CITS5502: Assignment 3 – Modelling the Production of Code

Ash Tyndall, 20915779

October 4, 2014

Contents

1 Introduction

If one hopes to predict the time a software development project will take, it is important to be able to understand how to predict the behaviour of code production, or at the very least, know the pitfalls of such predictions. A key part of any such prediction is an investigation of effort and learning. It is readily understood that most processes involving people become more efficient as those people acquire more knowledge about the process; the same could be said of software development. A variety of models have been proposed over the years attempting to model this so called "learning curve", such that with sufficient prior data on similar projects, one could make predictions as to how much more efficient a new project with some similarities could be.

There are two different parts to this learning curve; knowledge acquisition and knowledge encoding. Knowledge acquisition deals with general understanding of the problem domain; in the example problems proposed in this unit, they dealt with spherical geometry and various measurements thereof. Knowing generally the equations and algorithms relevant to this mathematical field would be an example of knowledge acquisition.

Knowledge encoding deals with the understanding a person has of the resources they posses to convert the knowledge they have acquired into a software solution that executes a specific task within the problem domain. Knowing the APIs of the given programming language, how to debug it effectively, how to take inputs and outputs as well as the specific syntax to encode the acquired mathematical formula all form part of this knowledge encoding understanding.

Both of these activities are explored within this report within the context of a large data set of solely total programming timings provided from prior CITS8220 course, as well as a smaller data set of personal timings separated into more discrete phases. Using these data sets, several different models of "learning" will be explored in terms of how well they appear to operate in real world scenarios, and the parameters these models provide will be analysed to draw conclusions about the nature and effect of prior learning on knowledge acquisition, knowledge encoding, effort estimation, solution consistence and limiting factors. The effectiveness and evolution of my personal software process will also be discussed in this context.

2 Program Development

2.1 Measurement Process & Data

The personal data collected for this assignment was collected via a paper worksheet, with the first program being assigned $t = 0$, the second $t = 1$, etc. This sheet asked the user to note down the "phase" of their personal process they were currently in, as well as the minutes they spent in that phase. This paper was next to each participant as they researched and wrote their solution. This type of data recording was adequate, however, there were inaccuracies introduced into the collection by the phases not being immediately recorded on the sheet when they were exited. Participants would also become distracted by their research or development, and switch phases quickly several times, and these phases were only recorded some minutes after they had occurred. This has a potential to introduce inaccuracies.

Also, in some cases the phases the participants had defined in their processes were being iterated through too fast to readily note on the paper worksheets. For instance, in my personal process, there is a phase for both the testing of the solution and the fixing of found bugs. Generally, a test phase would last mere seconds, as it involved running the program and comparing its output, and the resulting fix phase would also be very quick as the errors were normally quite apparent. This test and fix phase would be alternated between many times in the space of minutes. This behaviour is difficult to record accurately on a paper worksheet.

An alternate way of measuring this software development would be to utilise screen recording software on the development machine, then record the amount of time spend in each phase after the fact. This would prevent the overhead of the paper worksheet, as participants would not have to ensure they are noting their phases down at that time. It would also lead to a more

	Problem 1 Language A				Problem 2 Language A				Problem 1 Language B			
t	0	1	2	3	0	1	2	3	0	1	2	3
P1	270	170	117	114	50	61	45	40	105	80	60	60
P2	55	20	19	18	83	14	13	13	115	35	29	41
P3	205	165	58	64	150	115	107	102	159	113	87	83
P4	169	82	47	27	73	38	91	59	106	22	28	28
P5	150	75	63	45	75	79	27	26	73	135	28	66
μ	169.8	102.4	60.8	53.6	86.2	61.4	56.6	48.0	111.6	77.0	46.4	55.6

μ: Mean

Table 1: CITS8220 raw data

organic definition of the phases, as it was discovered that my personal process as defined did not map to my personal process in practice (further discussion in subsection 5.2 on page 11).

2.2 Issues

In terms of the software development itself, overall there were few issues with the implementation of the programs themselves, with the exception of $t = 2$. With $t = 2$, I was asked to develop a program that could calculate the area of a quadrilateral given four points on the Earth's surface. I had forgotten at this point that in the prior program we had used a simple non-right-angle triangle area formula in combination with great circle distance calculations to calculate the area of a given three points on the Earth's surface. Due to this, much time was lost researching the calculation of the area of a spherical triangle using spherical geometric methods, which proved to be difficult to understand and implement without prior understanding of them. Ultimately, the time this program took was completely blown out, and serves as an extreme outlier within my personal development data set.

To counteract this, the quadrilateral area program was redeveloped again blind as $t = 3$, and data analysis within this assignment is performed with either the outlier included and excluded (see individual captions for clarification) to account for the effect this has on the data set where appropriate.

3 Effort Models

Four separate models for effort were proposed, each of which models effort (in minutes, for our situation) on a different curve. These curves all share the same constants, a, b, and c, which represent different parts of the effort modelled, as well as the factor t representing time progression.

$$\mathit{Effort} = \frac{a + bct}{bt + 1} \tag{A}$$

$$\mathit{Effort} = (a - c)(t + 1)^{-b} + c \tag{B}$$

$$\mathit{Effort} = (a - c)\mathrm{e}^{-bt} + c \tag{C}$$

$$\mathit{Effort} = a + bt + ct^2 \tag{D}$$

Each model has a slightly different curve to represent the combination of factors;

- Model A represents these factors in terms of the positive part of a hyperbolic equation. This equation is based upon the work of John Musser, among others.

- Model B also represents these factors in terms of a hyperbolic equation, however, in this version the effect of b is more highly emphasised. This equation has its roots in psychological Learning Theory.
- Model C represents these factors in terms of the basic exponential equation $y = ae^{bx}$, with modifications to be negatively exponential and include the c parameter.
- Model D represents these factors in terms of a general quadratic parabolic equation.

3.1 Model Suitability

To suitably model a "learning curve", a model must meet certain criteria. Some primary criteria the model must cater for include;

- The model must show reduced effort over time, as knowledge acquisition over time results in reduced effort.
- The model must reflect that all tasks have a non-zero "minimum effort" required to complete them, regardless of level of knowledge (parameter c).
- The model must reflect that initial knowledge about a task affects the initial effort of the task on the curve at $t = 0$ (parameter a).
- The model must reflect that different people learn at different rates, therefore the curve's rate of decrease differs from person to person (parameter b).

Models A, B and C reflect these parameters well. All three of these models show the effort reduction $f(t+1) < f(t)$ for $t > 0$, and they include the relevant parameters a, b, and c to allow both the fixed effort, the initial effort and the learning curve to be appropriately modelled.

However, model D does not meet $f(t + 1) < f(t)$ for $t > 0$, as it is parabolic in nature. Due to this, this model will actually reflect *increased* effort as t rises beyond a certain point. This makes the model wholly unsuited for modelling effort, as such an increase of effort is not consistent with knowledge acquisition principles.

3.2 Best Fitting Model

Five people from the supplied CITS8220 data were chosen randomly and relabelled as persons P1–P5. For each of the three problem sets on which data exists (Problem 1 Language A, Problem 2 Language A, and Problem 1 Language B) the four datapoints of each of the five persons were averaged to produce 3 sets of four points (see table 1 on the previous page).

These points were fitted to all models,[1] using a non-linear least-squares regression method in Matlab (see appendix section D on page 25). Each of these graphs (see appendix section C on page 19) produced an R^2 value indicating the quality of the fit.[2] Table 2 on the following page shows these results for each person and model. The mean of these values is then produced to show on average which model is the best fit against the person datasets.

Models A, B, and C all fit very well to the data. The best fitting model was Model C, with the slightly higher $\overline{R^2} = 0.972$. Models A and B followed closely behind. Model D, due to its poor modelling capacity previously discussed, could not be fitted at all, with all R^2 values equalling zero.

[1] While assignment directions specified choosing 2 models, it was no more effort to fit against all 4.

[2] Traditionally, R^2 values are not negative. To quote the Matlab documentation; "Note that it is possible to get a negative R-square for equations that do not contain a constant term. Because R-square is defined as the proportion of variance explained by the fit, if the fit is actually worse than just fitting a horizontal line then R-square is negative." For our purposes, to avoid affecting averages, any R^2 value that is negative has been set to 0, indicating no fit.

3.3 Personal Process

In terms of my personal process, Models A though C all fit perfectly to the outlier excluded model ($R^2 = 1$); this is to be expected as that model only has three decreasing parameters. How these curves map to the data points can be seen in figure 1 on the next page. With the outlier included model, no correlation within the constraints of $b > 0, a > c > 0$ could be found. Exceedingly weak correlations without this constraints can be seen in appendix subsection B.2 on page 17.

4 Measurement Discussion

4.1 Prior Learning, Different Specification

Using the CITS8220 fitted data collected, we can investigate to what extent we carry our learning from previous projects over to subsequent projects of similar type but different specification; an investigation of knowledge acquisition. Within the data set provided, P1LA and P2LA provide the data points we can use to compare. Parameter a deals with the initial effort required in such situations, so its value will be examined to provide insight.

In Table 3 on page 7, $\%\Delta_1$ represents the percentage change between the a of these two problem sets. We can see that on average, there was a reduction of the initial effort parameter of 49.6% when switching between problem 1 and 2, both of which were written in the same programming language, but had different specifications.

This result shows that to a massive extent learning from previous projects is carried over to other projects in the same area but with a different specification. In this instance, nearly half the initial effort was reduced for the subsequent problem.

4.2 Prior Learning, Different Resources

Similar to the prior section, using the CITS8220 data we can examine the extent which we carry our learning from previous projects over to subsequent projects that require different resources by examining the a parameter of our fitted data; this is an investigation of knowledge encoding. P1LA and P1LB provide the data points we can use to compare.

In Table 3 on page 7, $\%\Delta_2$ represents the percentage change between these two. We can see that on average, there was a 34.3% reduction in the initial effort when switching from programming language A to programming language B, but keeping the problem the same.

This reduction in effort based on resource changes is less than that from the above specification changes, however it is still significant, representing a third reduction in effort when moving between programming languages.

4.3 Effect of Practice on Estimation

With the estimate data provided from CITS5502 in Table 4 on page 7, it is possible to examine the relationship between practice and accuracy of estimation. By viewing the percentage differences ($\%\Delta$) for the estimates and the actuals, we can see that for the first two problems, the percentage difference in the mean time taken (μ) is decreasing by a factor of first 14.7%,

	Model				
	A	B	C	D	μ^1
P1LA	0.991	0.988	0.995	0.000	0.991
P2LA	0.987	0.987	0.982	0.000	0.985
P1LB	0.929	0.928	0.939	0.000	0.932
μ	0.969	0.968	0.972	0.000	

[1] Excludes Model D values

Table 2: CITS8220 fitting average R^2

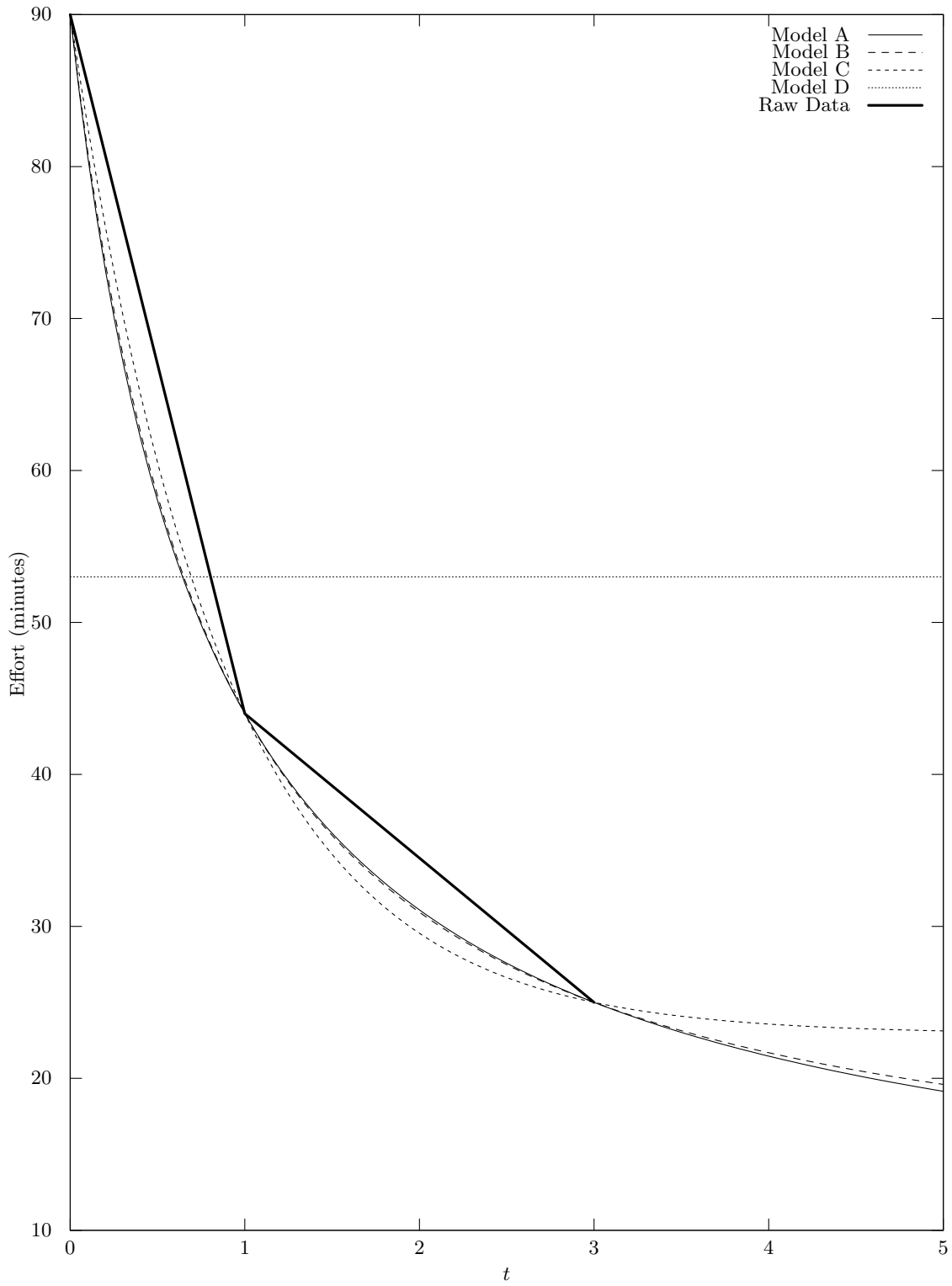

Figure 1: Models A–D fitted to personal data, excluding outlier

Model	Problem P1LA	P2LA	P1LB	$\%\Delta_1$	$\%\Delta_2$
A	170.605	86.082	112.088	-49.5	-34.3
B	171.213	86.030	112.253	-49.8	-34.4
C	170.400	85.916	112.429	-49.6	-34.0
D	–	–	–	–	–
μ	170.739	86.010	112.257	-49.6	-34.3

$\%\Delta_1 = (\text{P2LA} - \text{P1LA})/\text{P1LA} \times 100$
$\%\Delta_2 = (\text{P1LB} - \text{P1LA})/\text{P1LA} \times 100$
D is excluded from averages due to no fit.

Table 3: CITS8220 fitting a parameter comparison

then 27.4%. We can similarly see that the $\%\Delta$ of the Standard Deviation decreases between problems $t = 0$ and $t = 1$. Problem $t = 2$ presents an anomaly in this dataset, as $\%\Delta$ actually increases in both μ and Standard Deviation (σ). This anomaly however can be explained as a result of the problem specification being significantly different, calling the validity of this data point within the set into question.

Unfortunately the data at hand does not allow us to make strong predictions about the nature of practice improving the quality of estimates, as the dataset is not large enough. It is also hard to draw conclusions, as we have not isolated the estimation aspect appropriately; problem familiarity is also increasing, resulting in actuals that are also dramatically decreasing.

Working with what does exist, the data between programs 0 and 1 does show that the difference between the actual and the estimate are increasing. There are several hypotheses as to how this dataset would continue if it were larger;

1. As users become more knowledgeable, the Mean $\%\Delta$ erratically approaches 0. This would indicate with enough knowledge users will approach perfect estimates.

2. As users become more knowledgeable, the Mean $\%\Delta$ approaches an n where $n < 0$. This would indicate with enough knowledge users will approach an overestimate that generally has the same margin.

3. On a long enough time scale, $\%\Delta$ will average to 0.

To properly determine the extent at which practice enables the production of more predictable estimations of effort, more experimental variables would need to be controlled.

4.4 Learning Patterns

The properties of the "learning curve" in each discussed model is described by the b parameter. This parameter modifies the rate of decrease of each model by a factor, representing the difference in rate of learning between each individual. By examining Table 5a on the following page, we can determine how similar this "learning curve" was for each of the three different programming problems present in the CITS8220 data set.

By examining this data we can see that the learning curve value b increases by approximately 7% through each subsequent generation of the program. This translates to a curve that decreases

	$t = 0$			$t = 1$			$t = 2$		
	Est	Actual	$\%\Delta$	Est	Actual	$\%\Delta$	Est	Actual	$\%\Delta$
μ	107.4	91.6	-14.7	74.8	54.3	-27.4	100.6	123.7	23.0
σ	38.1	31.6	-17.1	25.7	24.7	-3.9	40.4	59.4	47.0

σ: Standard deviation
$\%\Delta$: Percentage difference between Est and Actual

Table 4: CITS5502 estimation data

more quickly, i.e. a higher number means learning more quickly. This is consistent with an understanding of knowledge acquisition; in each subsequent problem, the individual knows more about the problem area, and are thus able to complete subsequent problems more quickly.

4.5 Limiting Factors

Within each of the effort models proposed, parameter c exists as the "limiting factor". This factor expresses that even with perfect knowledge, a problem will take a non-zero about of time to program. Consequently, in each model the c parameter is added to the model, shifting it upwards by the specific amount.

By examining the CITS8220 data within Table 5b, we can view the values for c for each of the problem sets and models to determine what this limiting factor is. For the second and third problem set, this number is around 35 minutes. For the first problem set's model C, it is similar, but for models A and B in that set, fitting did not produce a C parameter of appreciable size.[3]

4.6 Solution Consistency

Table 6 on the following page provides a variety of metrics to measure the complexity of the code personally produced as part of this assignment. We can use these metrics and the coefficient of variation ($c_v = \sigma/\mu$) to determine now similar (or consistent) each of the produced solutions are as knowledge increases. If the solutions are similar in complexity, we can more readily assume that given the same amount of knowledge regarding the problem space at each t (not the case here, as learning is occurring) each solution would have taken a similar amount of time to create. This helps us eliminate some sources of variation from the data set.

The metric we will use to examine the complexity is the LLOC, or Logical Lines of Code, which measures the number of lines of code if each logical statement was placed on its own line. Using LLOC eliminates issues whereby more complex programs can be expressed in fewer lines by making each line more complex, and also removes blank lines, which are irrelevant to program complexity. Other common metrics of software complexity are included to provide context for the reader. The c_v provided by this metric across the personal programs written is 0.336, suggesting that these programs overall do not differ much in complexity. Excluding the outlier, c_v' (0.328) shows they differ even less. This is correct at a broad level, as the difference between the minimum and maximum LLOC is only 30, however this metric does not adequately express the difficulty of encoding the knowledge associated with each program.

If we wish to more easily represent the difficulty of each program, we can introduce a new unit of measurement l/min (Logical Lines of Code per minute), which expresses on average how difficult a given program was to encode. Excluding $t = 2$, we can see that this is a trend upwards, which makes sense, as knowledge acquisition becomes easier with more practice.

[3] C was non-zero, but very small in unrounded results.

	Problem			%Δ	
	P1LA	P2LA	P1LB	1–2	2–3
A	0.759	0.859	0.852	13.2	-0.8
B	0.845	0.798	0.909	-5.6	14.0
C	0.767	0.881	0.948	14.8	7.6
D	–	–	–	–	–
μ	0.791	0.846	0.903	7.5	6.9

(a) b parameter comparison

	Problem			%Δ	
	P1LA	P2LA	P1LB	1–2	2–3
A	0.00	35.32	24.90	–	-29.5
B	0.00	31.23	25.01	–	-19.9
C	37.52	46.87	46.23	24.9	-1.4
D	–	–	–	–	–
μ	12.51	37.81	32.05	24.9	-16.9

(b) c parameter comparison

%Δ n–m: % diff between problem n and m.

Table 5: Parameter comparisons

t	Actual	Est.	Est. $\pm$	Avg CC	LOC	LLOC	SLOC	H. Diff.	H. Len.	l/min
0	105	120	40	1.0	80	21	64	3.157	96	0.20
1	54	40	10	1.0	92	42	68	4.052	219	0.78
2	461	120	50	1.3	105	51	77	4.675	506	0.11
3	25	60	20	1.3	61	36	43	4.000	246	1.44
μ	161.25	85.00	30.00	1.15	84.50	37.50	63.00	3.97	266.75	0.63
σ	202.55	41.23	18.26	0.17	18.70	12.61	14.40	0.62	172.34	0.61
c_v	1.256	0.485	0.609	0.151	0.221	0.336	0.229	0.157	0.646	0.972
μ'	61.33	73.33	23.33	1.10	77.67	33.00	58.33	3.74	187.00	0.81
σ'	40.50	41.63	15.28	0.17	15.63	10.82	13.43	0.50	79.96	0.62
c_v'	0.660	0.568	0.655	0.157	0.201	0.328	0.230	0.134	0.428	0.770

c_v: Coefficient of variation
x': Excludes $t = 2$ outlier from calculation x
H.: Halstead
CC: Cyclomatic Complexity
LOC: Total number of lines of code, including blank lines.
LLOC: Total number of lines of code; each logical line contains 1 statement.
SLOC: Total number of lines of code, excluding blanks.
l**/min**: Logical Lines of Code per minute coding

Table 6: Personal program times

5 Personal Development

5.1 Work Variation & Major Shifts

In my personal development for the provided problems, there were several areas in which variation in effort were introduced. The primary area, which has been discussed at length thus far, was for $t = 2$, in which due to a misunderstanding, the problem complexity was made artificially harder through the introduction of complex spherical geometry calculations. This significantly inflated the amount of effort as my knowledge of the problem was severely reduced.

Excluding that point, overall the effort across the problems reduced, as my knowledge of the problem, as well as the resources had increased far more than the problem size and complexity. This downward trend can be seen in Figure 1 on page 6 and within the data of Table 6.

Figure 2 on the next page shows the changing proportion of the project spend in each of the broad categories of work. I converted the data of my personal process to the specified four broader categories for ease of presentation (how this conversion occurred can be seen in appendix section E on page 27).

Within the figure, we can see that from $t = 0$ to $t = 1$, the knowledge encoding phase decreased; this touches on the fact that much of the problem understanding that was gained in $t = 0$ was portable to $t = 1$, leading to a reduction of effort in that regard.

The data surrounding $t = 2$ clearly shows where the major issues were in that project; both the solution validation and the knowledge encoding phases of that project ballooned out of proportion. Both of these were due to misunderstandings as to the units used in many of the spherical geometry equations found to assist with the solution; much iteration was done to determine what these units were and what conversions were required to get to them.

Beyond those discussed areas, if one excludes $t = 2$, there is no more than a 12% variation between each stage, which can be attributed to fluctuations in problem difficulty and particular pitfalls.

Only one particular trend could be noticed with these data points (even excluding $t = 2$); that is the solution valuation as a proportion is slowly increasing (see figure 3 on the next page). It is hard to say if this is an actual trend, or just a data anomaly. One hypothesis is that the tutorial sessions that the program is coded in have something to do with it; if there is a set amount of time in which people wish to finish the program, if people finish early, they might spend some of the remaining time ensuring that the program operates correctly. If more trend

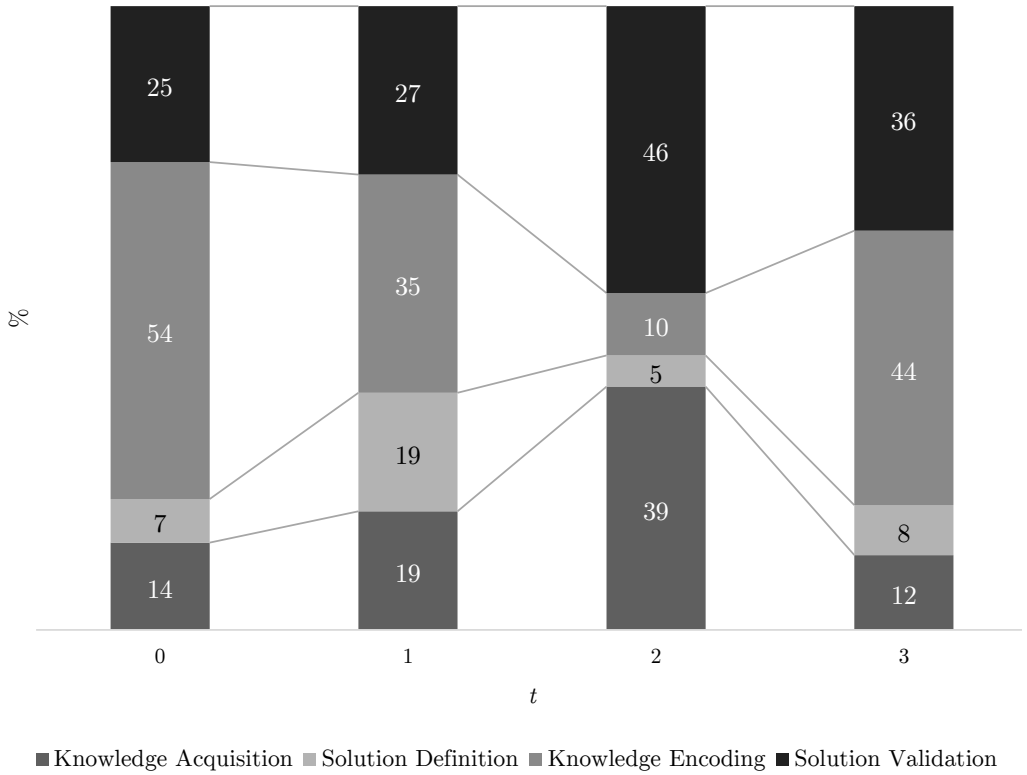

Figure 2: Changes in effort as proportion of whole (including outlier)

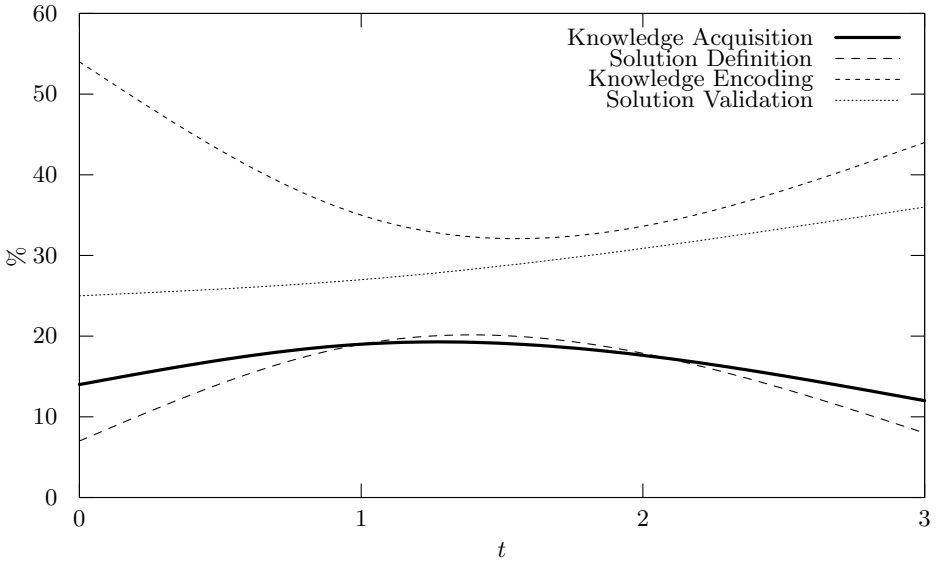

Figure 3: Changes in effort excluding outlier

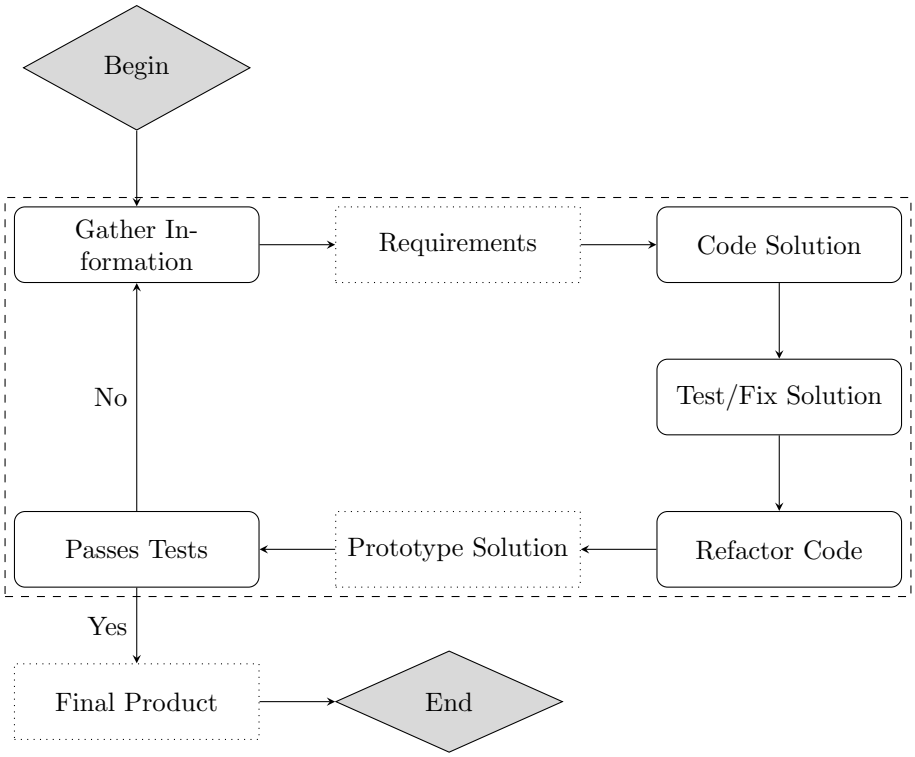

Figure 4: Revised Personal Process

information was wishing to be derived, then a much larger data set would need to be used.

5.2 Personal Process Evolution

My personal process (original version shown in appendix section A on page 13) has evolved significantly over the duration of the course. It was quickly realised with the application of the process to real world situations that many of the parts of the process are not observed or happen implicitly as part of other phases instead. Some of the differences observed between the "ideal" and real version include;

- Design Options are rarely evaluated explicitly: A design option is picked, and the code is refactored throughout with changing knowledge.
- Technical Options are not evaluated at all with such a small problem: A language that I am proficient in was picked at the very start, and due to a ban on using third-party libraries, no further decisions of that nature needed making.
- The Test Solution and Fix Found Bugs parts of the process are effectively merged in practice: They form a very tight iteration in which the code is modified, a new prototype is generated, which is then tested again.
- Code refactoring takes place at every stage of the process: Usually when it is noticed that a particular section is a source of error or is particularly complex.
- Feedback is not particularly sought within these problems: As the problems have very objective answers, testing acts as the core feedback mechanism to determine if the solution is satisfactory.

A revised version of this personal process can be seen in Figure 4.

6 Conclusion

Four different effort models were examined to determine how easily they could model real-world data, and if any conclusions could be derived from the parameters that were produced from the fitting of those models.

The difficulties in collecting the CITS5502 real world data were discussed, and a different approach was proposed to eliminate them. Problems with an outlier in my personal CITS5502 data were discussed, an alternate data was generated to help overcome this problems to some extent.

Both the CITS8220 data provided and the CITS5502 data created were fitted against all four models, with similarly broad success in the case of Models A though C, but with utter failure in the case of Model D due to poor model design, which is discussed in some detail.

The parameters a, b and c generated from the CITS8220 fitted data were discussed in great detail, outlining the function of each of these parameters in the model, and analysing how they changed between each of the three CITS8220 problem sets provided interesting insight into how prior experience in knowledge acquisition and knowledge encoding can provide significant advantage into subsequent problems in the same problem domain or resource domain respectively.

Estimation data from the whole CITS5502 cohort was examined to attempt to draw conclusions regarding how estimation can improve over subsequent program generations, however due to Problem 2 being of significantly greater initial complexity, it is hard to draw any firm conclusions with that data. Several hypotheses were proposed as to how the data could continue to trend.

The consistency between each personal solution was also analysed within the context of Logical Lines of Code and the actual time taken, coming to the conclusion that the solutions did differ in complexity, but only significantly in the case of $t = 2$.

The variation between the different personal process phases was discussed in detail, with overall (outlier excluded), little by way of obvious trend to interpret. More data was needed for a thorough analysis.

Finally, how my personal process evolved over this unit's duration was discussed, as it had significantly, and a revised model of the process was provided.

Appendices

A Personal Process

Refer to Assignment 1 for flowchart key.

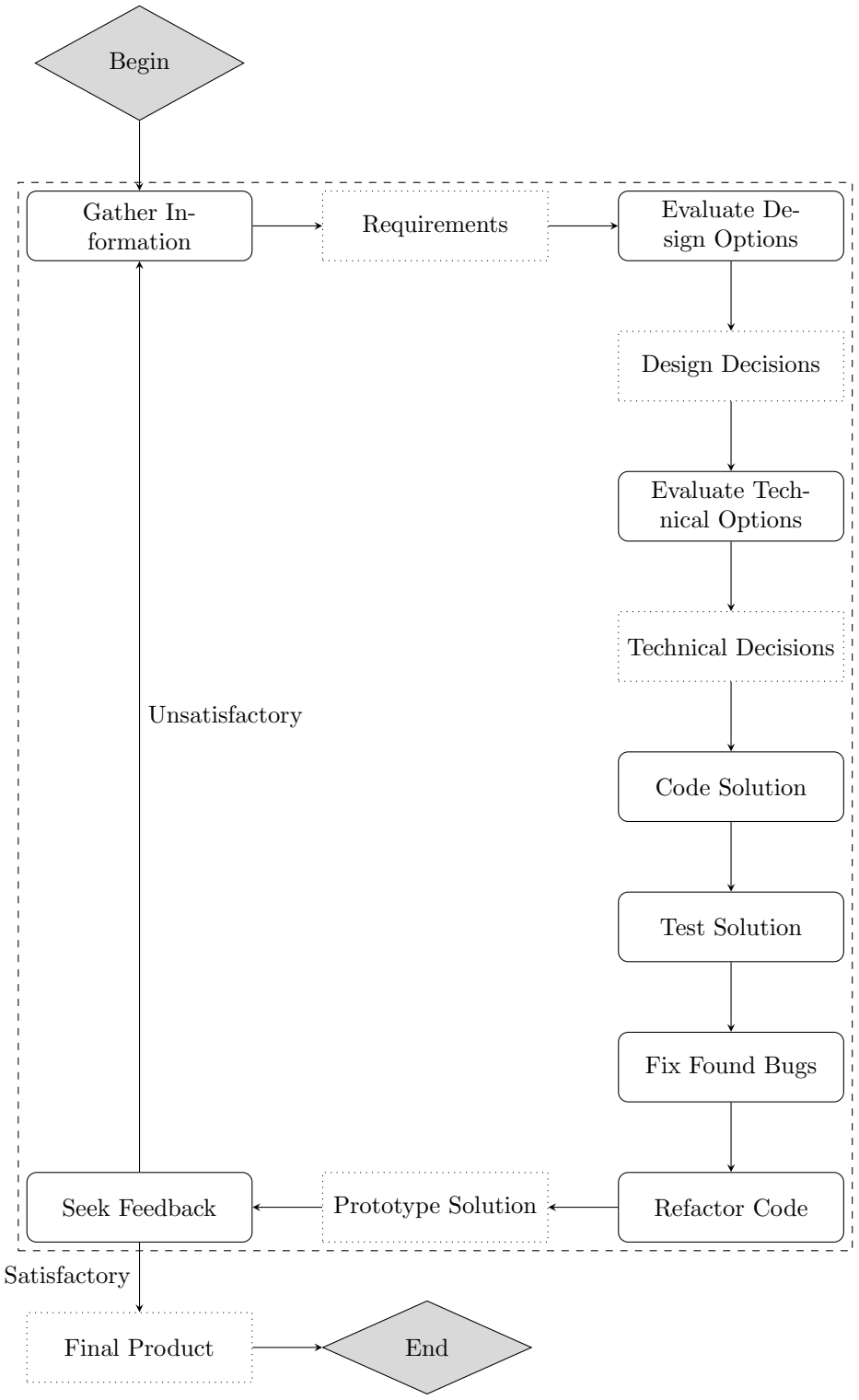

Figure 5: Personal Process

B Personal Results

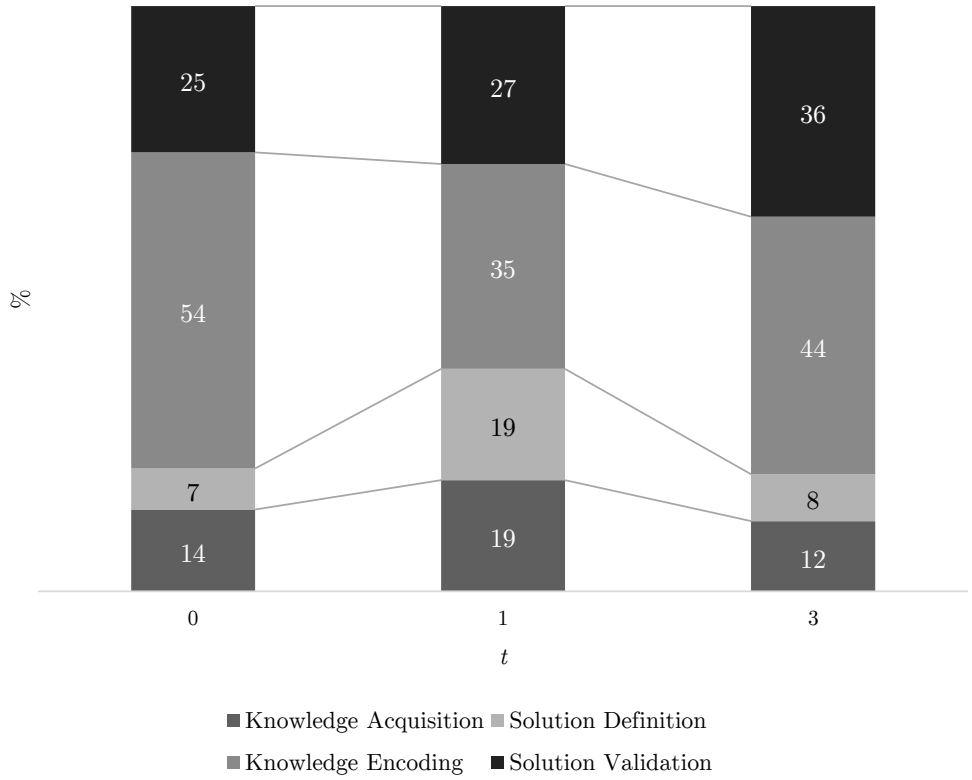

Figure 6: Changes in effort as proportion of whole (excluding outlier)

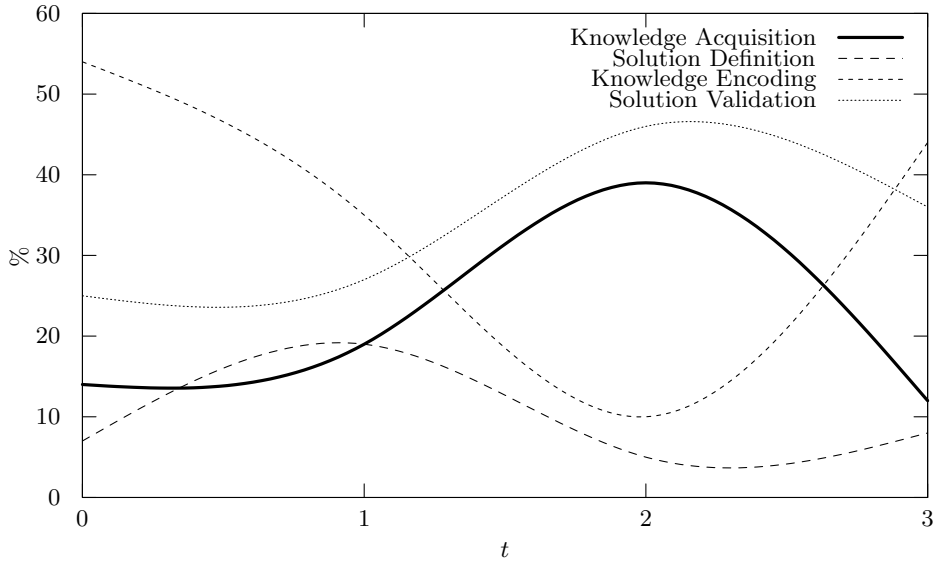

Figure 7: Changes in effort including outlier

B.1 Excluding Outlier

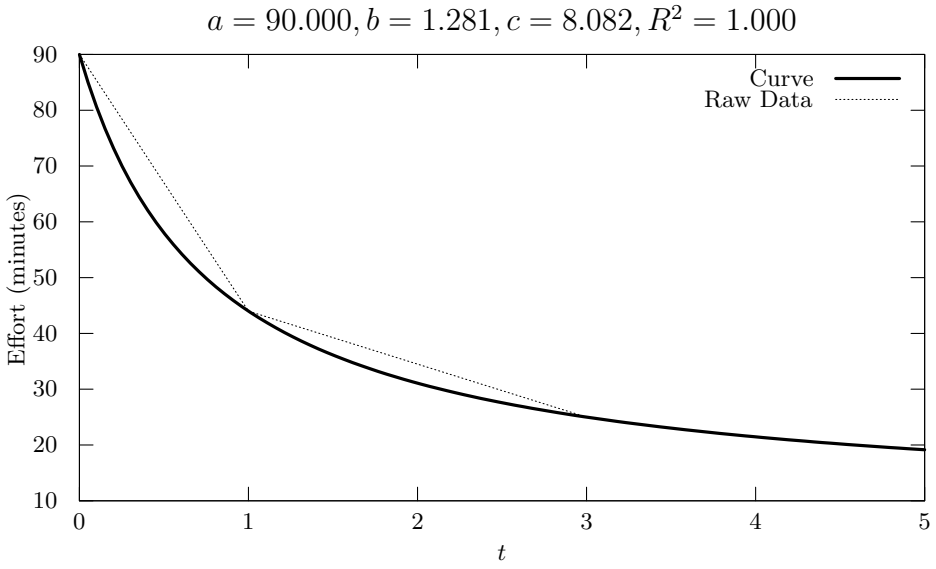

Figure 8: $\frac{a+bct}{bt+1}$

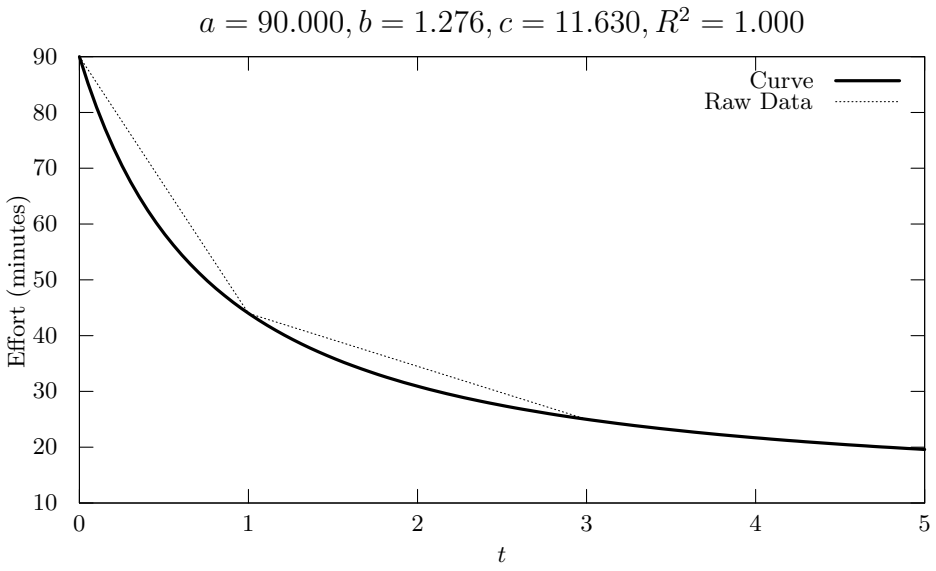

Figure 9: $(a - c)(t + 1)^{-b} + c$

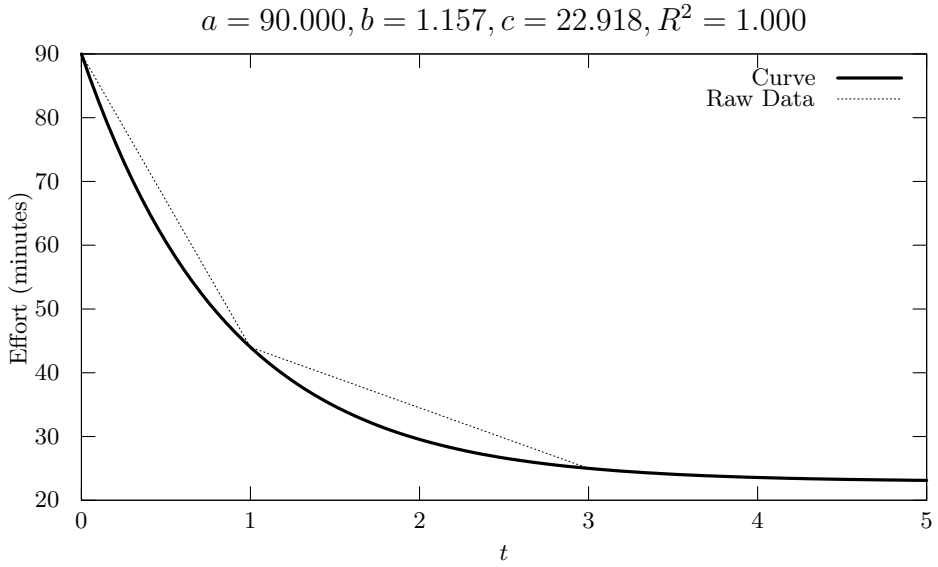

Figure 10: $(a - c)e^{-bt} + c$

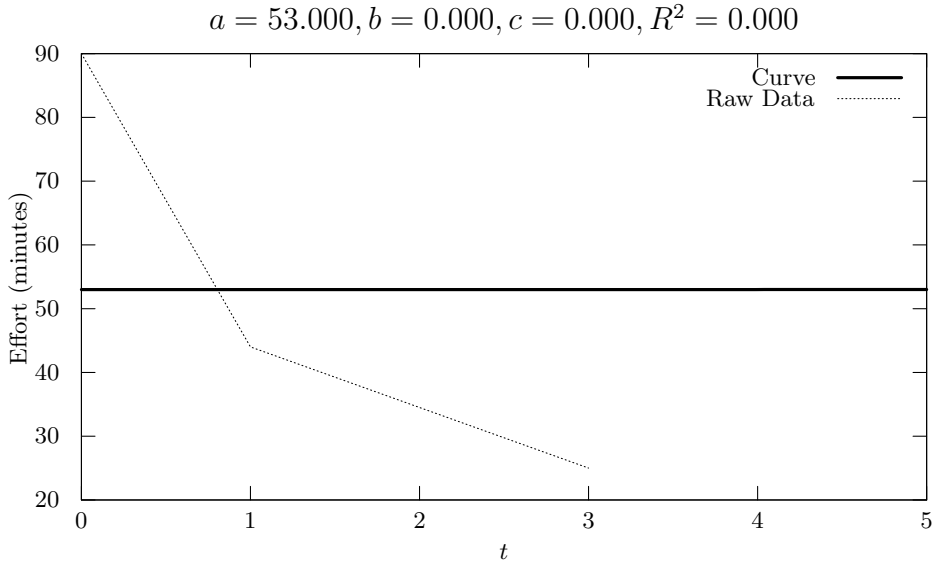

Figure 11: $a + bt + ct^2$

B.2 Including Outlier

Note: These graphs do not meet specified fitting constraints.

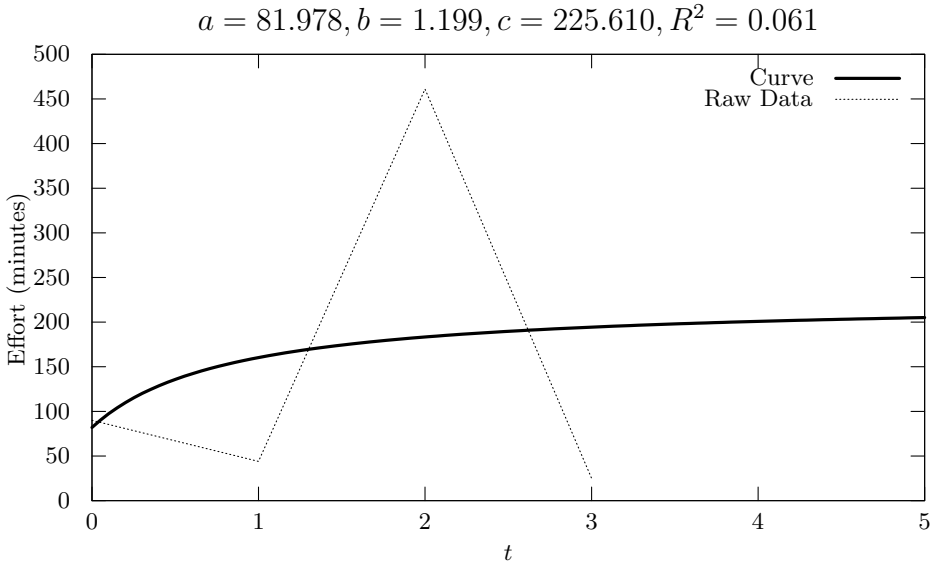

Figure 12: $\frac{a+bct}{bt+1}$

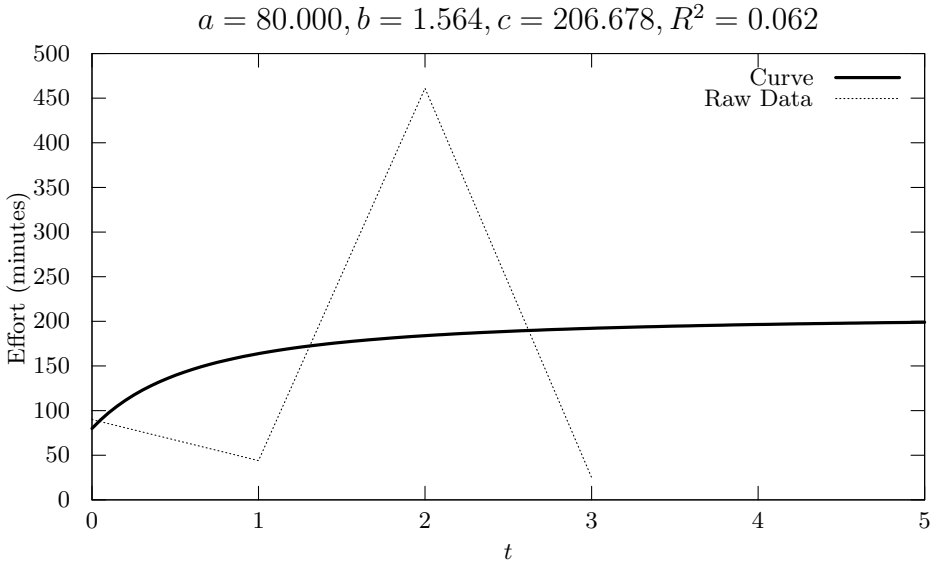

Figure 13: $(a-c)(t+1)^{-b}+c$

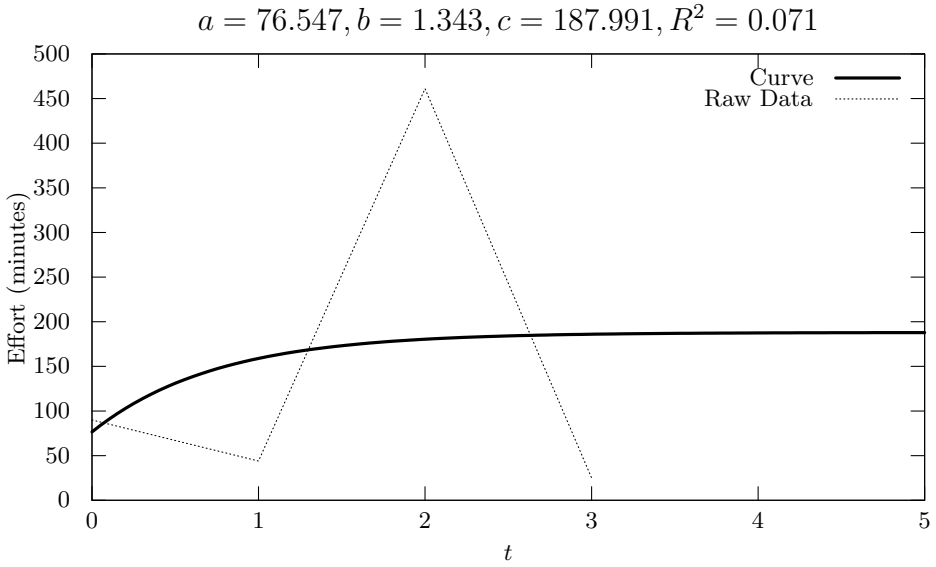

Figure 14: $(a - c)e^{-bt} + c$

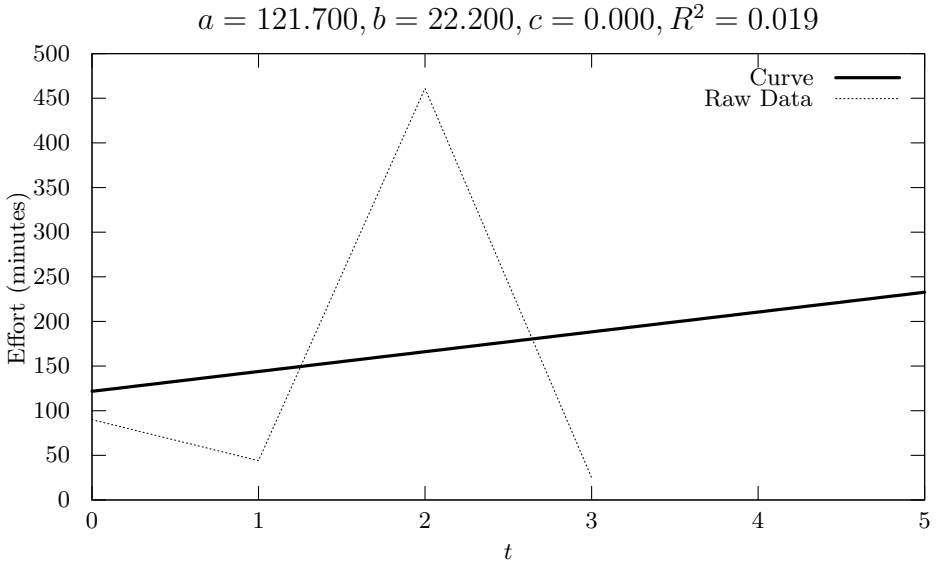

Figure 15: $a + bt + ct^2$

C CITS8220 Results

C.1 Model A

$$\frac{a + bct}{bt + 1}$$

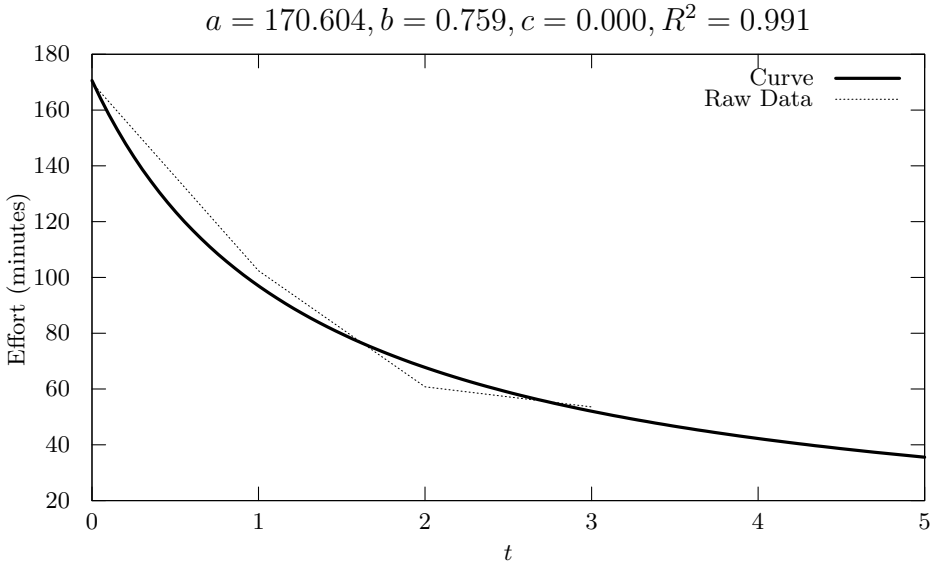

Figure 16: Problem 1 Language A

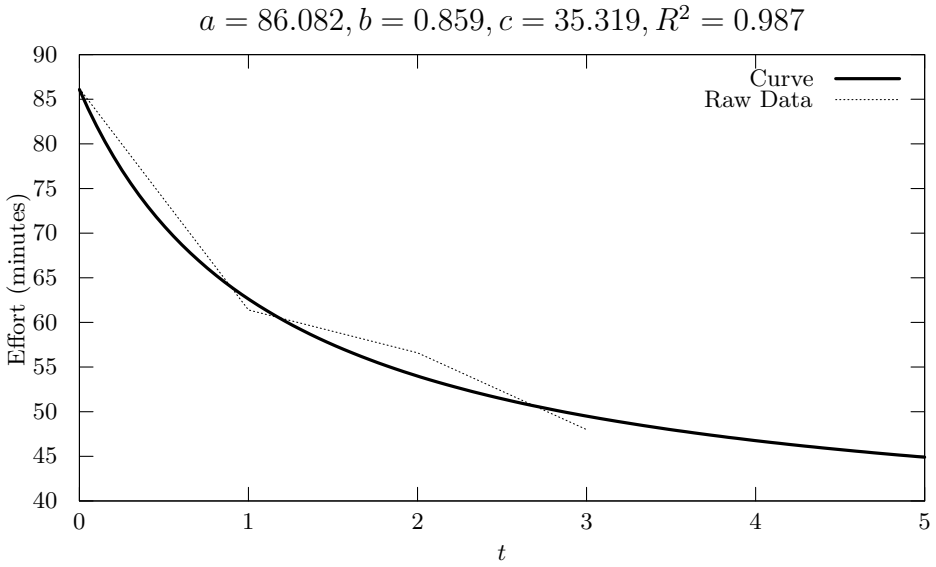

Figure 17: Problem 2 Language A

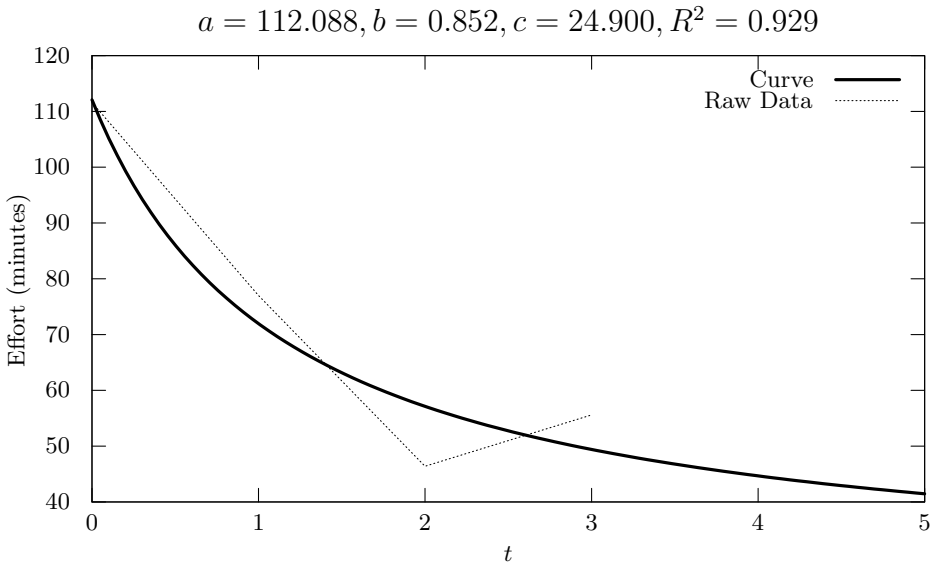

Figure 18: Problem 1 Language B

C.2 Model B

$$(a-c)(t+1)^{-b}+c$$

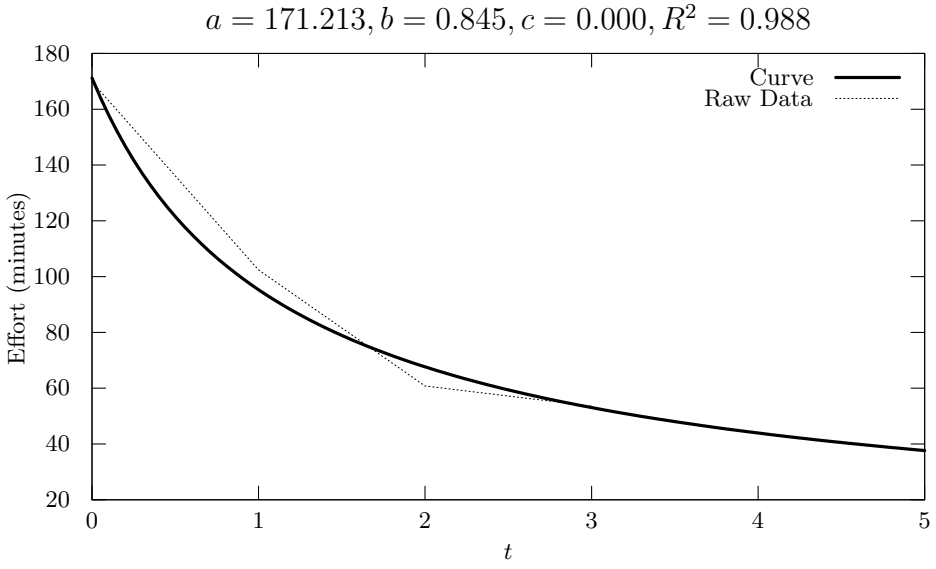

Figure 19: Problem 1 Language A

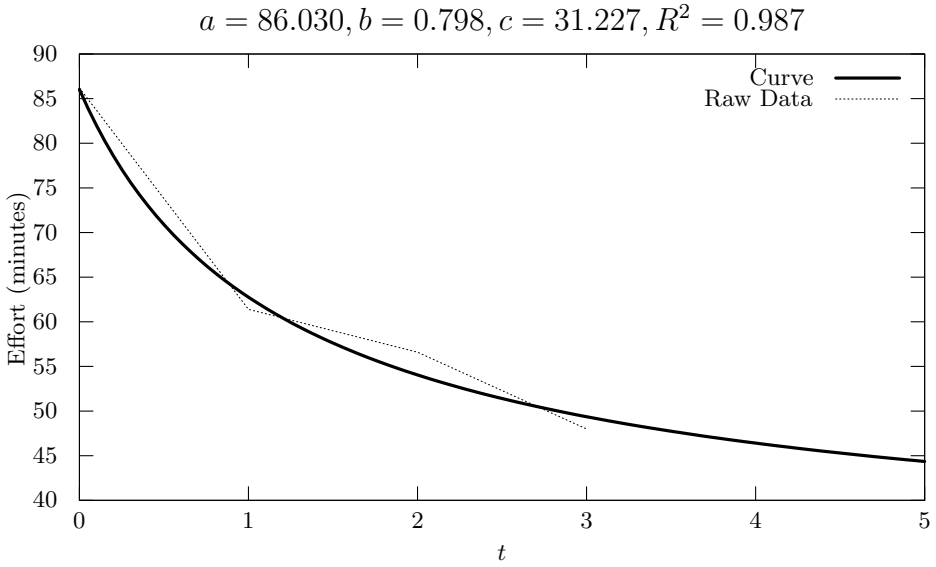

Figure 20: Problem 2 Language A

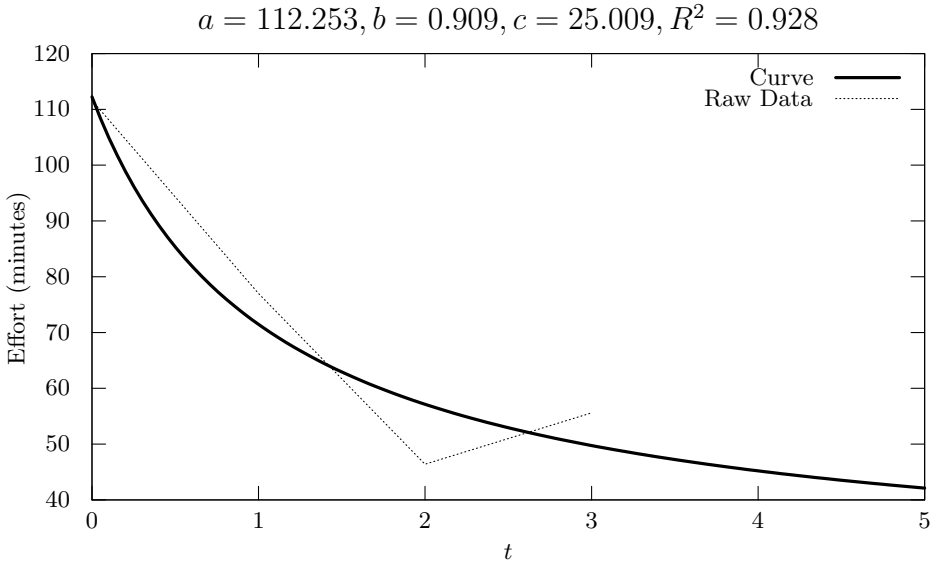

Figure 21: Problem 1 Language B

C.3 Model C

$$(a-c)e^{-bt}+c$$

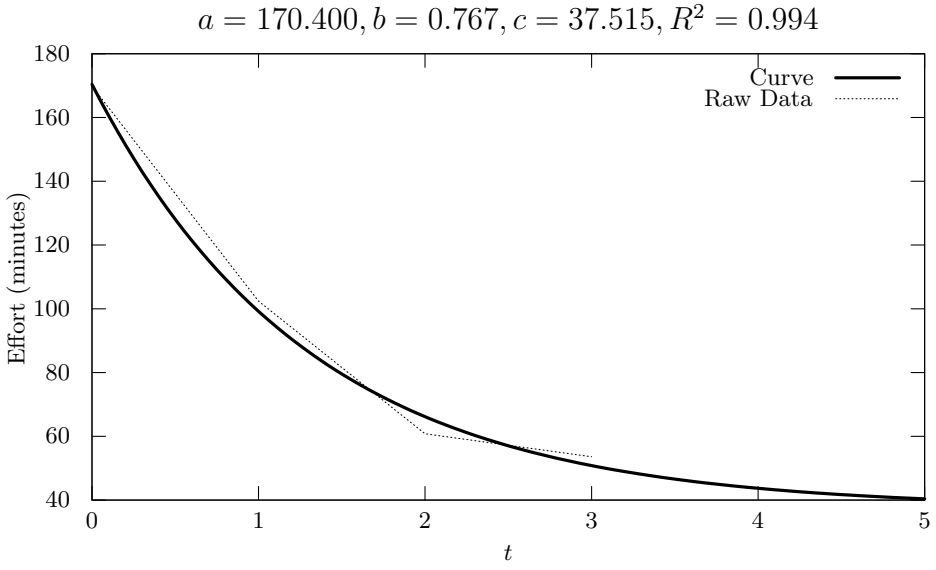

Figure 22: Problem 1 Language A

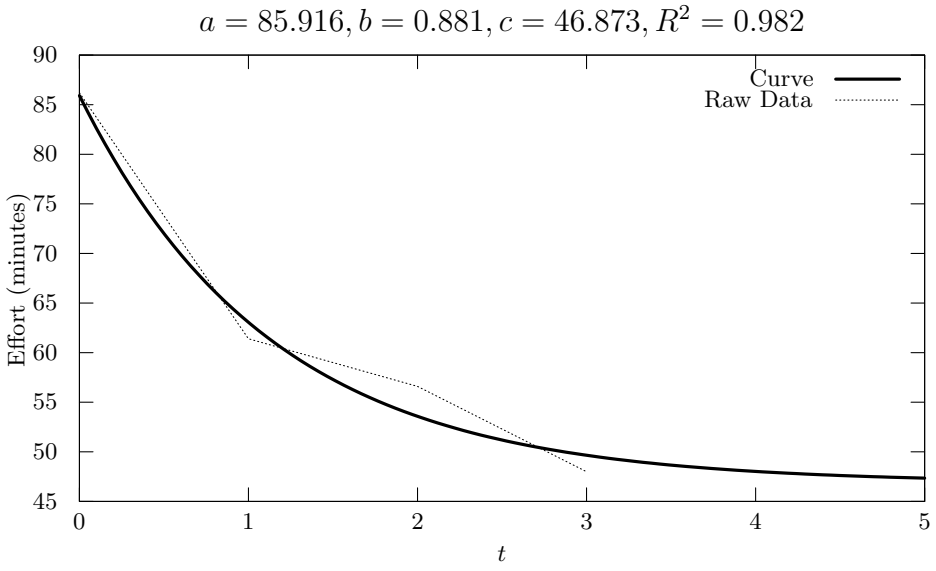

Figure 23: Problem 2 Language A

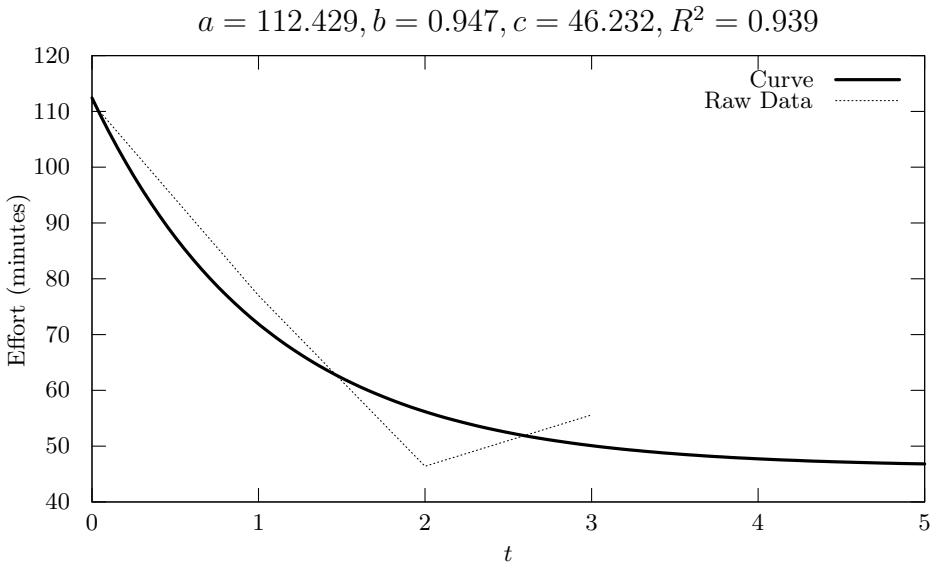

Figure 24: Problem 1 Language B

C.4 Model D

$$a + bt + ct^2$$

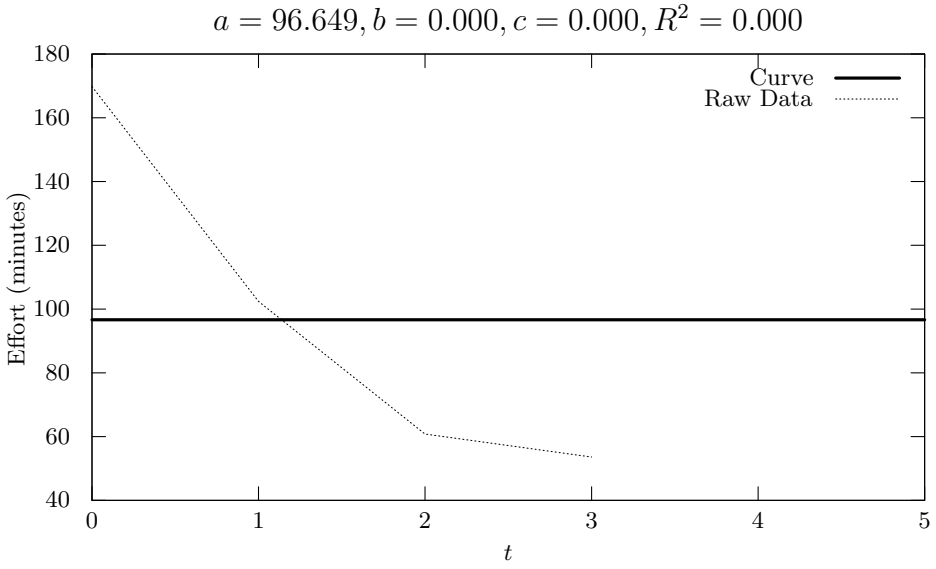

Figure 25: Problem 1 Language A

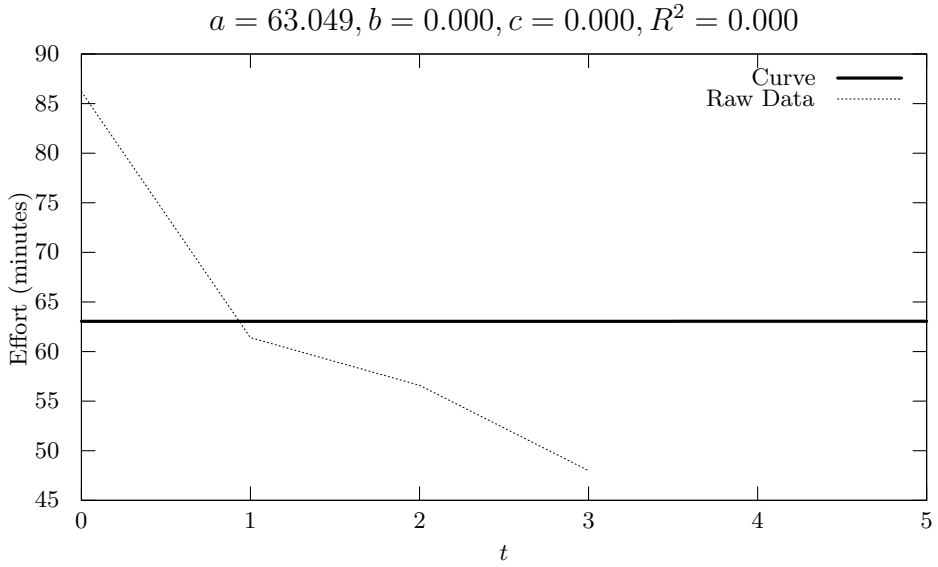

Figure 26: Problem 2 Language A

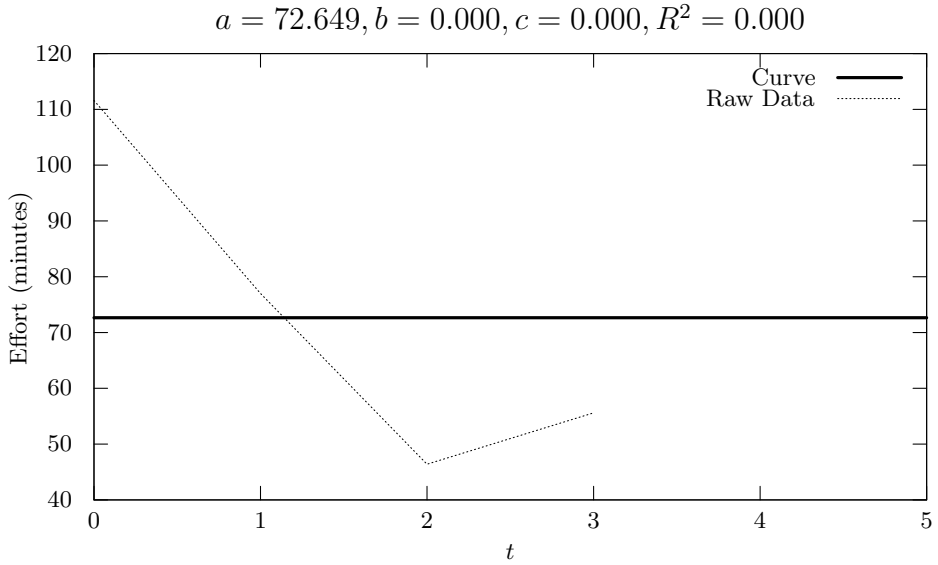

Figure 27: Problem 1 Language B

D Graph Generation Code

```
% Written for Matlab. Outputs GnuPlot files which produce LaTeX figures.

% createFit.m
function [as, bs, cs, rsqs] = createFit(prefix, dataX, dataY, init)
eqs = char({'(a+b*c*x)/(b*x+1)', '(a-c)*(x+1)^(-b)+c', ...
        '(a-c)*exp(-b*x)+c', 'a+b*x+c*x^2'});

eqs_gp = char({'(a+b*c*x)/(b*x+1)', '(a-c)*(x+1)**(-b)+c', ...
        '(a-c)*exp(-b*x)+c', 'a+b*x+c*x**2'});

sze = size(dataY);
data = [dataX, dataY];
dlmwrite(strcat(prefix, '-data.txt'), data, '\t');

as = zeros(4, sze(2));
bs = zeros(4, sze(2));
cs = zeros(4, sze(2));

rsqs = zeros(4, sze(2));

for i=1:4,
  for j=1:sze(2),
    [xData, yData] = prepareCurveData(dataX, dataY(1:end,j));

    % Set up fittype and options.
    ft = fittype( eqs(i,1:end), 'independent', 'x', 'dependent', 'y' );
    opts = fitoptions( ft );
    opts.Display = 'Off';
    opts.Lower = [0.0002, 0.0001, 0.0001];
    opts.StartPoint = init;
    opts.Upper = [Inf Inf Inf];

    % Fit model to data.
    [fitresult, gof] = fit( xData, yData, ft, opts );
    rsq = gof.rsquare;
    indx = coeffnames(fitresult);
    vals = coeffvalues(fitresult);

    a = vals(find(ismember(indx,'a')));
    b = vals(find(ismember(indx,'b')));
    c = vals(find(ismember(indx,'c')));

    if rsq < 0
      rsq = 0;
    end

    as(i,j) = a;
    bs(i,j) = b;
    cs(i,j) = c;
    rsqs(i,j) = rsq;

    fp = fopen(strcat(prefix, '-plot', int2str(i), '-', int2str(j), '.p'), 'w');

    fprintf(fp, 'set term tikz monochrome dashed size 5in,3in font ",9"\n');
    fprintf(fp, 'set style func linespoints\n');
    fprintf(fp, 'set style data lines\n');
```

```
    fprintf(fp, 'set nogrid\n');
    fprintf(fp, 'set xrange [0:5]\n');
    fprintf(fp, 'set xlabel "$t$"\n');
    fprintf(fp, 'set ylabel "Effort (minutes)"\n');
    fprintf(fp, 'set title "$a = %.3f, b = %.3f, c = %.3f, R^2 = %.3f$"\n', ...
                a, b, c, rsq);
    fprintf(fp, 'set title font ",12"\n');
    fprintf(fp, 'set format xy "%%g"\n');
    fprintf(fp, 'set output "%s-plot%d-%d.tex"\n', prefix, i, j);
    fprintf(fp, 'a = %f\n', a);
    fprintf(fp, 'b = %f\n', b);
    fprintf(fp, 'c = %f\n', c);
    fprintf(fp, 'f(x) = %s\n', eqs_gp(i,1:end));
    fprintf(fp, 'plot f(x) lw 3 lt 1 smooth csplines title "Curve", ');
    fprintf(fp, '"%s" using 1:%d lw 1 lt 4 title "Raw Data"\n', ...
                strcat(prefix, '-data.txt'), j+1);
    fprintf(fp, 'set output\n');
    fprintf(fp, 'unset ylabel\n');
    fprintf(fp, 'unset xlabel\n');

    fclose(fp);
  end
end
end

% est.m
data1 = [169.8, 102.4, 60.8, 53.6;
          86.2, 61.4, 56.6, 48;
          111.6, 77, 46.4, 55.6].';

data2 = [90; 44; 461; 25];
data3 = [90; 44; 25];

dataX = [0; 1; 2; 3];

dataX2 = [0; 1; 3];

[a,b,c,r] = createFit('other', dataX, data1, [0.1321 0.7227 0.1104])
[a,b,c,r] = createFit('myincl', dataX, data2, [0.7232 0.3474 0.6606])
[a,b,c,r] = createFit('myexcl', dataX2, data3, [0.7232 0.3474 0.6606])
```

E Programs

E.1 Raw Timings

	Program			
	1	2	3	4
Code	57	19	46	11
Test/Fix	26	15	210	9
Refactor	7	10	13	2
Gather	15	10	180	3
Design	0	0	12	0
Total	105	54	461	25

Table 7: Raw Timings

Category	Maps to
Knowledge Acquisition	Gather
Solution Definition	Design, Refactor
Knowledge Encoding	Code
Solution Validation	Test/Fix

Table 8: Categorising phases

E.2 Program 0: Distance

```
# Written for Python 3.4
# Output:
# Distance calculations!
#
# Distance from perth to uwa
#  = 4.287899604948252
#
# Distance from perth to greenwich
#  = 14485.978495782614
#
# Distance from perth to southpole
#  = 6715.917481245382
#
# Distance from uwa to greenwich
#  = 14485.767563601663
#
# Distance from uwa to southpole
#  = 6712.656749767587
#
# Distance from greenwich to southpole
#  = 15192.844590121906

from decimal import *
from math import radians
import itertools

# In degrees
PLACES = {
  #'uwa':         ('-31.981179', '115.81991'),
  #'perth':       ('-31.954265', '115.8523935'),
  #'greenwich':   ('51.4825766', '-0.0076589'),
```

```
  #'southpole':  ('-85', '0'),
  #'mackay':     ('-21.141857', '149.1829713'),
  'unihall':    ('-31.974053', '115.819057'),
  'darwin':     ('-12.5948609','131.0078759'),
  'melbourne':  ('-37.8602828','145.079616'),
}

RADIUS_EARTH = 6378



# Uses the Haversine formula
# Assumes spherical earth and radian coordinates
def distance(coord1, coord2):
  from math import asin, sqrt, cos, sin

  (lat1, long1), (lat2, long2) = coord1, coord2

  def haversin(v):
    return (sin(v/2))**2

  return 2 * RADIUS_EARTH * asin(
      sqrt(
        haversin(lat2 - lat1) +
        cos(lat1) * cos(lat2) * haversin(long2 - long1)
      )
    )



if __name__ == "__main__":
  # Convert string coords into Decimal radian objects
  conv_places = {
    x : ( radians(Decimal(y1)), radians(Decimal(y2)) )
    for x, (y1, y2) in PLACES.items()
  }

  print('Distance calculations!')
  print()

  # Iterate over all place combinations and print the answers
  for (n1, c1), (n2, c2) in itertools.combinations(conv_places.items(), 2):

    print('Distance from {} to {}'.format(n1, n2))
    print('  = {}'.format(distance(c1, c2)))
    print()
```

E.3 Program 1: Area

```
# Written for Python 3.4
# Output:
# Calculating the area of ('unihall', 'darwin', 'melbourne')
# Distance between unihall and darwin
#  = 2657.806808418724
# Distance between darwin and melbourne
#  = 3140.2840494728316
# Distance between melbourne and unihall
#  = 2737.824326481584
# Area between points
#  = 3443496.4985877923


PLACES = {
  'unihall':    (-31.974053, 115.819057),
  'darwin':     (-12.5948609,131.0078759),
  'melbourne':  (-37.8602828,145.079616),
}

AREAS = [
  ('unihall', 'darwin', 'melbourne'),
]

RADIUS_EARTH = 6378

# From https://docs.python.org/3.1/library/itertools.html#recipes
def pairwise(iterable):
  "s -> (s0,s1), (s1,s2), (s2, s3), ..."
  from itertools import tee
  a, b = tee(iterable)
  next(b, None)
  return zip(a, b)


# Uses the Haversine formula, accepting lats/longs in radians
def dist(c1, c2):
  from math import sin, cos, asin, sqrt

  def haversin(v):
    return (1 - cos(v)) / 2

  (lat1, long1), (lat2, long2) = c1, c2

  return 2 * RADIUS_EARTH * asin(sqrt(
    haversin(lat2 - lat1) +
    cos(lat1) * cos(lat2) * haversin(long2 - long1)
  ))

# Converts coordinate pair from degrees to radians
def coord_radians(c):
  from math import radians
  c1, c2 = c
  return (radians(c1), radians(c2))

# Calculates area of triange
def area(l1, l2, l3):
```

```
  from math import sqrt

  p = (l1 + l2 + l3) / 2
  return sqrt(p * (p - l1) * (p - l2) * (p - l3))



if __name__ == "__main__":
  # Convert the degrees to radians
  rplaces = {
    x : coord_radians(y) for (x, y) in PLACES.items()
  }

  for s in AREAS:
    if len(s) != 3:
      exit('AREAS must be sets of 3')

    print("Calculating the area of {}".format(s))

    links = list(pairwise(s))
    links.append( (s[-1], s[0]) ) # Connect last and first point

    sides = []

    for a1, a2 in links:
      print("Distance between {} and {}".format(a1, a2))

      d = dist(rplaces[a1], rplaces[a2])
      sides.append(d)

      print("\t= {}".format(d))

    print("Area between points")
    print("\t= {}".format(area(*sides)))
    print()
```

E.4 Program 2: Quad Area 1

```
# Written for Python 3.4
# Output:
# WARNING: This solution only functions correctly for concave quads
#  whose points are declared clockwise
# WARNING: This solution assumes a spherical earth that has no
#  variation from sea level
# Distance from perth to darwin = 2647.8km
# Distance from darwin to sydney = 3101.7km
# Distance from sydney to melbourne = 693.7km
# Distance from melbourne to perth = 2725.2km
# Area between ('perth', 'darwin', 'sydney', 'melbourne') = 4603815.4km^2

PLACES_DEG = {
  'perth': (-31.9688836,115.9313409),
  'darwin': (-12.5948609,131.0078759),
  'sydney': (-33.7969235,150.9224326),
  'melbourne': (-37.8602828,145.079616),
}

EARTH_RADIUS = 6371

DISTANCE_BETWEEN = [
  ('perth', 'darwin'),
  ('darwin', 'sydney'),
  ('sydney', 'melbourne'),
  ('melbourne', 'perth'),
]

AREA_BETWEEN = [
  ('perth', 'darwin', 'sydney', 'melbourne'),
]


def distance(c1, c2):
  from math import cos, sqrt, asin

  def haversin(v):
    return (1 - cos(v))/2

  la1, lo1 = c1
  la2, lo2 = c2

  return 2 * EARTH_RADIUS * asin(sqrt(haversin(la2 - la1) +
    cos(la1) * cos(la2) * haversin(lo2 - lo1)))

# Source; http://math.stackexchange.com/questions/9819/
#    area-of-a-spherical-triangle
def area_triangle(c1, c2, c3): # Must be presented as clockwise triangle
  from math import sqrt, cos, sin, tan, atan, acos

  def to_3vector(c):
    lat, lng = c
    return (
      cos(lat) * cos(lng),
      cos(lat) * sin(lng),
      sin(lat)
```

```
    )

  def dot(a, b):
      return a[0]*b[0] + a[1]*b[1] + a[2]*b[2]

  def mag(a):
      return sqrt( a[0]**2 + a[1]**2 + a[2]**2 )

  def ang(a, b):
    return acos( (dot(a,b)/(mag(a)*mag(b))) )

  A = to_3vector(c1)
  B = to_3vector(c2)
  C = to_3vector(c3)

  a = ang(B, C)
  b = ang(C, A)
  c = ang(A, B)

  s = (a + b + c)/2.0

  E = 4.0 * atan(sqrt( tan(s/2.0) * tan((s-a)/2.0) *
    tan((s-b)/2.0) * tan((s-c)/2.0) ))
  area = E * EARTH_RADIUS**2

  return area


# Source; http://stackoverflow.com/questions/12239876/
#     fastest-way-of-converting-a-quad-to-a-triangle-strip
def area_quad(A, B, C, D):
  AC_len = distance(A, C)
  BD_len = distance(B, D)

  if AC_len < BD_len:
    return area_triangle(A, B, C) + area_triangle(A, C, D)
  else:
    return area_triangle(A, B, D) + area_triangle(D, B, C)


if __name__ == "__main__":
  from math import radians
  print("WARNING: This solution only functions correctly for "+
    "concave quads whose points are declared clockwise")
  print("WARNING: This solution assumes a spherical earth that "+
    "has no variation from sea level")

  PLACES_RAD = {n : (radians(a), radians(b))
    for n, (a, b) in PLACES_DEG.items()}

  for a, b in DISTANCE_BETWEEN:
    print("Distance from {} to {} = {:.1f}km"
      .format(a, b, distance(PLACES_RAD[a], PLACES_RAD[b])))

  for c in AREA_BETWEEN:
    coords = [PLACES_RAD[n] for n in c]
    print("Area between {} = {:.1f}km^2"
      .format(c, area_quad(*coords)))
```

E.5 Program 3: Quad Area 2

```
# Written for Python 3.4
# Output:
# WARNING: Convex quads in clockwise point order must be supplied,
#   otherwise results may not be accurate.
# Distance between perth and darwin = 2647.77
# Distance between perth and sydney = 3258.13
# Distance between perth and melbourne = 2725.22
# Distance between darwin and sydney = 3101.69
# Distance between darwin and melbourne = 3136.84
# Distance between sydney and melbourne = 693.71
# ('perth', 'darwin', 'sydney', 'melbourne') area = 4493712.76
from itertools import combinations

PLACES = {
  'perth': (-31.9688836,115.9313409),
  'darwin': (-12.5948609,131.0078759),
  'sydney': (-33.7969235,150.9224326),
  'melbourne': (-37.8602828,145.079616),
}

QUAD = [
  ('perth', 'darwin', 'sydney', 'melbourne')
]

EARTH_RADIUS = 6371

def shortest_distance(c1, c2):
  from math import sin, cos, asin, sqrt

  lat1, lng1 = c1
  lat2, lng2 = c2

  def haversin(x):
    return (1 - cos(x)) / 2

  return 2 * EARTH_RADIUS * asin(sqrt(
      haversin(lat2 - lat1) + cos(lat1) * cos(lat2)
        * haversin(lng2 - lng1)
  ))

def area_triangle(c1, c2, c3):
  from math import sqrt

  a = shortest_distance(c1, c2)
  b = shortest_distance(c2, c3)
  c = shortest_distance(c3, c1)

  s = (a+b+c)/2

  return sqrt(s * (s-a) * (s-b) * (s-c))

def area_quad(a, b, c, d):
  ac_len = shortest_distance(a, c)
  bd_len = shortest_distance(b, d)

  if ac_len < bd_len:
```

```
    return area_triangle(a, b, c) + area_triangle(a, c, d)
  else:
    return area_triangle(a, b, d) + area_triangle(d, b, c)


if __name__ == "__main__":
  from math import radians
  print("WARNING: Convex quads in clockwise point order must be supplied, "+
        "otherwise results may not be accurate.")

  PLACES_RAD = {x : (radians(y1), radians(y2))
    for x, (y1, y2) in PLACES.items()}

  for q in QUAD:
    for a, b in combinations(q, 2):
      print("Distance between {} and {} = {:.2f}"
        .format(a, b, shortest_distance(PLACES_RAD[a], PLACES_RAD[b])))

    areas = [PLACES_RAD[x] for x in q]
    print("{} area = {:.2f}".format(q, area_quad(*areas)))
```