objective truths, those truths are only proxies for the project's progress insofar as their definitions align with the abstract concept of "progress". Because of this, is important to know how easily these metrics can be influenced. Can a project appear fine through the lens of certain metrics, while not actually improving in quality? Can a commitment to the minimisation of a given metric lead to a worse, not better, quality product?

One way in which this can be tested is through simulation: By taking real-world data and a set of assumptions, one can trial different strategies for running a project's test and repair stage, allowing the generation of data based on test and repair strategies that would otherwise be overly time consuming, impractical, or unethical.

The objective of this report is to investigate these questions through the modelling of some such data. A set of assumptions will be defined, and a simulation model will be developed that will allow the derivation of a variety of real-world metrics from a set of test and repair strategies. The results of this simulations will be discussed in this report.

2 Assumptions

A variety of assumptions have been made while developing the simulation framework used within this project, these are outlined here;

- The data provided lists defects found each week independent of each other week.
- Defects can always be corrected in the week they are found.
- When there are no other criteria differentiating two defects, the least recent defect will always be selected first.
- "User impact" and "hours to fix" are both entirely accurate in all cases.
- t is equal to the number of the current week, $t = 1$ being the first week.
- p_f is equal to the number of people who are testers (finders) in the current week.
- p_r is equal to the number of people who are repairers in the current week.
- $p_r + p_f = p$, $p = 3$, $p_r, p_f \geq 0$.
- p_r and p_f must remain constant over a given week.
- Workers can be moved between testing and repairing roles with no efficiency penalty.
- Workers work at a constant and identical efficiency.
- The data provided assumes the number of bugs found per week were found with $p_f = 1$.
- Increasing p_f to n will result in the the bugs found in weeks $w_t, ..., w_{t+n-1}$ being found in week w_t. e.g. If $n = 2$, in week 1 we will find week 1 and week 2's bugs, in week 2 we will find week 3 and week 4's bugs, etc.
- For any week where $p_f = 0$, no bugs are found.
- n people work n times as fast as 1 person; efficiency increases linearly.
- To reduce the impact of outliers, smoothed values are used in graphs through. Smoothed values are created through the use of a 3-week moving average and the GNUPlot csplines option.

3 Measurement Process & Data Discussion

The data provided for this report could be measured through a a variety of means, both passive and active. Generally, one would expect that defect reports would be entered into some sort of database system as a matter of course, with impacts decided in collaboration between the software development firm and the end user. The majority of the data collected could be queried from such a database.

The "user impact" and "hours to fix" are very simplistic values that make the estimation process easier, but less accurate. Both this variables are numbers that represent much more complicated sets of variables.

One of the core assumptions about defects always being able to be corrected in the week they are found stems from the fact that there is no data on which day a defect was found in, making it difficult to determine what proportion of defects it is reasonable to expect could be fixed in the week they are found. If data was provided on a daily basis, the simulation accuracy could be improved in this way.

Further accuracy could be attained by replacing the "user impact" with set of values that measured both the visibility of the bug (e.g. the likelihood of encountering it), as well as the severity of the bug when encountered. This would allow much finder grained simulation of priority, as there is a clear trade-off between considering the difficulty of resolving the issue, how likely the defect is to occur, and how damaging the defect is when it does occur.

In a similar way, having further data on the general efficiency of workers over the week and day periods, the specific efficiency of each worker in both testing and repair and the performance penalty resulting from the change of roles would further assist in improving the simulation accuracy.

4 Fitting Defect Detection Curves

We can fit the number of found defects to the exponential curve below to provide an estimate of the number of defects remaining in the system after the test and repair phase concludes.

$$y = ce^{bx}$$

The following values were determined by fitting the set of found defects over the twenty week period provided using the method of least squares within Microsoft Excel, with a coefficient of determination (R^2) of approximately 99.1%

$$c = 60.113, b = -0.177, R^2 = 0.991$$

To then calculate the number of defects remaining, this function was integrated from the end of the test and repair phase ($t = 20$) to infinity, yielding the following result (numbers rounded to 3 decimal places);

$$\int_{20}^{\infty} 60.113e^{-0.177x}\,\mathrm{d}x = 9.840 \approx 10$$

Thus if our assumption that the amount of defects will continue to decrease in the exponential fashion we have defined, there are approximately 10 defects remaining in the system. This curve up to $t = 25$ is shown in figure 1 on page 4.

5 Metrics & Strategies

The interaction between metrics and strategies will be investigated to see how and why certain strategies cause certain metrics to differ. These variations will then be evaluated in terms of how those metrics affect the customer satisfaction vs the system reliability. Here we define a customer as being satisfied when the metric remains at a low level or trends downwards, or vice versa. The system is considered reliable with respect to a given strategy if that strategy would lead to little or no defects remaining in the software product at the end of the test and repair period ($t = 20$), and the strategy prioritises the most impactful issues.

Four metrics were chosen to be investigated, they were;

- Ratio of defects fixed to defects found
- Total importance of found defects still to be fixed

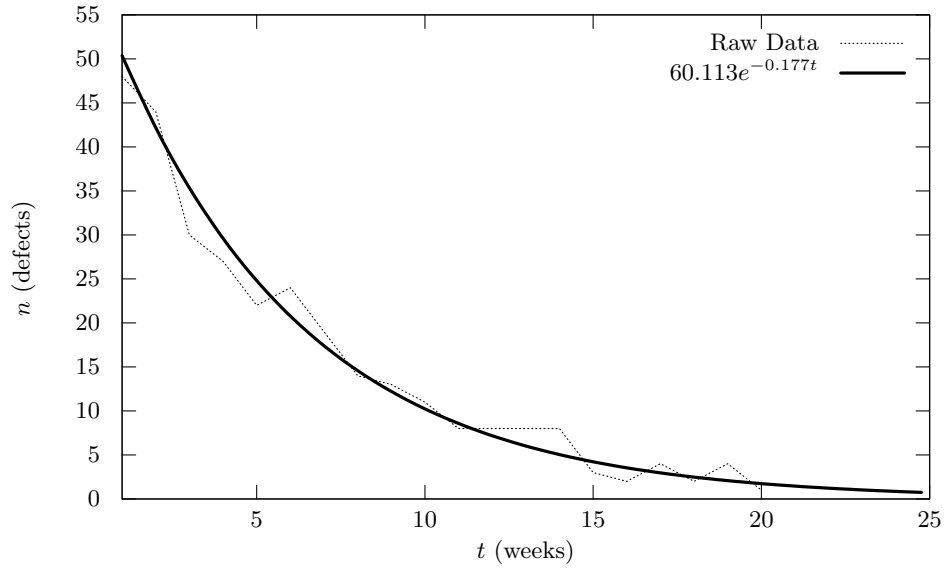

Figure 1: Exponential prediction vs. actual data

- Average time in queue of major defects still to be fixed
- Estimate of number of unfixed defects still in the software

Six strategies of prioritising defect resolution were run against a model calculating the results of these metrics, they were;

- Easy defects to hard defects, major defects to minor defects (i.e. Easy Major, Easy Minor, Hard Major, Hard Minor)
- Hard defects to easy defects, major defects to minor defects
- Major defects to minor defects, easy defects to hard defects (i.e. Easy Major, Hard Major, Easy Minor, Hard Minor)
- Minor defects to major defects, easy defects to hard defects
- Random ordering
- First in, first out ordering (e.g. oldest defect first)

5.1 Ratio of defects fixed to defects found

Figure 2 (page 5) shows the strategy results for the ratio of defects fixed to defects found. This was calculated as

$$\frac{d(t)}{df(t)}$$

Where $d(t)$ is the number of defects found in week t, and $df(t)$ is the number of defects fixed in week t.

As the graph shows, clearly the poorer strategy to minimise this metric in the beginning is "Hard–Easy, Major–Minor". This is due because by focusing on the most difficult defects with priority in the early weeks where many defects are being found, few are being fixed due to their difficulty, thus massively inflating the metric.

The "Random", "FIFO", "Major-Minor, Easy–Hard" and "Minor–Major, Easy–Hard" strategies sit in the middle of the best and worst strategies in the beginning, then become better than average at around weeks 8–10. This coincides with the massive drop-off of new defects being found around that point.

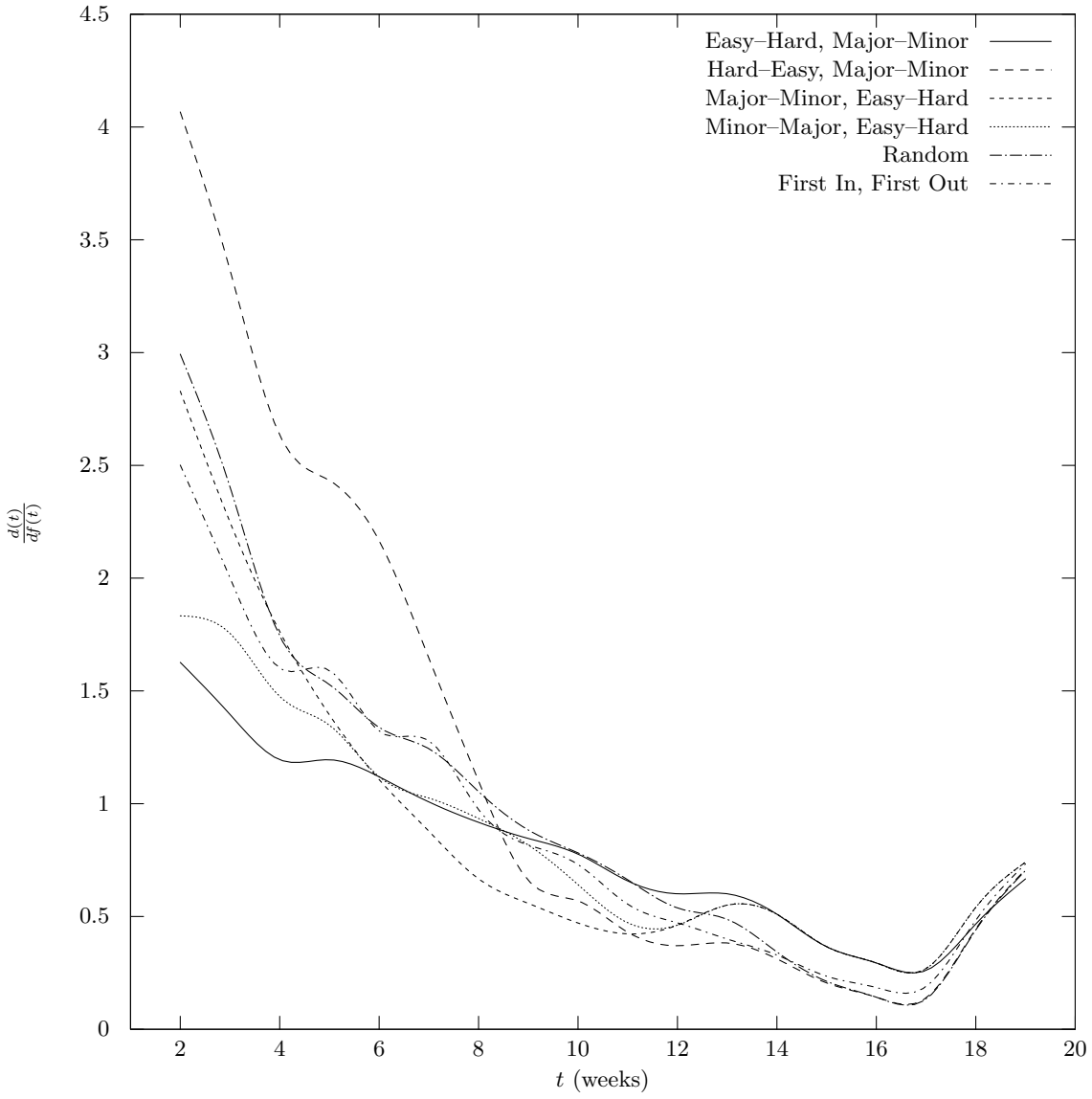

Figure 2: Ratio of defects fixed to defects found; aggregate results

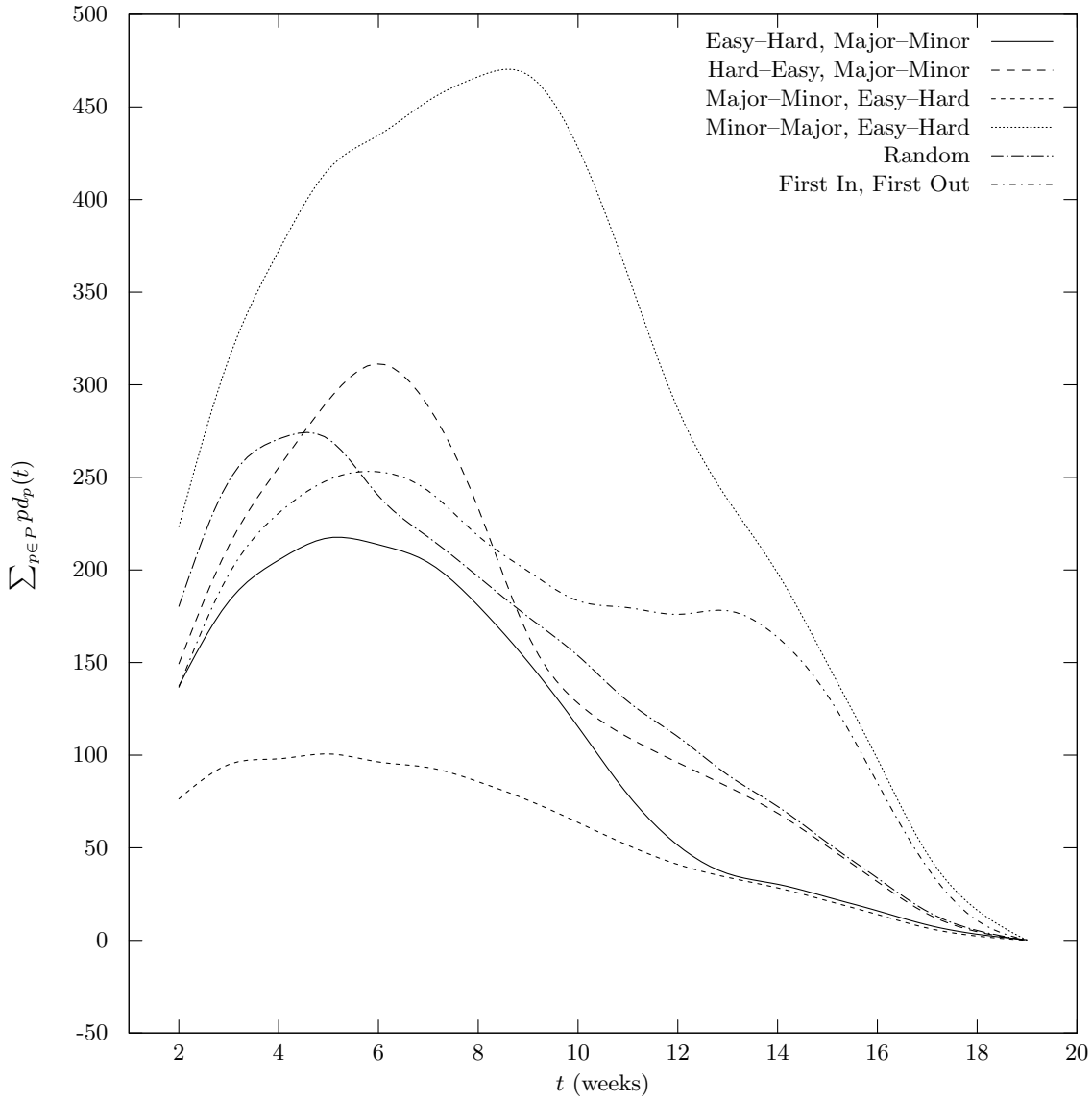

Figure 3: Total importance of found defects still to be fixed; aggregate results

We see that the "Major-Minor, Easy–Hard" and "Minor–Major, Easy–Hard" have very different starting points, with "Minor–Major" starting at around 1.8 vs 3, however they converge around week 10. We can attribute this difference to the higher prevalence of minor defects vs major defects in the data set before that point; with more minor defects being present, and minor defects being more likely to be easy in this dataset, on average more bugs can be fixed per week in the early stages by preferring minor bugs.

The best strategy in the early term is clearly "Easy–Hard, Major–Minor", as this is the strategy that prioritises the easiest to fix defects, leading to a very high number of defects fixed per week.

Here we find that customer satisfaction is at odds with the system reliability, as this metric encourages the software development firm to underfind defects, as well as prioritise strategies for fixing that do not necessarily involve fixing the most important issues first, but rather the easiest ones.

5.2 Total importance of found defects still to be fixed

Figure 3 (page 6) shows the strategy results for the total importance of found defects still to be fixed. This was calculated as

$$\sum_{p \in P} p d_p(t)$$

where P is the set of importances of different priorities and $d_p(t)$ is the number of defects found of priority p in week t.

This graph is more readily understood that the previous one, as the performance of each element on this graph can be explained as an optimisation of the most major defects that can be solved most quickly. The clear winner is "Major–Minor, Easy–Hard", as it ranks defects from the most important easiest to the least important hardest. A runner up is the similar "Easy–Hard, Major–Minor", however it fails to be most optimal by prioritising Hard Major over Easy Minor.

The worst result is the that of "Minor–Major, Easy–Hard", which effectively places major issues at the lowest priority of the set of strategies.

Customer satisfaction and this metric are closely related, as this metric closely maps with the resolution of the most important defects first, which will ultimately lead to less important bugs remaining in the final product. Similarly, system reliability with this approach will be maximised, as those issues which are most important are those which impact the system reliability the most.

5.3 Average queue time of major defects still to be fixed

Figure 4 (page 8) shows the strategy results for the average queue time of major defects still to be fixed. This is calculated as

$$\frac{df(t) + dc(t)}{df(t)}$$

where $df(t)$ is the number of defects fixed at week t and $dc(t)$ is the total number of defects remaining unfixed from week 1 to week t.

The worst strategy "Minor–Major, Easy–Hard", which was cut off in figure 4 due to its sheer size (it can be seen separately in figure 5 on page 9). It peaks of an average queue time of around 70 weeks, as it fully deprioritises major defects, which are the entire basis of this metric.

The best strategy is clearly "Major–Minor, Easy–Hard", as it prioritises the quickly fixable easy defects, which lead to a very low metric for the entire period.

Customer satisfaction and system reliability in this metric are closely related, as a strategy the customer is happy with would also promote system reliability. However, as the graphs show, all strategies converge around 1 week, demonstrating that all approaches would lead to major issues being fixed, but not necessarily with priority.

5.4 Estimate of number of unfixed defects still in software

Figure 6 (page 10) shows the strategy results for the estimate of number of unfixed defects still in software, which were the same for every strategy explored. This result is calculated by performing a regression on $\{d(1), ..., d(t)\}$ for week t, to calculate values b and c for the exponential regression. Then the results of the integral

$$\int_t^\infty b e^{cx} \, \mathrm{d}x$$

are calculated for each $t > 1$, then graphed. Because all previously mentioned strategies used the same allocation of $p_f = 1$, $p_r = 2$, this metric does not change within test strategies. See variants that differ from this and associated discussion in section 6.

Customer satisfaction and system reliability are not discussed with this metric as there is no difference between scenarios.

6 Employee Reallocations

To investigate how different metrics change with specifically tester/repairer variations, 3 variants of the "Major–Minor, Easy–Hard" were devised;

- **Scenario A (Baseline)**: Major–Minor Easy–Hard priority

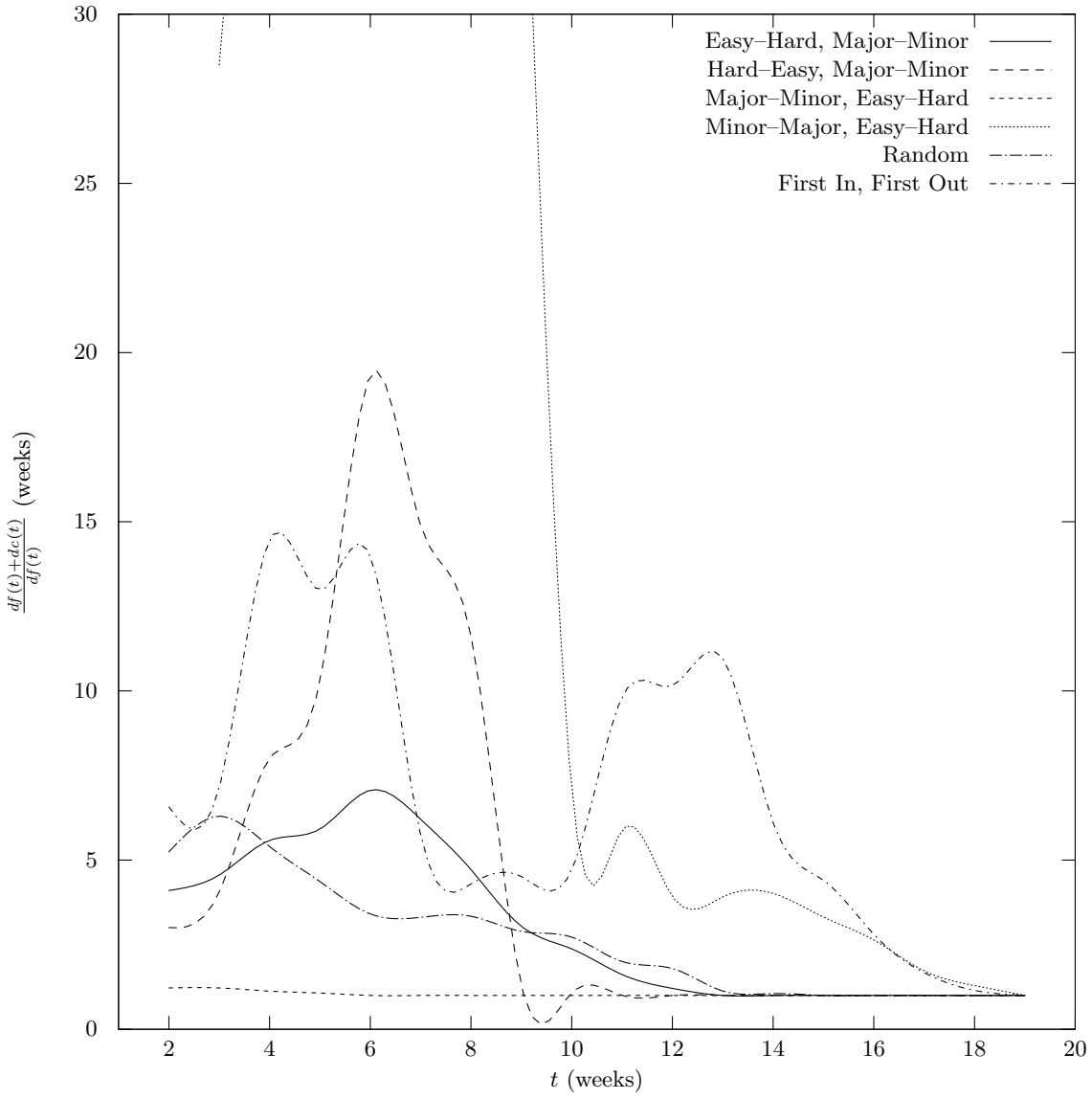

Figure 4: Average queue time of major defects still to be fixed; aggregate results

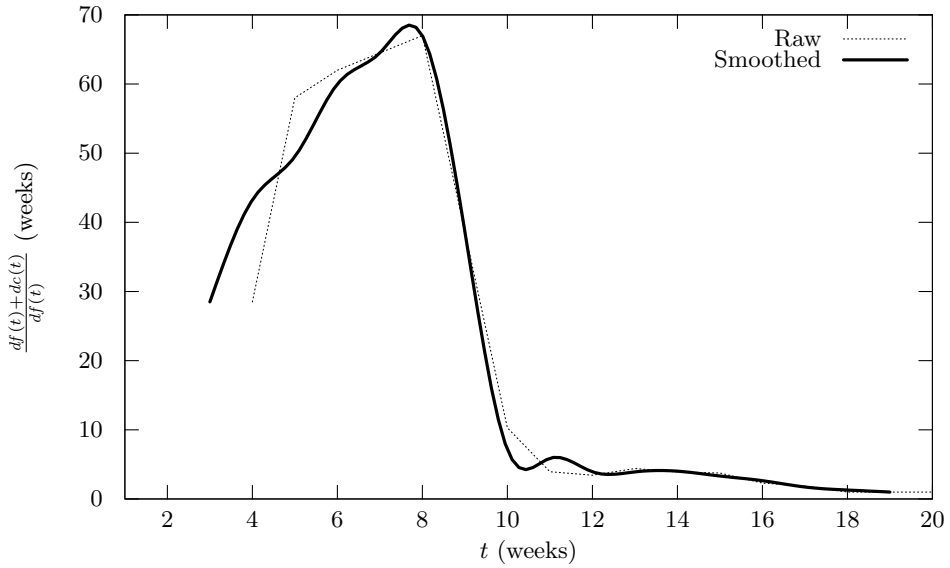

Figure 5: Average queue time of major defects still to be fixed; Minor–Major, Easy–Hard

- **Scenario B**: $p_f = 2$, $p_r = 1$ for $1 \leq t \leq 4$, otherwise $p_f = 1$, $p_r = 2$
- **Scenario C**: $p_f = 0$, $p_r = 3$ if $t \bmod 3 = 0$, otherwise $p_f = 1$, $p_r = 2$
- **Scenario D**: $p_f = 2$, $p_r = 1$ for $1 \leq t \leq 4$, $p_f = 0$, $p_r = 3$ for $17 \leq t \leq 20$, otherwise $p_f = 1$, $p_r = 2$

Some data points with each of this scenarios could not be graphed due to limitations of exponential regression functions or division by zero errors. In the former case, affected datapoints have been excluded when at the end or beginning of the dataset, otherwise these points have been replaced by an average of adjacent points. In the latter case, affected formulas have been redefined as equation 1 for the case of Defects Fixed:Defects Found and equation 2 for average queuetime.

$$\frac{d(t)+1}{df(t)+1} \tag{1}$$

$$\frac{df(t)+dc(t)+1}{df(t)+1} \tag{2}$$

6.1 Testers to Repairing

We can see from data displayed in figures 7–10 that moving testers to repair roles has various effects for the different metrics. The primary scenario that represents this move is Scenario D, which moves testers to repairs in the last four weeks.

We see no difference in the ratio of defects found to defects fixed (figure 7, page 11), as by that point in time, most defects have been located and handled. Similarly with the average queue time of major defects (figure 9, page 12). We see a slight difference in the total importance of found defects (figure 8, page 11) as compared to Scenario B in the final weeks, most likely due to a slightly faster repair cycle. We also see a spike in the estimate of number of unfixed defects (figure 10, page 12) at the same point where testers move to repairs; this is most likely due to a reevaluation of the exponential regression that is less favourable.

6.2 Repairers to Testing

We can see from data displayed in figures 7–10 that moving repairers to testing roles also has various effects for the different metrics. The primary scenario that represents this move is Scenario B, which moves repairers to testing for the first four weeks.

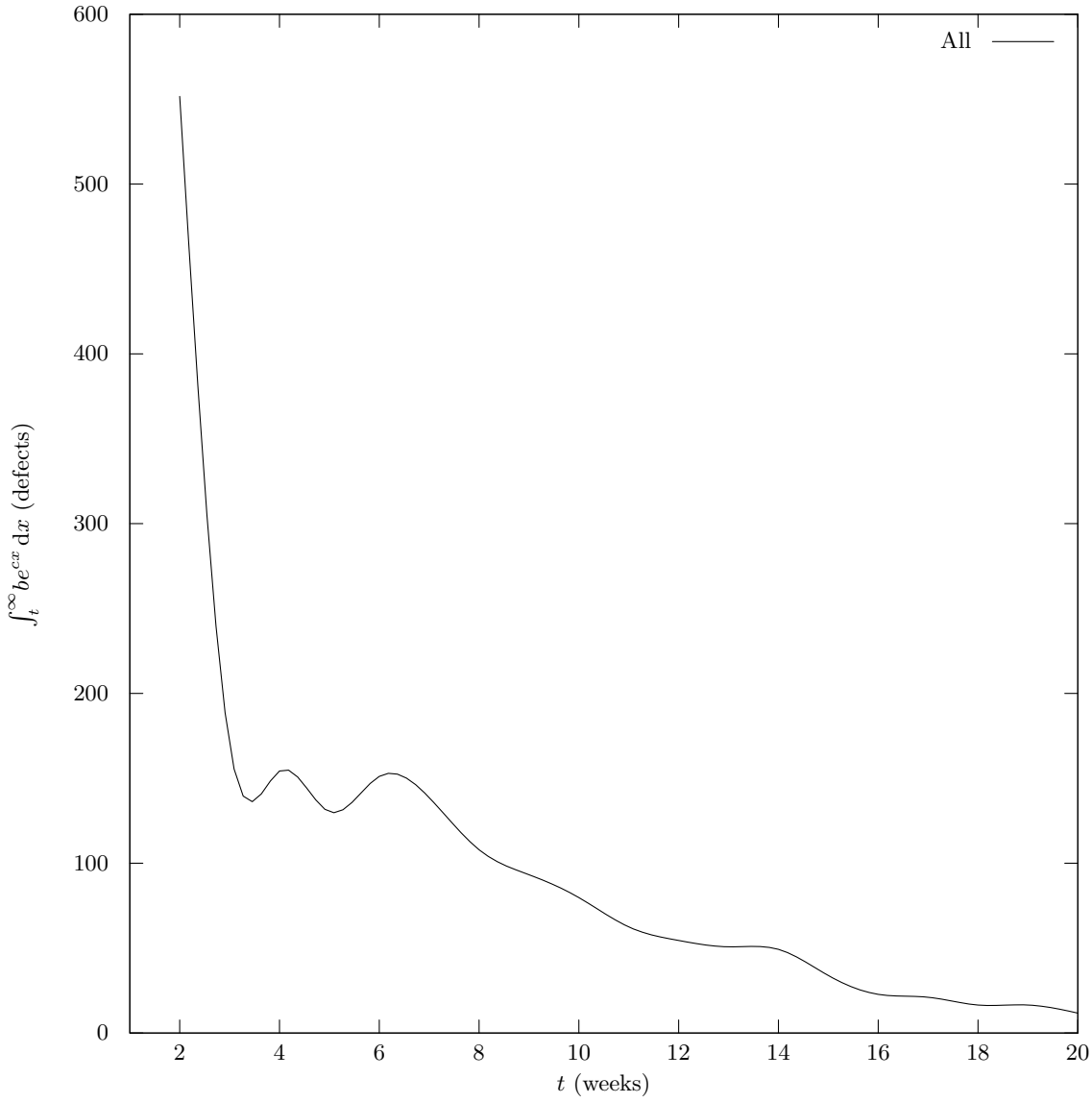

Figure 6: Estimate of number of unfixed defects still in software; aggregate results

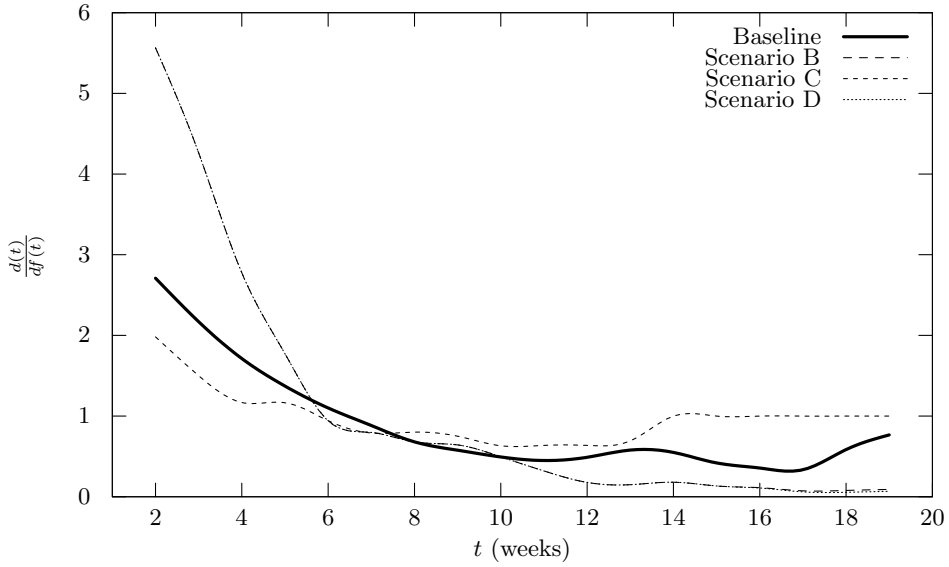

Figure 7: Ratio of defects fixed to defects found; comparison

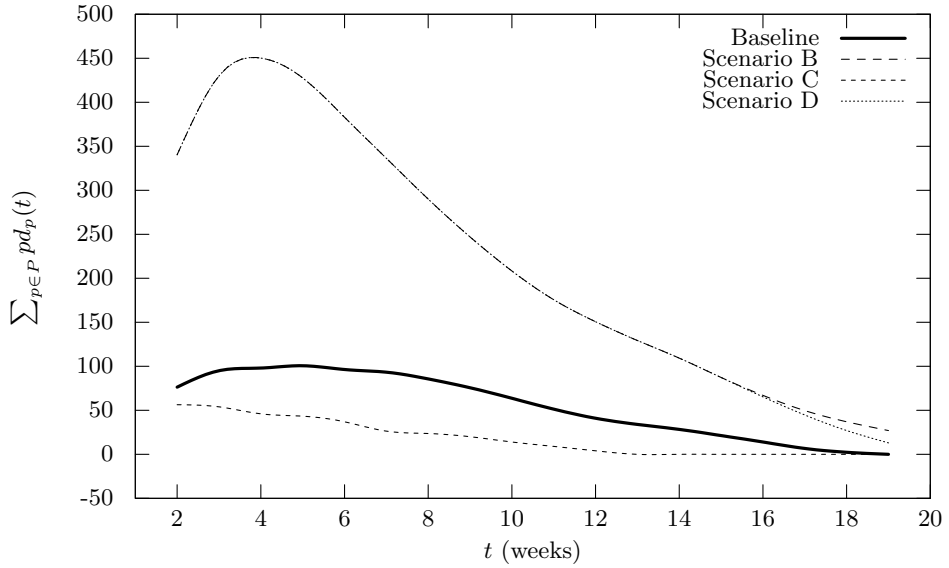

Figure 8: Total importance of found defects still to be fixed; comparison

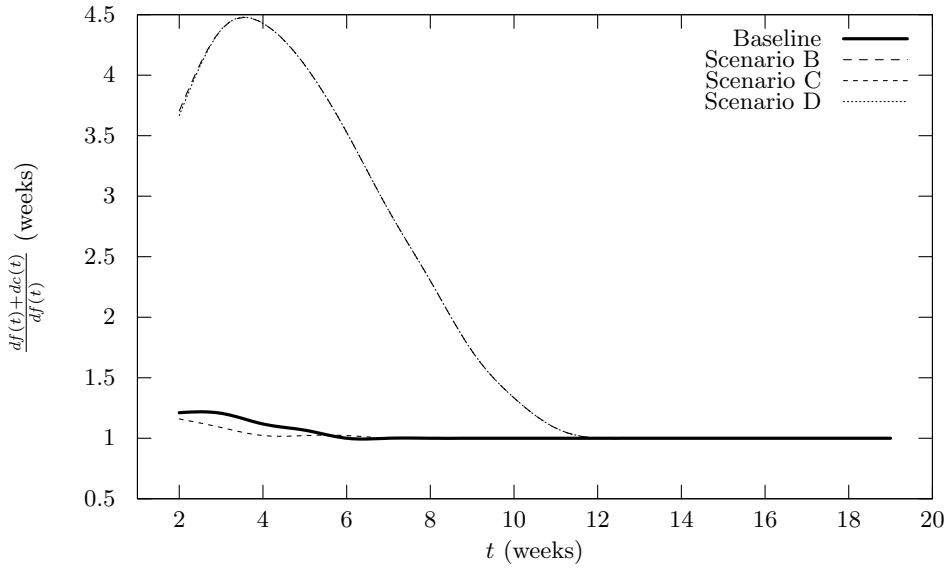

Figure 9: Average queue time of major defects still to be fixed; comparison

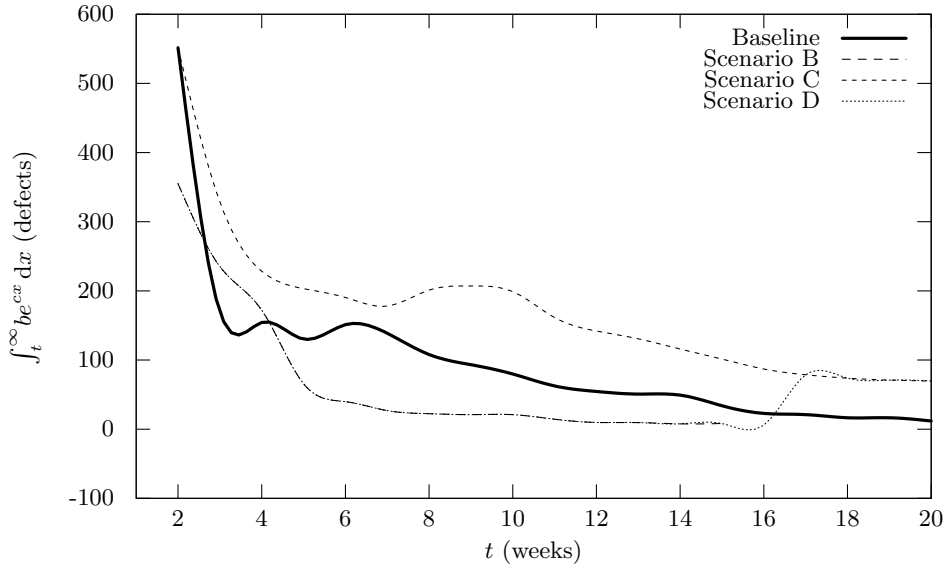

Figure 10: Estimate of number of unfixed defects still in software; comparison

We see very poor performance in the ratio of defects found to defects fixed due to finding more defects much faster with this allocation of testers. We similarly see poor performance in the total importance of found defects and the average queue time due to this. However, there is a noticeably better performance in the estimate of number of unfixed defects, as the rate of finding defects leads to a regression curve that predicts a much faster downward slope.

6.3 Optimum Staff Allocation

To minimise the time major defects went unfound and unfixed, these graphs indicate a dynamic allocation of staff would be in order. An optimum allocation of staff would most likely consist of staff moving between "sprints" of finding defects, then fixing them. These sprints would need to be dynamically calculated based on the estimated number of unfound major defects and the amount of staff hours required to fix the current backlog of major defects. There would be an optimisation of many staff being allocated to "finding" enough defects that n staff can fix them in exactly t weeks, then those staff moving to repairs to resolve those defects.

This however would most likely lead to highly variable metrics in many cases, so it would be important to choose the correct metrics to ensure customer satisfaction with this approach.

7 Conclusion

In conclusion, we can see that while most strategies investigated in this document lead to most, if not all, found defects being resolved before the 20 week test and repair period had elapsed, there was a clear variety in how those strategies showed in the different metrics chosen to be investigated in this report.

While metrics are objective measures of something, they are not necessary objective measures of the "heath" of a software project. Metrics, it has been shown, can be manipulated by different strategies to make them more or less favourable to the customer and to system reliability. Interestingly, this manipulation in most cases has little impact on most defects being resolved before the project's test and repair deadline, but rather the order in which they are resolved.

From a software developer standpoint, clearly the metrics used should inform the testing strategy used, or vice-verse, as customer satisfaction will be tied heavily to those metrics indicating a "healthy" software engineering project.

From a customer standpoint, to actually measure the health of the project, it would be naive to choose merely one metric, as it would not provide the requisite data definition to get a true handle on the heath of the software project. It would be important to consider that there are many healthy project scenarios that do not necessarily have metrics that constantly trend downwards and to use a mixture of metrics alongside other qualitative and quantitative data.

Appendices

A Results

Note: Different graphs are not of comparative scale.

A.1 Ratio of defects fixed to defects found

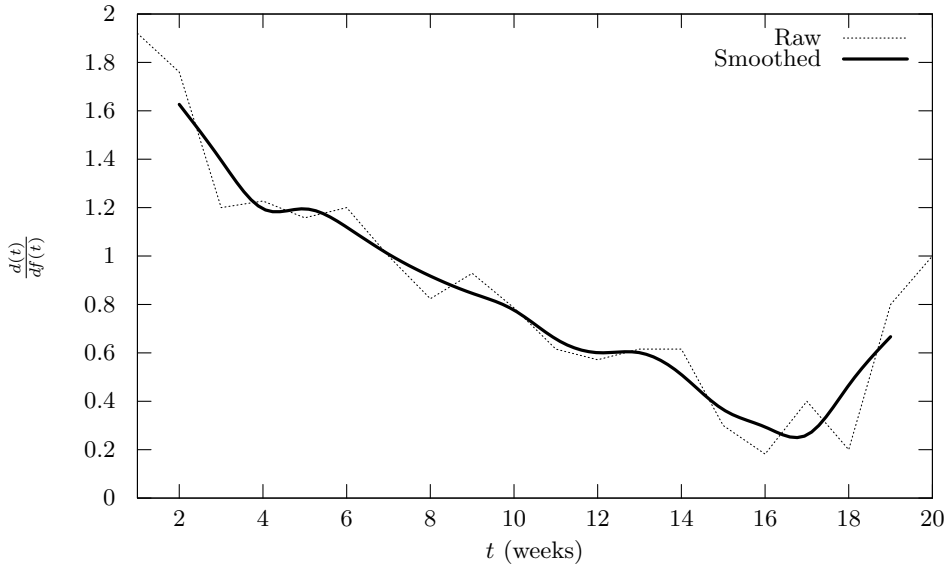

Figure 11: Easy–Hard, Major–Minor

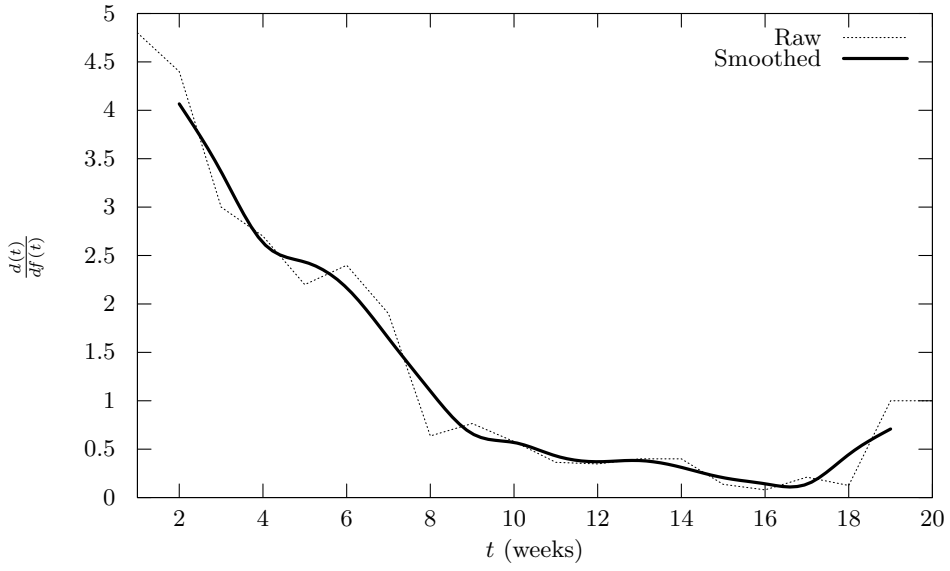

Figure 12: Hard–Easy, Major–Minor

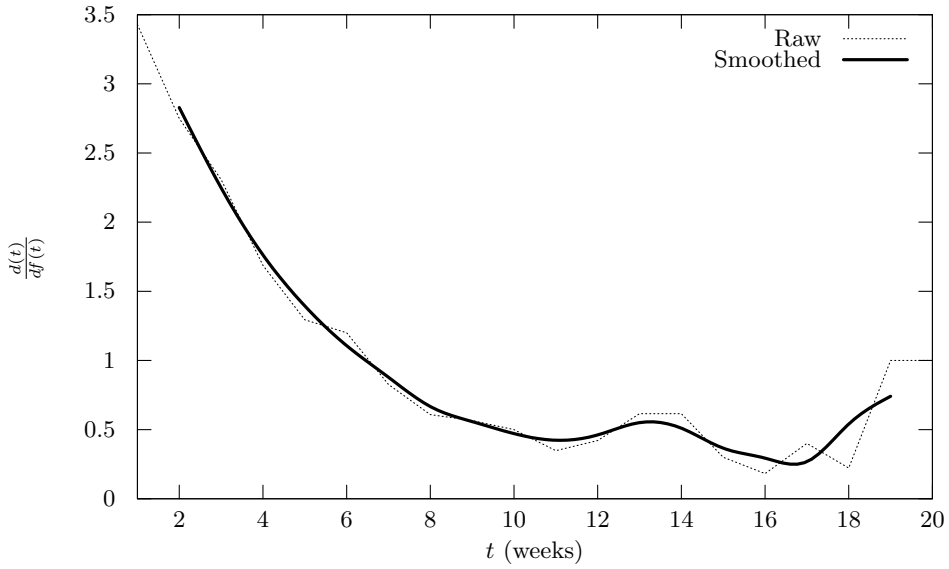

Figure 13: Major–Minor, Easy–Hard

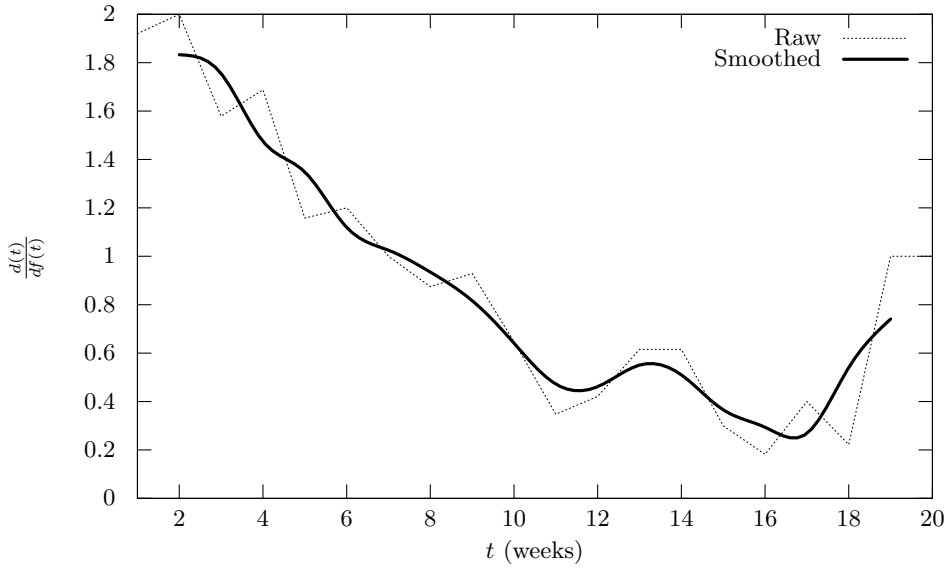

Figure 14: Minor–Major, Easy–Hard

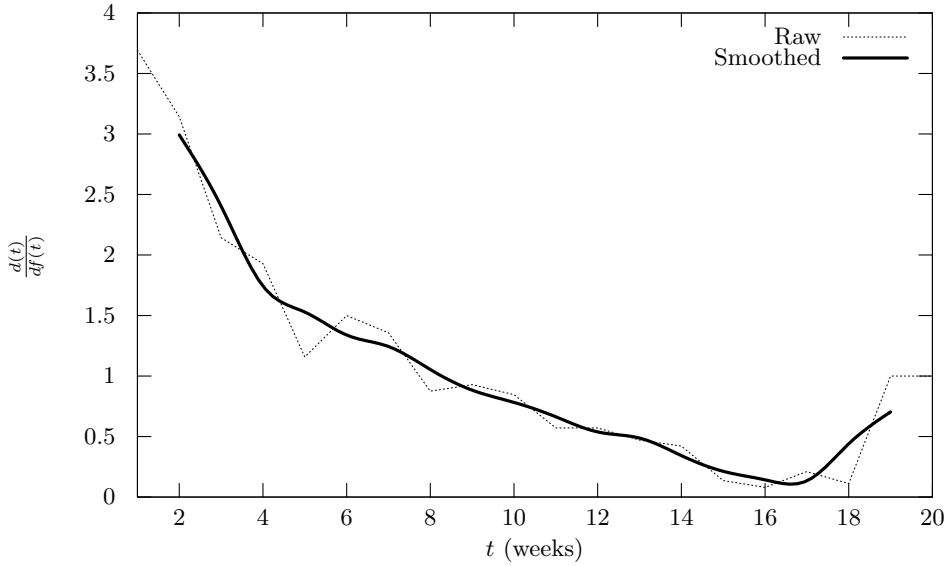

Figure 15: Random

A.2 Total importance of found defects still to be fixed

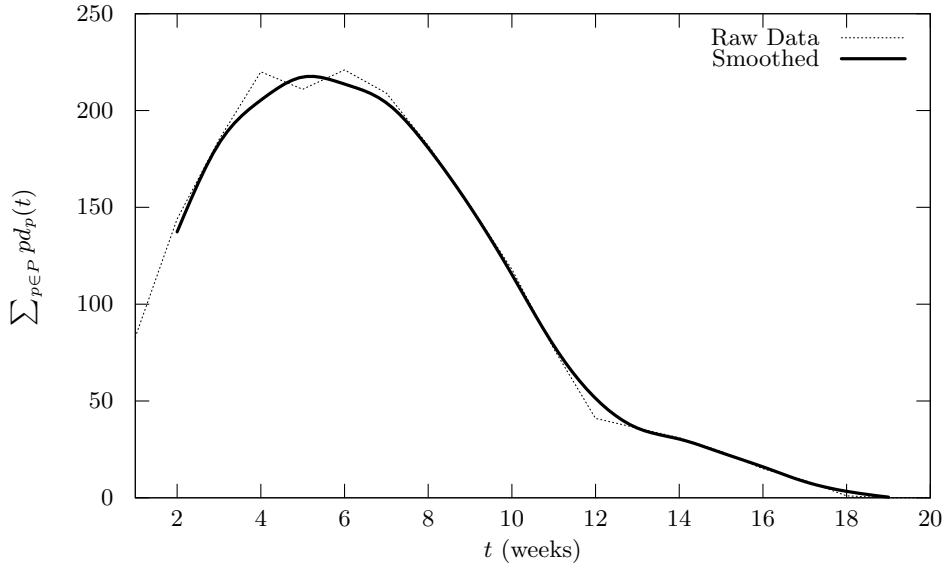

Figure 16: Easy–Hard, Major–Minor

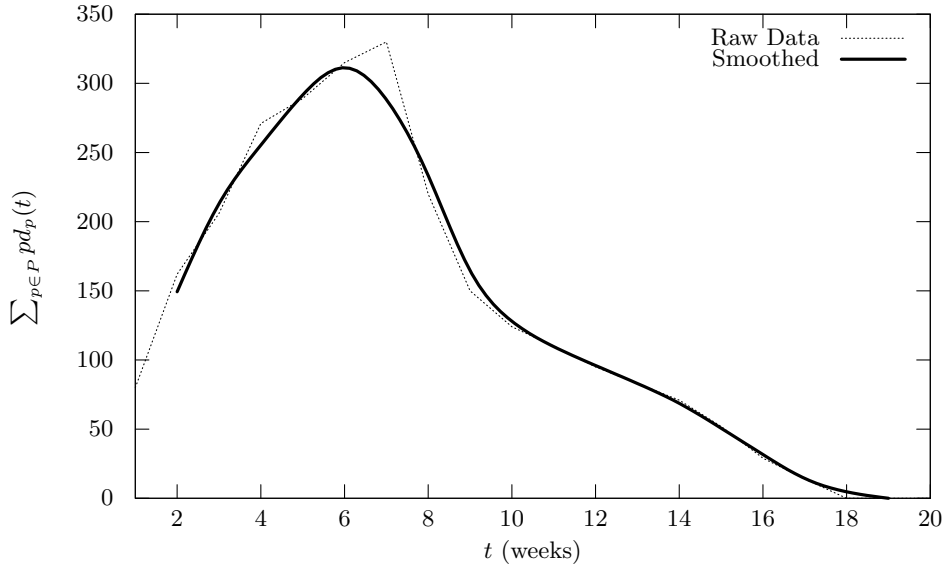

Figure 17: Hard–Easy, Major–Minor

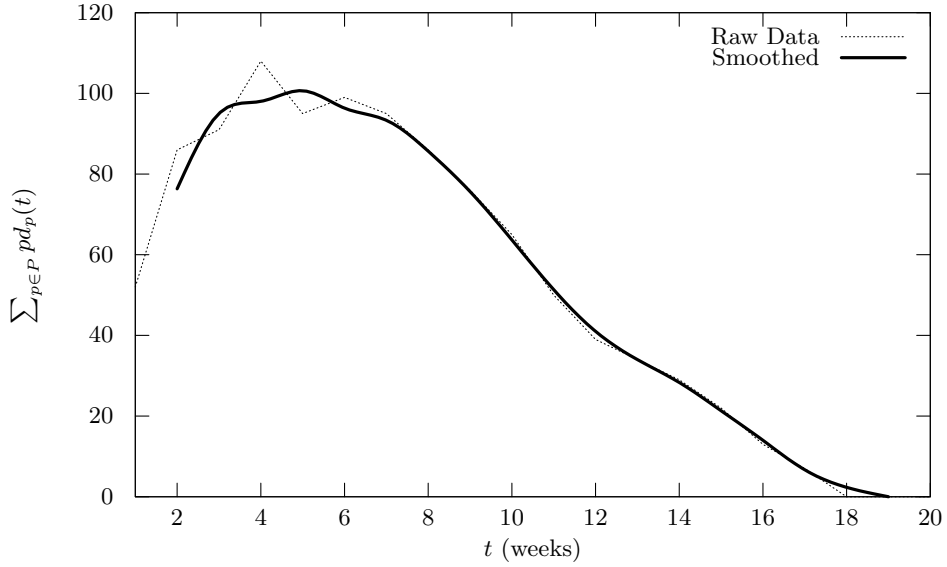

Figure 18: Major–Minor, Easy–Hard

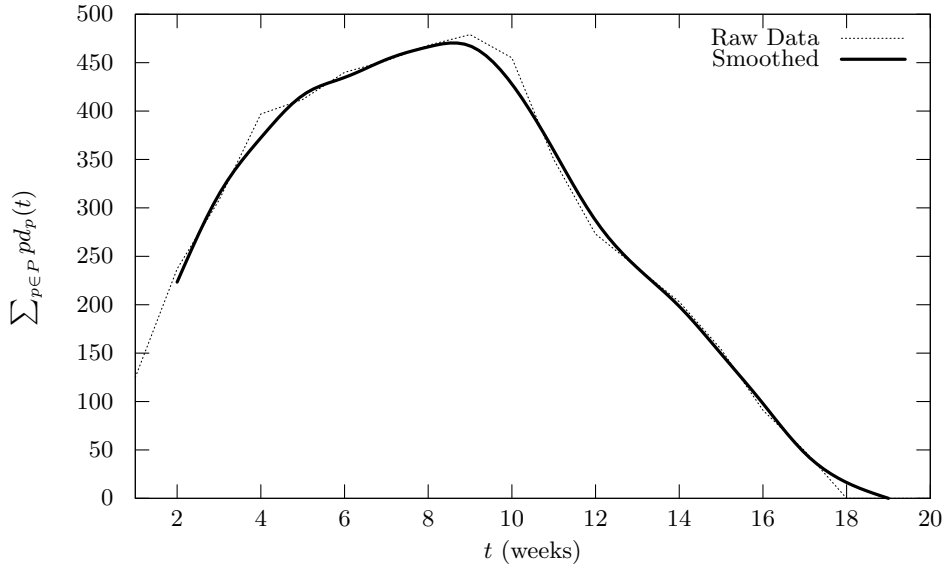

Figure 19: Minor–Major, Easy–Hard

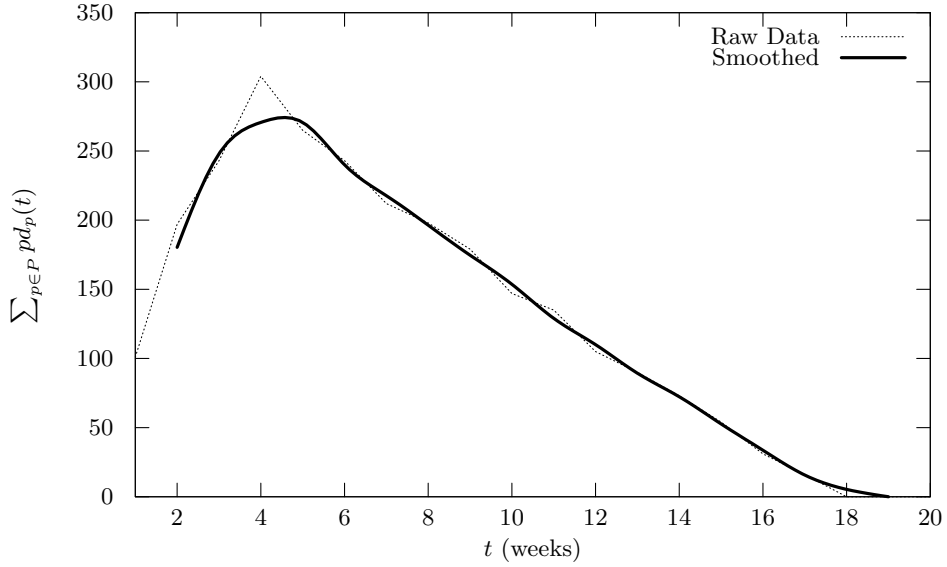

Figure 20: Random

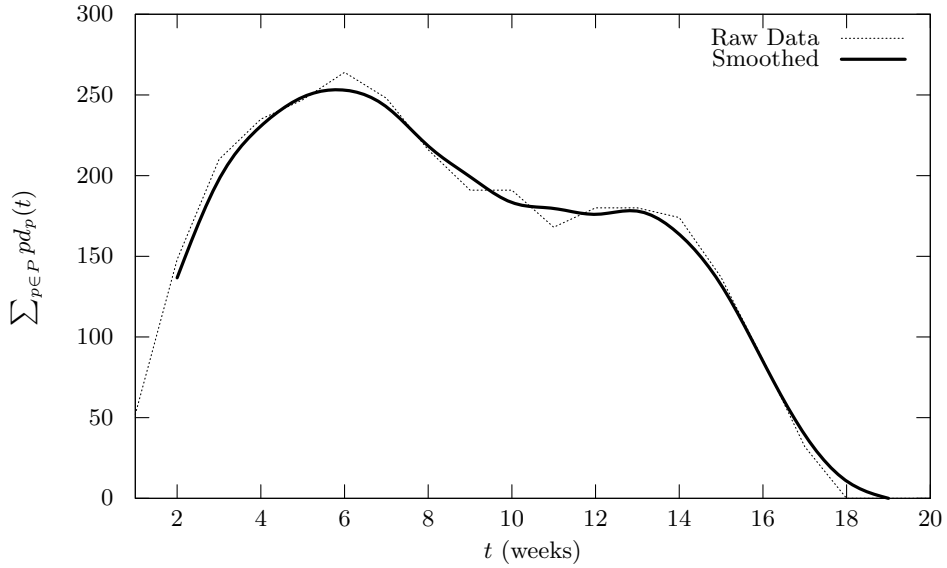

Figure 21: First In, First Out

A.3 Average queue time of major defects still to be fixed

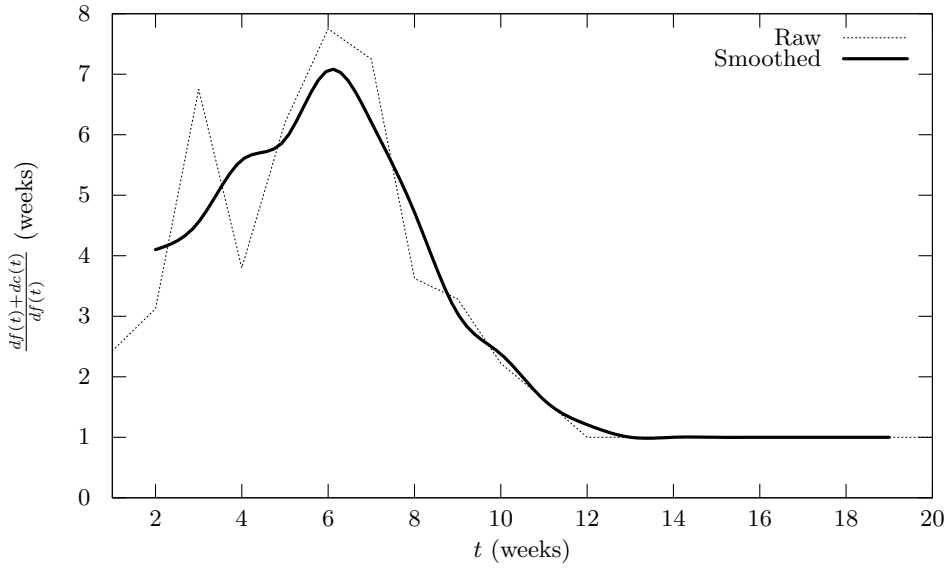

Figure 22: Easy–Hard, Major–Minor

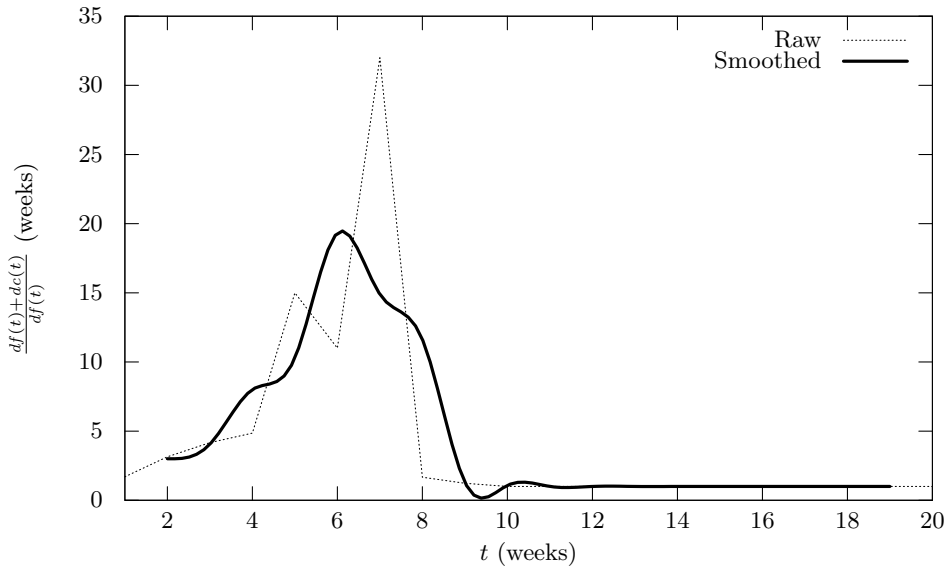

Figure 23: Hard–Easy, Major–Minor

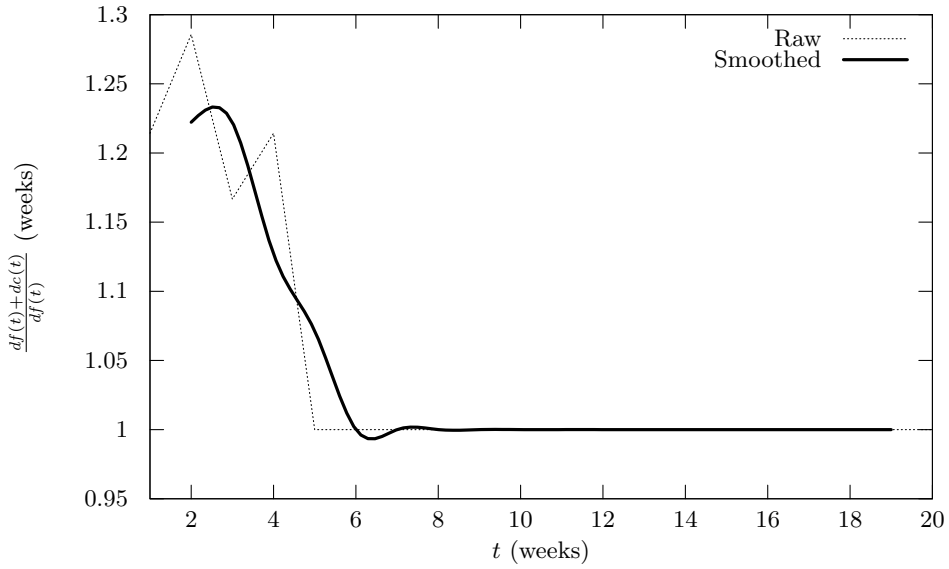

Figure 24: Major–Minor, Easy–Hard

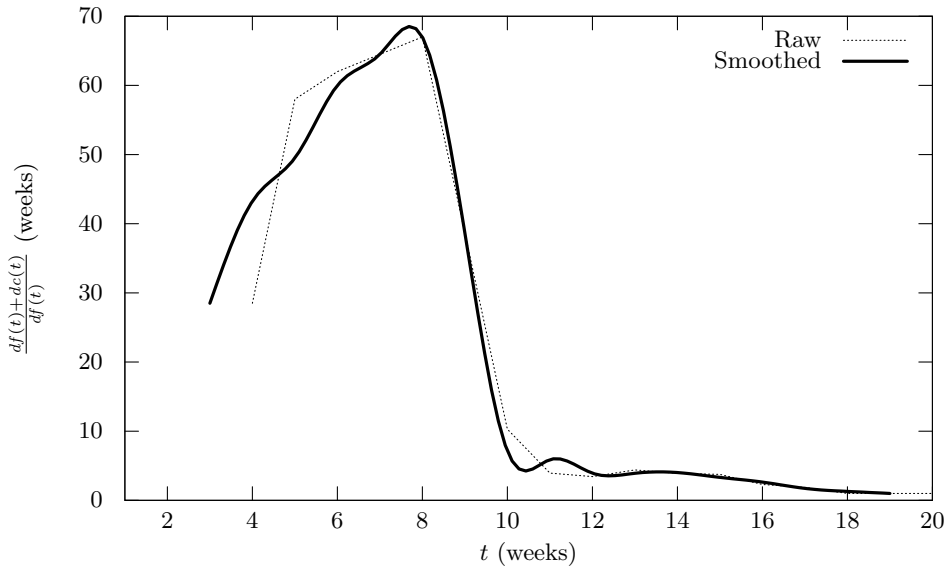

Figure 25: Minor–Major, Easy–Hard

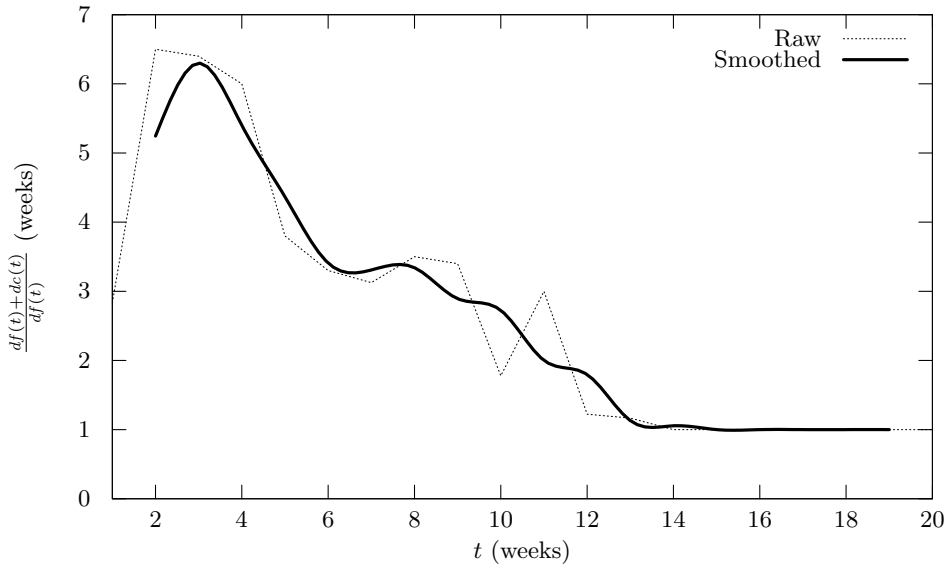

Figure 26: Random

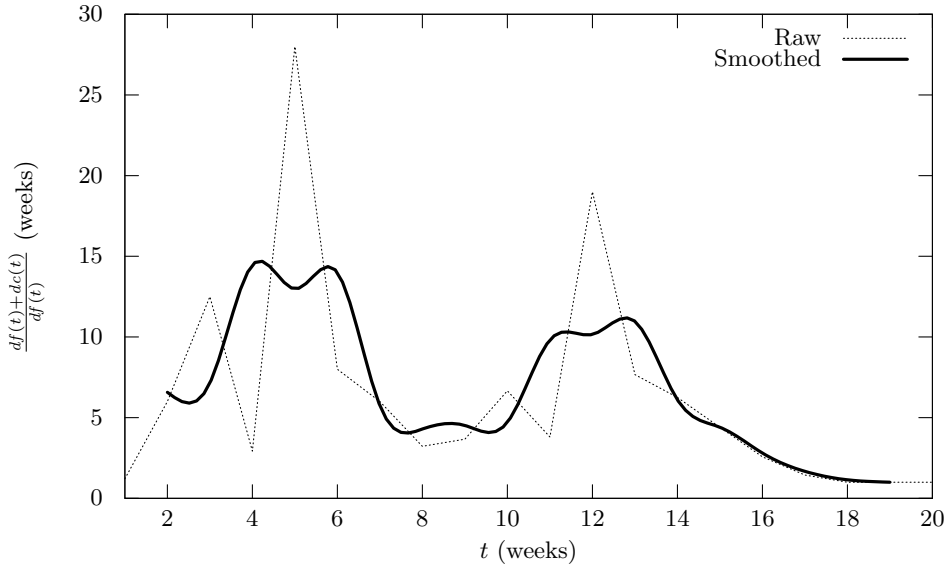

Figure 27: First In, First Out

A.4 Estimate of number of unfixed defects still in software

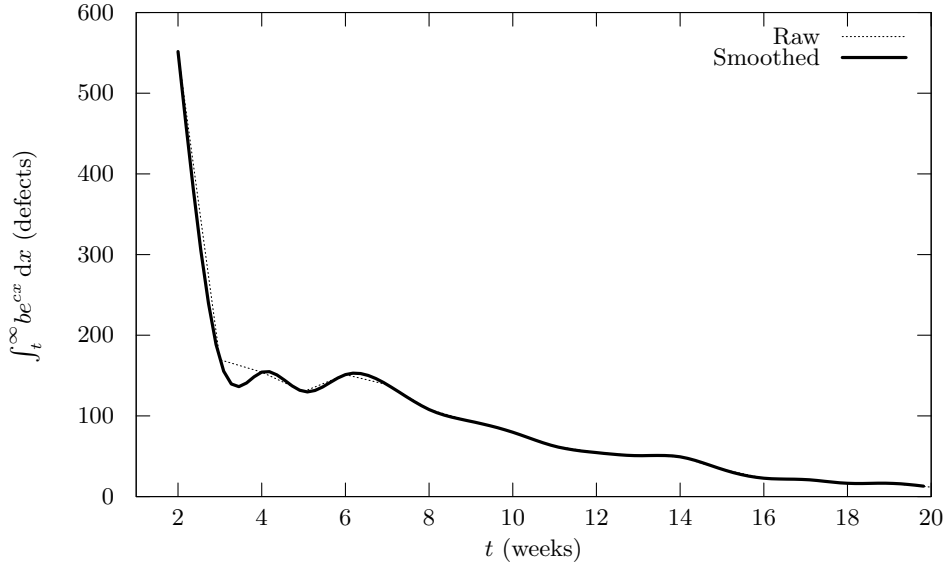

Figure 28: All curves

B Variants

Note: Different graphs are not of comparative scale.

B.1 Ratio of defects fixed to defects found

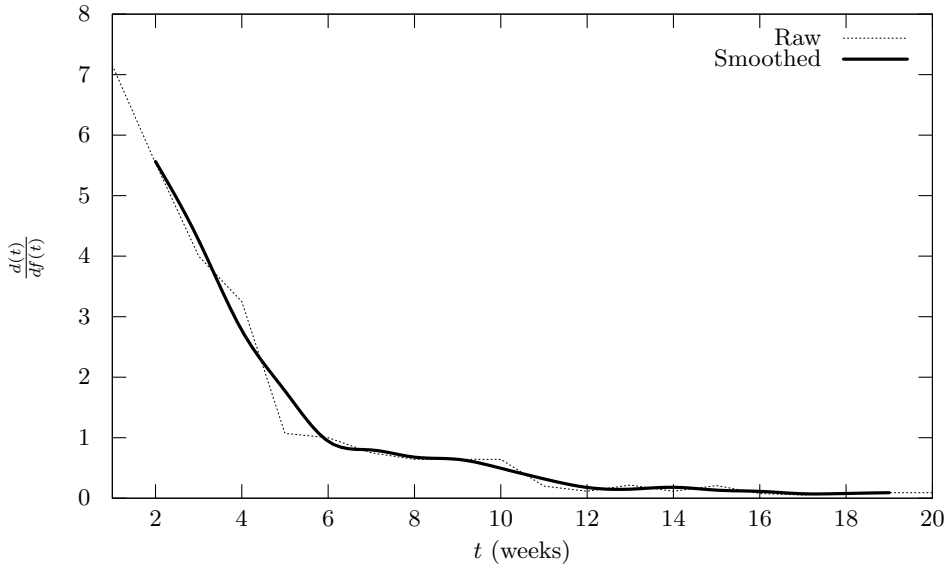

Figure 29: Scenario B Result

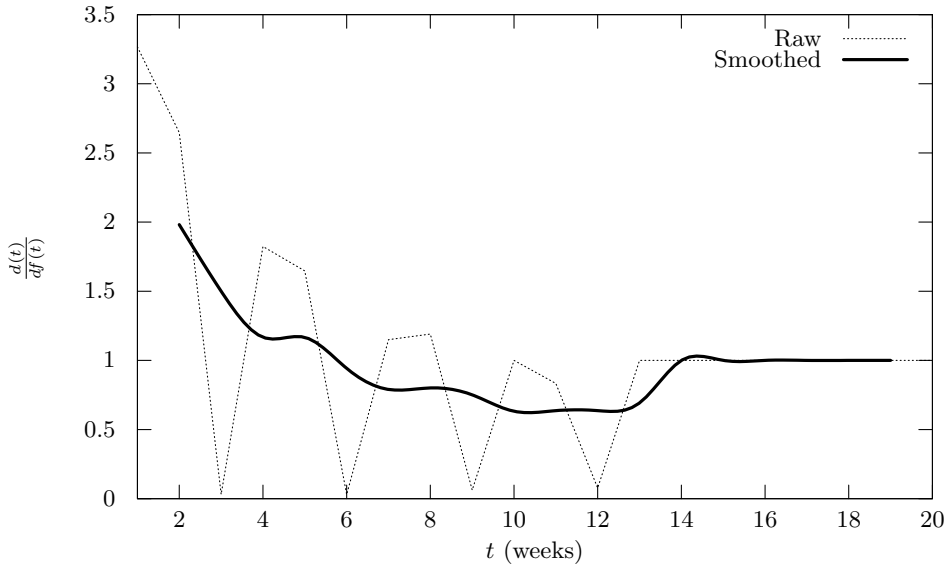

Figure 30: Scenario C Result

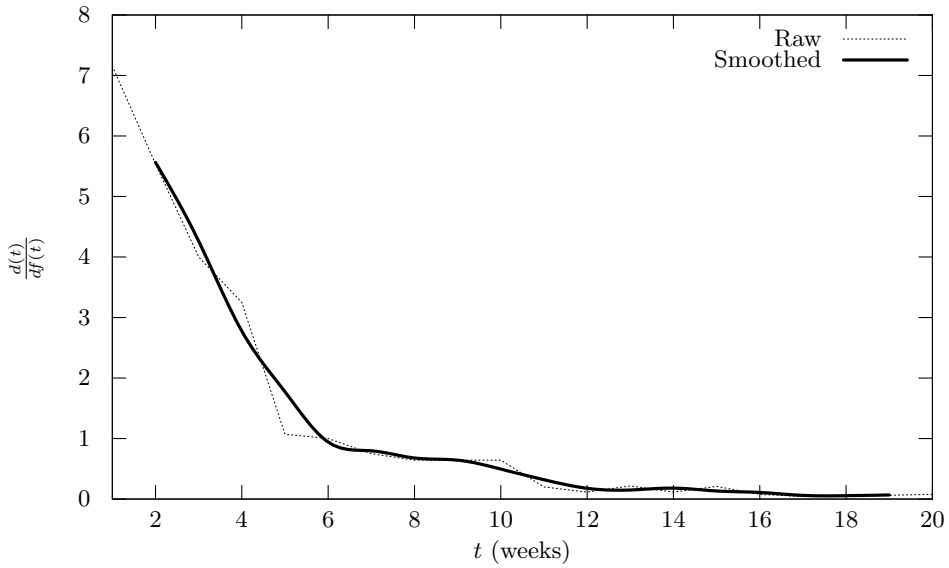

Figure 31: Scenario D Result

B.2 Total importance of found defects still to be fixed

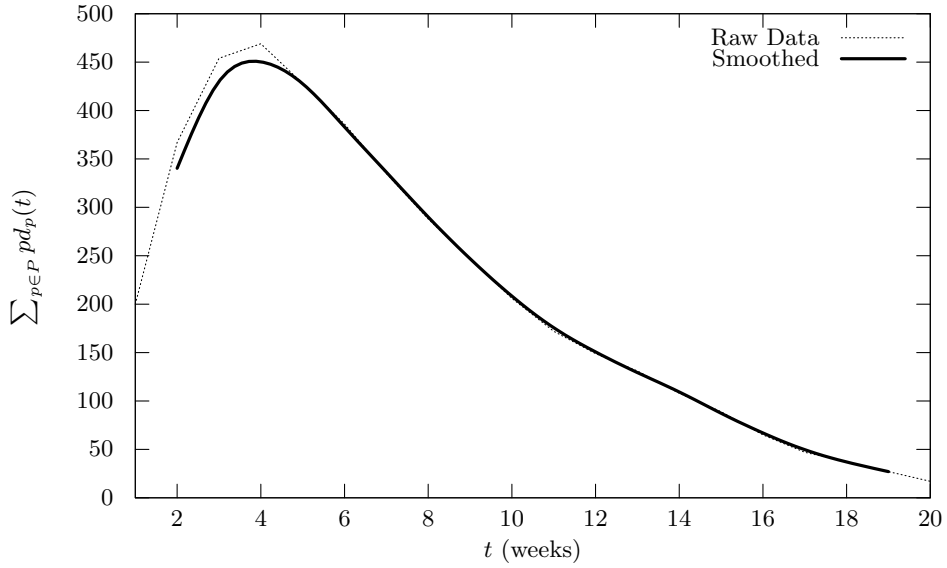

Figure 32: Scenario B Result

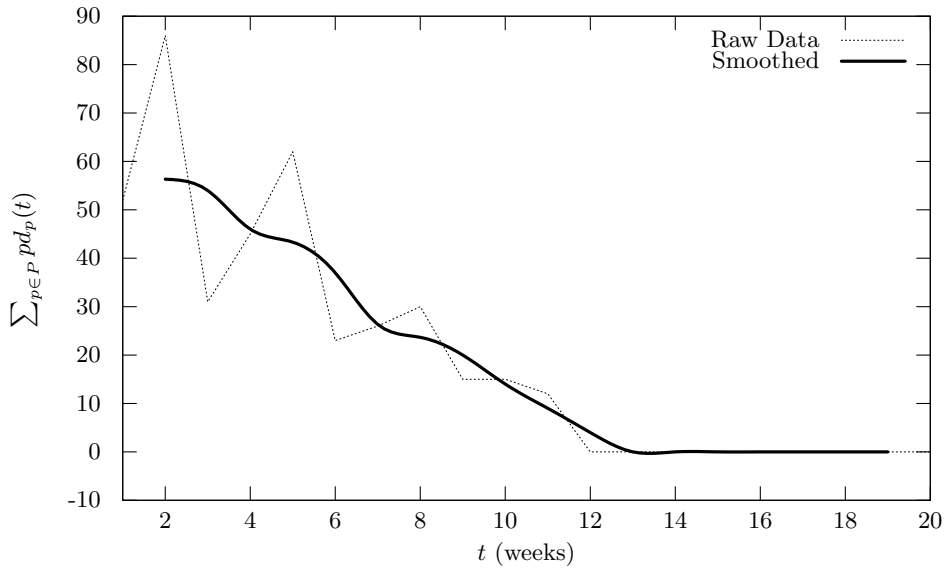

Figure 33: Scenario C Result

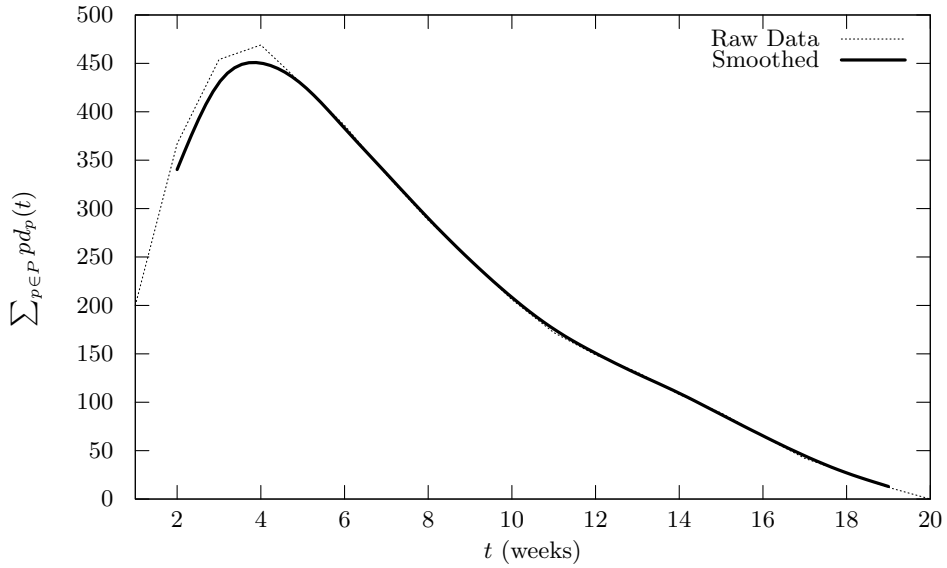

Figure 34: Scenario D Result

B.3 Average queue time of major defects still to be fixed

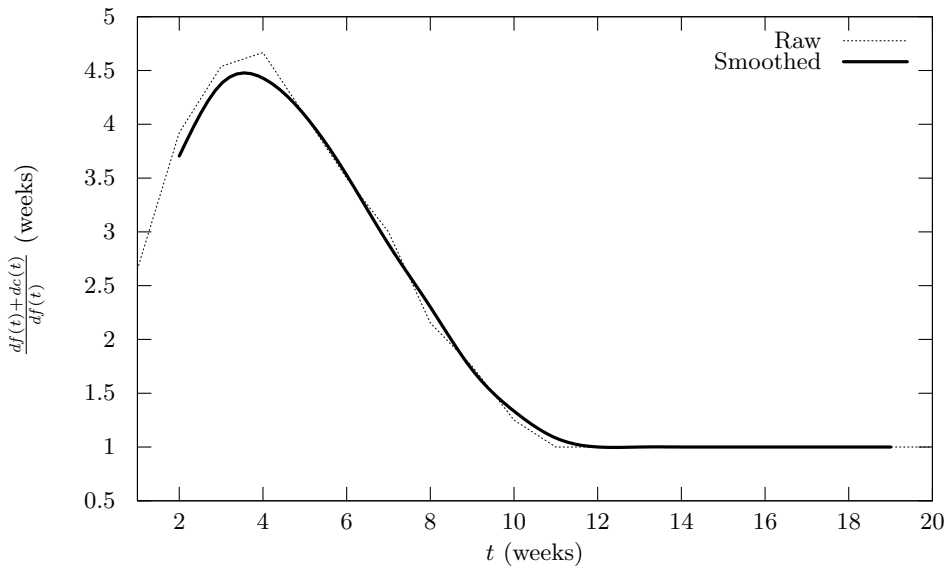

Figure 35: Scenario B Result

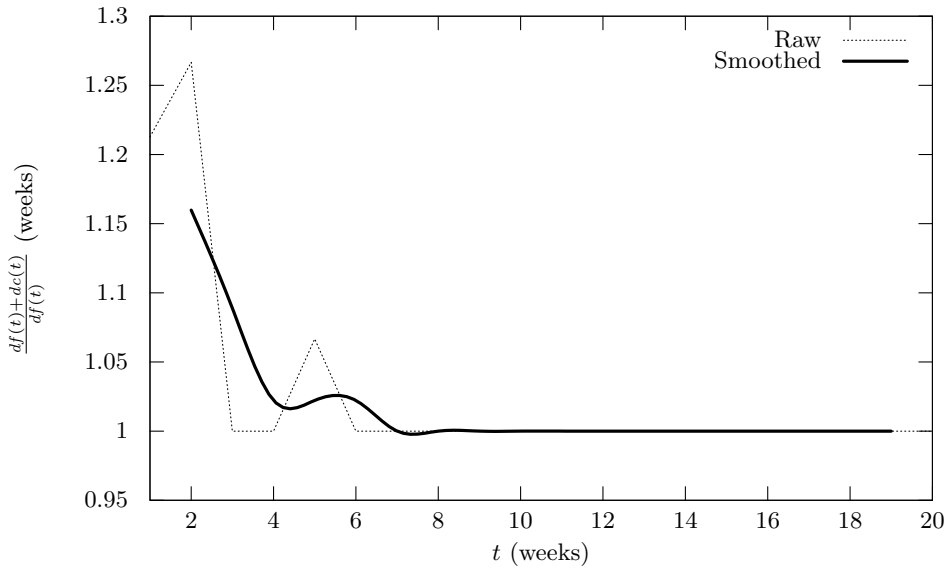

Figure 36: Scenario C Result

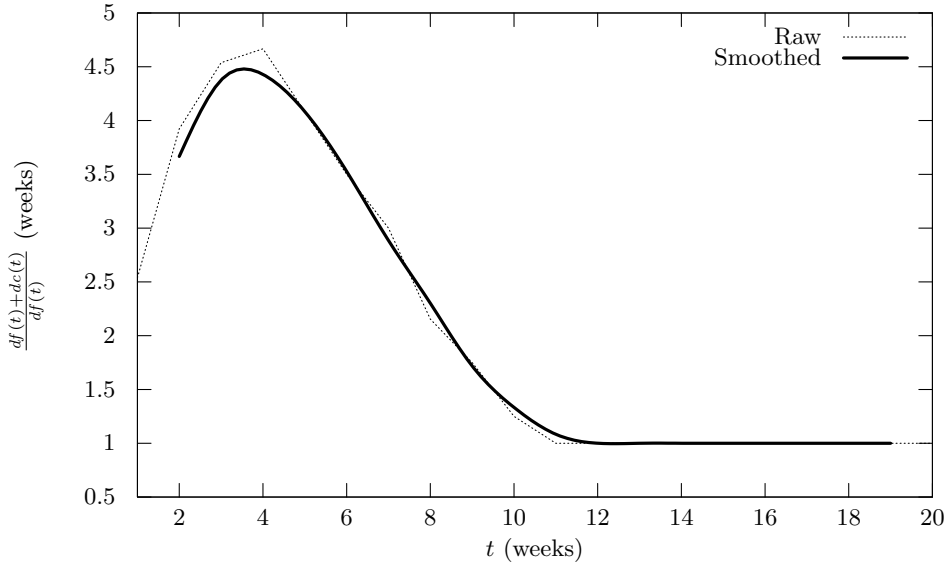

Figure 37: Scenario D Result

B.4 Estimate of number of unfixed defects still in software

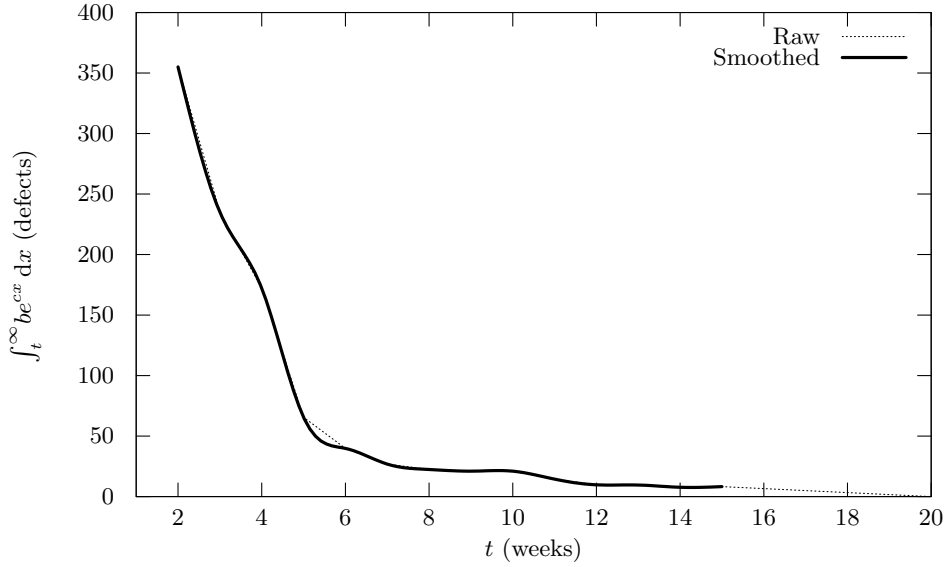

Figure 38: Scenario B Result

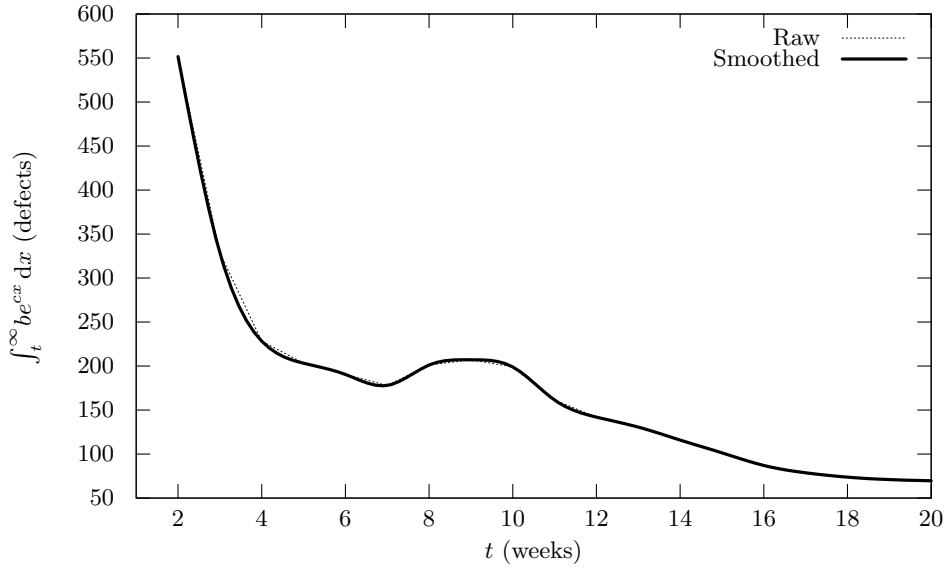

Figure 39: Scenario C Result

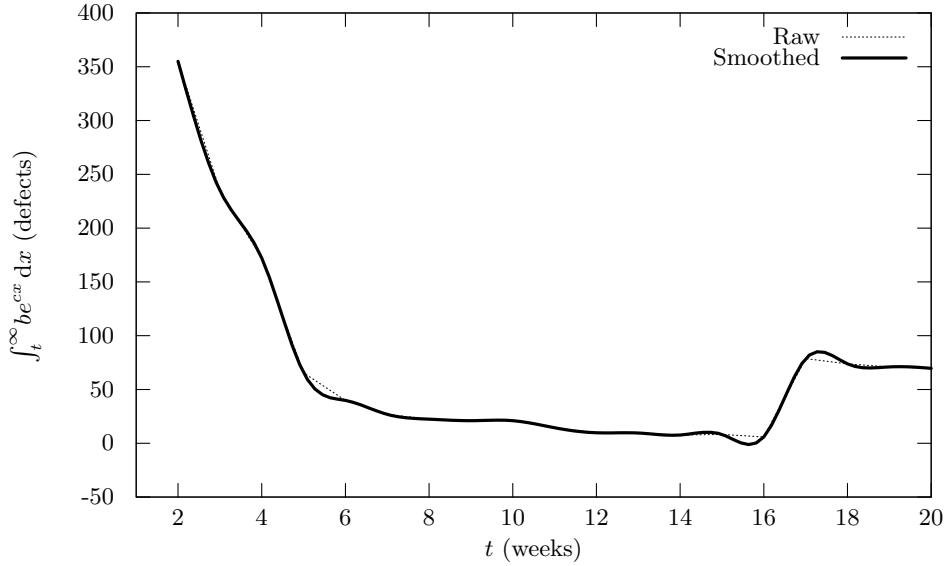

Figure 40: Scenario D Result