Moneyware: Simulating Software Portfolio Quality Management

Robert David Beverly

Western Kentucky University, robert.beverly879@topper.wku.edu

Follow this and additional works at: http://digitalcommons.wku.edu/theses

Part of the Software Engineering Commons

Recommended Citation

http://digitalcommons.wku.edu/theses/1478

This Thesis is brought to you for free and open access by TopSCHOLAR®. It has been accepted for inclusion in Masters Theses & Specialist Projects by an authorized administrator of TopSCHOLAR®. For more information, please contact topscholar@wku.edu.
MONEYWARE:
SIMULATING SOFTWARE PORTFOLIO QUALITY MANAGEMENT

A Thesis
Presented to
The Faculty of the Department of Computer Science
Western Kentucky University
Bowling Green, Kentucky

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science

By
Robert David Beverly

May 2015
Dedicated to my daughter, Jordan, who continues to amaze me each and every day.

Also, I dedicate this work to Dr. Kristie Jones whose love, constant support and encouragement kept me focused throughout.
ACKNOWLEDGMENTS

I wish to express my earnest gratefulness to my thesis and research advisor, Dr. David Erbach, for guiding me towards achieving my M.S. degree. I would like to thank him for pushing me beyond what I thought was possible and trusting I would make the correct decisions. He has been guiding and motivating me from my very first day at Western Kentucky University. He has provided me with a sounding board at times as well as pulling back the reins when I became a little too ambitious. It has been a true pleasure to work under his guidance as both a student and graduate assistant. I extend my heartfelt gratitude to him for his invaluable support throughout my research.

I gratefully acknowledge Dr. Uta Ziegler for the time she has invested in reading and providing feedback on my thesis. I have enjoyed our informal meetings and discussions on my thesis topic more than I can express. I am very thankful that she without hesitation, took the time out of her busy schedule to offer suggestions and critique my research.

I would like to specially thank Dr. James Gary and the Computer Science department for providing me with a graduate assistantship. I would also like to thank all my faculty members for helping me learn the computer science skills required for this thesis and as I pursue my PhD.

I would like to thank my brother, Patrick Beverly, whose knowledge and support of C# helped propel the simulator to a new level. My daughter, parents and family have been a tremendous support to me all along my M.S studies. Without their encouragement and support this would have been much more difficult if not impossible.
CONTENTS

Introduction ..................................................................................................................1
1. The Business Context for MoneyWare.................................................................8
  1.1 Project Life-Cycle............................................................................................9
  1.2 Software Quality versus Portfolio Quality......................................................14
2. Software Tasks.......................................................................................................16
  2.1 Task Characterization ....................................................................................17
  2.2 Tasks as a Workload Stream..........................................................................23
  2.3 Final Observations before Choosing a Scheduling Policy .........................24
3. Scheduling Policies ...............................................................................................25
  3.1 Scheduling the Tasks ......................................................................................26
    3.1.1 Tasks with Resource Supply Equal to Task Demand...............................26
    3.1.2 Tasks with Resources Less than Task Demand.......................................33
    3.1.3 Tasks with Surplus Resources ..................................................................36
  3.2 Analysis ............................................................................................................38
    3.2.1 FIFO Analysis ..........................................................................................39
    3.2.2 SJF Analysis ............................................................................................40
    3.2.3 ITF Analysis ............................................................................................42
  3.3 Comparison of all Three Policies .................................................................44
4. Scheduling for Optimized Quality ......................................................................50
  4.1 Scheduling the Tasks ......................................................................................51
    4.1.1 Tasks with Equal Resources ..................................................................52
    4.1.2 Tasks with Resources Less than Task Demand.......................................54
    4.1.3 Comparison Against Baseline Results ....................................................56
  4.2 Analysis ............................................................................................................57
    4.2.1 PTS Analysis ..........................................................................................57
    4.2.2 HBP Analysis ..........................................................................................59
  4.3 Comparison of All Policies ............................................................................61
5. Case Study ............................................................................................................67
  5.1 Comparing the Scheduling Policies ...............................................................69
  5.2 Observations ....................................................................................................69
6. Conclusions and Future Direction .....................................................................71
  6.1 Conclusion .......................................................................................................72
  6.2 Future Directions ............................................................................................73
    6.2.1 Future Investigation of Scheduling Algorithms .......................................73
6.2.2 Creation of a Marketable Product .................................................................74

6.3 Lessons Learned .............................................................................................75

Appendix A: Statistical Distributions .................................................................77
Appendix B: MoneyWare Users Guide .................................................................78
Bibliography ...........................................................................................................79
LIST OF FIGURES

Figure 1.1: The Context of IS Development................................................. 2
Figure 1.2: Data flow between customers and sales and marketing staff................... 10
Figure 1.3: Data flow from sales and marketing staff to the CIO........................ 10
Figure 1.4: Data flow between CIO and development staff.................................. 12
Figure 1.5: Task flow after an iteration....................................................... 13
Figure 1.6: Domain Model........................................................................... 14
Figure 2.1: Screen Shot for MoneyWare task characterization input...................... 21
Figure 3.1: FIFO with Resource Supply Equal to Task Demand.......................... 29
Figure 3.2: SJF with Resource Supply Equal to Task Demand............................ 30
Figure 3.3: ITF with Resource Supply Equal to Task Demand............................ 32
Figure 3.4: FIFO with Resource Supply Less than Task Demand......................... 33
Figure 3.5: SJF with Resource Supply Less than Task Demand.......................... 34
Figure 3.6: ITF with Resource Supply Less than Task Demand........................... 35
Figure 3.7: FIFO with Resource Supply Greater than Task Demand..................... 36
Figure 3.8: SJF with Resource Supply Greater than Task Demand....................... 37
Figure 3.9: ITF with Resource Supply Greater than Task Demand....................... 38
Figure 3.10: Quality Ratio for FIFO, SJF and ITF........................................ 45
Figure 3.11: Comparison of Tasks Completed and Quality Ratio.......................... 45
Figure 3.12: Quality Point Comparison of FIFO, SJF and ITF............................ 47
Figure 3.13: Backlog Growth........................................................................ 48
Figure 4.1: PTS with Resources Equal to Task Demand..................................... 53
Figure 4.2: HBP with Resources Equal to Task Demand.................................... 54
Figure 4.3: PTS with Resources Less than Task Demand........................................55
Figure 4.4: HBP with Resources Less than Task Demand........................................56
Figure 4.5: Comparison of Tasks Completed and Quality Ratio..................................61
Figure 4.6: Quality Comparison of all Five Policies..................................................62
Figure 4.7: Quality Ratio Comparison......................................................................63
Figure 4.8: Quality Points Comparison.................................................................64
Figure 4.9: Backlog Growth.....................................................................................65
LIST OF TABLES

Table 2.1. Task Definition Summary .......................................................... 21
Table 2.2. Sample Task Characteristics ......................................................... 23
Table 3.1. FIFO Comparison by Iteration ....................................................... 39
Table 3.2. FIFO Comparison using average of multiple trials ....................... 40
Table 3.3. SJF Comparison by Iteration ......................................................... 41
Table 3.4. SJF Comparison using average of multiple trials ....................... 42
Table 3.5. ITF Comparison by Iteration ......................................................... 43
Table 3.6. ITF Comparison using average of multiple trials ....................... 43
Table 3.7. Quality Points Comparison ......................................................... 47
Table 4.1. HBP with Fewer Resources Compared to Baseline Results .......... 57
Table 4.2. PTS Comparison by Iteration ......................................................... 58
Table 4.3. PTS Comparison using average of 10 trials .................................. 59
Table 4.4. HBP Comparison by Iteration ......................................................... 59
Table 4.5. HBP Example Completed Task Queue ......................................... 60
Table 4.6. HBP Comparison using average of 10 trials .................................. 60
Table 4.7. Comparison of All Scheduling Policies ......................................... 62
Table 4.8. Quality Points Comparison for all Scheduling Policies ................ 64
Table 4.9. Final Rankings for all Scheduling Policies ..................................... 66
Table 5.1. STC’s Task Characteristics for 52 weeks ..................................... 68
Table 5.2. Comparison of All Policies using STC’s Parameters .................... 69
In this research we introduce MoneyWare, a simulator designed to explore and ultimately to provide guidance on simulating software portfolio quality management. The name “MoneyWare” is inspired by the movie Moneyball. It chronicled a baseball team which used more descriptive statistics to achieve a higher quality ball club with limited resources.

MoneyWare is inspired by the observation that the problem of software development is somewhat analogous. Management is faced with an incoming stream of tasks for development. The tasks vary in terms of size, priority, risk, and date needed. But, in any case, the demands come to more than the resources available to do them.

We then consider various policies which a software development manager might consider in determining which tasks to schedule for development work. Using different potential and plausible resource allocation policies, we explore long term estimates of the quality which the policy might sustain over an entire applications portfolio.

MoneyWare is able to accurately simulate at a basic level task generation, task completion and software portfolio generation. Our better scheduling policy has shown as much as a 20% increase in task completion and higher quality than any of the other simulated policies.
Introduction

“The bottleneck in software appears to be far more critical to the growth of the computer field than the bottleneck in hardware.” [4] This prescient observation was made in 1969 by J. Sutherland Frame, a well-known mathematician at Michigan State University, years before the founding of companies like Apple and Microsoft. The conundrum remains true today. The CHAOS Chronicles survey [5] has found only 29% success rate for IS projects with an average time and cost overruns of 84% and 54%, respectively. Hardware fails occasionally, but less frequently than software does. When hardware failure does occur, the consequences are usually less widespread and damaging than software failure.

The difficulties of software are a puzzle for software developers and for scholars who study the subject. But they pose a critical professional problem for individuals responsible for an organization’s IS development.

In 2013 Professor Erbach suggested to Sindhu Dharani Murthy the attempt to create a formal model which permits the exploration of the behavior of possible solutions to this problem.

Figure 1.1 below depicts the business context used as the basis for this research.
In the figure above the cloud drawing labeled business context represents the customers of a company. These customers generate proposals that are eventually received by the development team for processing. In the previous research [2] (where this figure was originally created) conducted by Sindhu Dharani Murthy and Dr. David Erbach, this represented internal customers. For our research we focus on external customers and tasks generated from these requests. Customer requests are the source of proposals received by the development team for processing.

The large rectangle labeled business operations in the figure are all activities that lead to an end product delivered to the customer. Efficient business operations help to reduce cost and improve customer satisfaction.

The shaded portion of the figure above represents managerial policy. It dictates what tasks are updated, replaced or ignored. These decisions are crucial due to the often limited amount of resources.
The active software portfolio rectangle in the figure above represents all software developed by the company. The portfolio is analyzed and a profile is generated for the software behavior. From this profile the quality of software can be determined.

**Chief Information Officer’s Role**

An organization’s IS Director or Chief Information Officer (CIO) is typically faced with an incoming stream of development requests. The resources available are unlikely to be sufficient to meet all demands made of development staff. The CIO must then reconcile the various competing needs. He or she must certainly decide what is to be developed. At the same time, decisions must be made about the priority and resources to be allocated to a project, and in the process, the quality of the output. (Of course, more resources do not guarantee better quality work. But it is a plausible first simplification.)

Dharani Murthy created a model iKriya and used it to explore task scheduling and the resulting quality of the software portfolio. While her model was mainly theoretical, our objective is to focus on moving more towards application.

**Prior Research**

This thesis is an extension to the research done by Sindhu Dharani Murthy. The goal of the research conducted by Dharani Murthy and Erbach was to create a simulator to produce a theoretical model of a company’s project load, scheduling algorithm and software portfolio. The processes used in her research were designed to produce results for a theoretical company. An excerpt from Dharani Murthy and Erbach’s unpublished paper describing their research [2]:

> “It explores the consequences of various development and maintenance policies which might be applied. These depend on the state of existing software portfolio,”
This thesis attempts to move from a purely theoretical approach to more of an applied model by incorporating data provided by SoftTrac Computing (STC) gathered during an internship in the summer 2014. Through a collaboration with STC it was determined that the simulator should account for each man-week in an iteration along with a much more narrow definition then Dharani Murthy’s of what constitutes quality in software. Using these new project attributes allows the simulator to be further refined to more closely reflect the needs of organizations which develop software for external customers.

While the previous simulator done by Murthy was coded entirely in Java, MoneyWare (the name chosen for our simulator) was coded in C#, XAML and WPF. This was done to take advantage of the inherent interface and binding abilities of the .Net framework. Specifically, a factory design pattern was implemented to allow for quick and easy integration of new scheduling algorithms. By using WPF and XAML all user interface controls can be modified with a minimum of effort. Lastly, the attributes for the tasks were mostly kept the same except for the added attribute of due date. The software quality algorithm was modified to account for the information gained from STC.

Plan of the Thesis

In this thesis I have three objectives:
i. to refine the iKriya model in various ways, bringing a new model which I call MoneyWare.

ii. to structure MoneyWare in a way which can be at least tentatively tested against actual corporate development policy.

iii. to explore what happens in the above situation.

MoneyWare assumes that management is faced with an incoming development task stream, and must make periodic decisions about what tasks to undertake, and what resources to allocate to them. The objective is to develop scheduling policies which will, in the long run, attempt to lead to the highest achievable quality over the aggregate of the organization’s software portfolio.

In Introduction we formally describe the research question, scope and methodology used to conduct the research used in producing the simulator. It is included to show the questions asked and the decisions made at the beginning of the research to produce this paper.

In Chapter 1 The Business Context for MoneyWare we discuss the problem from a business standpoint and the life cycle of a project. We finish this chapter with a discussion of software quality compared to software portfolio quality.

Chapter 2 Task Characteristic Definition sets out to formally describe the task based on the different parameters chosen. The characteristics chosen and the ones not chosen are examined in detail. We also discuss the task queue the incoming tasks are added to and how unfinished tasks are handled.
Chapter 3 *Scheduling Policies* presents three well-known scheduling policies and the reasons for using them. We create baselines using these policies and examine their respective strengths and weaknesses.

Chapter 4 *Scheduling for Optimized Quality* starts with a discussion on portfolio quality and how we can attempt to optimize a scheduling policy based on it. We present two attempts at improved scheduling policies. We compare them to the previous baselines documented in chapter 3.

Chapter 5 *Case Study* looks at what happens when we calibrate MoneyWare using data gathered from a real company. We discuss the value in dollars that could potentially be saved/lost by using the different policies.

Chapter 6 *Conclusions and Future Directions* reviews the entirety of the research, our reasoning behind conducting it and the inspiration for the name MoneyWare. We also look at the different directions future research might follow.

**Thesis Statement**

This section describes the reasoning, scope and methodology used in completing this research. It describes why the research was conducted, as well as some basic implementation details; it also describes what was and was not included in the research.

**Area of Research**

The purpose of this study is to examine the quality of simulated software portfolios created by different scheduling policies. To accomplish this, a task schedule simulator utilizing C#, XAML and WPF programming/modeling languages was implemented.
MoneyWare is a software based simulator which produces a simulated software portfolio using user defined specifications. 

In this study I will investigate and quantify multiple scheduling algorithms using a task scheduling simulator. During simulations the software will produce data in the form of a software portfolio. This data will be extracted into spreadsheets from which various graphs and tables are produced and used to analyze various parameters.

**Research Question**

How can a scheduling policy achieve maximum quality and by doing so increase the quality of the resulting software portfolio?

**Scope of the Thesis**

This thesis focuses on quantifying and analyzing the results produced using different task scheduling policies. Task scheduling is a major time-consumer in any company tasked with producing software for external customers. This research focuses only on software that is requested by customers external to the company. This is in contrast to the research done previously by Dharani Murthy and Erbach.

**Methodology**

Initially a domain and data flow diagrams were developed. The next stage of the development was generating C# classes based on these models. After completion of this step, a user interface was developed allowing the user to dynamically choose their distribution parameters. The last step was developing a method to present the results. This is accomplished by saving all results as a Microsoft Excel spreadsheet. From this spreadsheet multiple graphs and tables are produced along with access to the raw data.
Chapter 1 The Business Context for MoneyWare

An organization’s Chief Information Officer is responsible for delivering the highest quality software with the greatest return on investment (ROI). While this person does not produce the software, his/her ability to schedule the projects and allocate the resources is the biggest influence on the quality of software produced. In the worst case scenario, the loss in revenue caused by either poor quality software or software that never meets a deadline has the added detriment of causing decreasing morale in the technical staff.

For companies that specialize in custom software this first step poses a problem, how to evaluate projects from results of previous projects if there has not been enough data to compare. Finally, the CIO must accurately analyze the software portfolio to establish whether the chosen scheduling policy and resources used created high quality software. Not only does the CIO have to analyze the software portfolio but they must determine what constitutes high quality software.

For this research, high quality software can be described as software that is completed on time and with the required amount of resources which may be adjusted based on its risk level and priority. We can formally define quality and quality ratio as such:

- The quality-ratio measure of a completed task is determined by its priority. A quality weight factor is given to a task based on priority. If the task is completed on time and with the required resources it will receive the quality ratio equivalent to the quality weight factor. These weight factors are:
  - High priority = 1.5 quality weight factor
- Medium priority = 1.0 quality weight factor
- Low priority = 0.5 quality weight factor

- Since low priority tasks reduce the number of resources used, the quality-ratio for completing low priority tasks can be less than 1.0. This has the effect of punishing policies that favor completing lots of low priority tasks. These quality figures, through a weighted calculation, eventually form the portfolio’s aggregate quality estimate, as detailed below.

- The sum of weighted quality-ratio times the size of the completed tasks is designated as the estimate of the overall software portfolio quality.

  - QualityPoints = QualityRatio x Size

This simulator produces a software portfolio using user selected project attributes and resources. This allows the CIO to plausible estimate the software quality expected using a given number of resources and their chosen scheduling policy.

### 1.1 Project Life-Cycle

A good understanding of the project scheduling life-cycle is helpful for evaluating and researching quality-based scheduling policies. A project starts with the customer; for our purposes this is external customers only, making a request for enhancement of a previous task or the need for a new task. The simulator does not need to distinguish between enhancements or new tasks, although this is a very real issue for most companies. We can assume that the size, risk and priority of the task are independent of whether it is new or needing enhancement, therefore we do not need to specify it for the simulator. For the purpose of this research we are simulating companies that have a
dedicated software development team and produce software for external customers.

Figure 1.2 below demonstrates the data flow between customers and the corporate line staff.

![Diagram](image1.png)

**Figure 1.2 Data flow between customers and sales and marketing staff**

These task requests are handled by the sales and marketing staff. The sales and marketing staff works with the customer to determine the attributes of the prospective tasks.

The sales and marketing staff are in direct contact with the customer and are therefore the ones most likely to influence a number of attributes that directly affect the quality of software from the customers view. Changing due dates and adding requirements are a couple of issues the sales and marketing staff must address if software quality is to be maintained.

![Diagram](image2.png)

**Figure 1.3 Data flow from sales and marketing staff to the CIO**

The CIO may also be known as the IS Director or by any number of titles. For our purposes he/she is essentially the person in charge of scheduling all tasks and keeping
track of progress, quality and backlogs of tasks. The CIO uses his/her prior experience to properly schedule all the tasks to produce the highest quality ratios. The CIO receives all incoming tasks and decides on what scheduling policy to follow and how many resources to commit. The tasks are then prioritized and sent to the appropriate technical staff i.e. programmers and engineers. For example, if a task has a low priority and isn’t due for several weeks then it can be moved further down the priority queue. However, how does the CIO accommodate multiple similar tasks with limited resources? Relying on past experience will generally produce similar results to the past; this is not always an advantageous proposition. Using the simulator the CIO can attempt to determine if a different scheduling policy might produce better results. With the use of the simulator the CIO will be able to investigate the effects of adding more resources to a specific task scheduling policy.

The technical staff is directly responsible for the quality of software produced. We make the assumption that if enough resources are applied then high quality software will be achieved.

For this research project it is assumed the user of MoneyWare will determine the amount of resources available per simulation. This allows the simulator to apply the resources independently of knowing how that number was achieved. The development staff is also responsible for the risk attribute applied to the tasks. Risk is part of every project but is minimized with a seasoned staff that can accurately estimate task complexity. It is up to the user to estimate the ability of the staff when selecting this attribute in the simulator. An inexperienced staff may incur a substantially higher risk than a veteran staff and this
possibility should be accounted for. Figure 1.4 below illustrates the data flow between the CIO and technical staff.

Figure 1.4 Data flow between CIO and development staff

After the tasks have been completed the CIO will either sign-off as completed and deliver back to the corporate line staff for release to customers or return them into the project queue for further refinement. High quality software can be considered as any software that is completed on or before the date needed using the resources required. Figure 1.5 depicts the flow of software either completed and released to the customer or returned to the queue for further work. Tasks that have not completed at the end of an iteration are returned to the queue and ordered along with any incoming new tasks.
Task Scheduling

In MoneyWare, the user takes on the role of the CIO allowing the user to adjust the parameters for and run multiple simulations using the available task characteristics such as: number of iterations, number of projects, resources available, date needed of task, size of task, risk of task and priority of task.

Runs with varied parameters allow the user to evaluate their current scheduling policy versus any number of other scheduling policies. Another benefit is the ability to see the effect of adding or removing resources as well as of changing the number of projects being worked on at any one time. For many managers, task scheduling is not the most
attractive part of their work. By providing them with a simple simulation tool, they can more easily adjust their scheduling policy to best fit their company’s current needs. As resources become available or leave then the tasks can be simulated again with different policies to check for effectiveness.

Below in Figure 1.6 is a diagram depicting the entire domain of this research.

![Figure 1.6 Domain Model](image)

### 1.2 Software Quality versus Portfolio Quality

When most people discuss quality with respect to software they are discussing the number of bugs, how tightly the classes are coupled and so on. This is commonly referred to as software structural quality. Software structural quality refers to how it
meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability.

What we are interested in and what this research focuses on is the quality of the portfolio as analyzed by the CIO. This measure of software portfolio quality assumes the structural quality is a factor of several task characteristics along with the policies used to allocate the resources and order the tasks. Each completed task will have a quality but can different scheduling policies increase or decrease the aggregate quality of all tasks? We can assume for the sake of the research that if an adequate number of resources are allocated then software structural quality will be at an acceptable level. There are numerous tools available to check for structural quality such as: rcov, jcov and EMMA. But to the best of my knowledge only Dharani Murthy’s prior research (iKriya) and this work attempt to quantify portfolio quality. For these reasons MoneyWare was developed to examine different scheduling policies to examine their effectiveness on the software portfolio quality.
Chapter 2 Software Tasks

This chapter discusses how we define a task: what characteristics are used and why? Which ones were not chosen and why? Given how we define our tasks we then look at how they will appear as an incoming task stream for the CIO.

It is important to clarify the term “task” at this point. We use the term task to mean any stand-alone software request. This is also referred to as a “project” in some companies. From early on hardware was expected to be more complex than software. As time has passed the opposite has proven true. The complexity in software is not in the minute details of code i.e. more efficient algorithms, better design patterns and so forth, but in the money/time cost in task definition, scheduling, development and maintenance.

Before we can examine the efficiency of various development scheduling policies we must first specify what a task consists of. This brings forth the question of what is a software task.

A general characterization of a task is as follows:

- A task is designed to produce specific results.
- A task has a beginning and end (time-constrained and funding constrained)
- A task is typically proposed to bring about beneficial change or added value.

To be able to model task scheduling and portfolio quality we must first examine what characteristics tasks have in general. Then we must choose a task model that aspires to simplicity without sacrificing accuracy, or as close as we can come to it.

Section 2.1 Task Characterization, discusses a range of characteristics used to define a generic simulated task. We discuss the characteristics common for most tasks, which ones we chose and which ones we omitted.
Section 2.2 *Tasks as a Workload Stream* describes the distributions used for the simulator. We explain the need for each and how the parameters for the different distributions are used to create realistic tasks. Statistical definitions and formulas for the distributions in this chapter, are included in Appendix A.

Section 2.3 *Final Observations before Choosing a Scheduling Policy* reviews what factors we considered when choosing the task characteristics for MoneyWare. We also discuss what happens to the tasks once they are generated.

### 2.1 Task Characterization

Our underlying objective is to model an incoming stream of tasks, so our first problem is to start with a simplified formal definition a software task. Software tasks typically have some or all of the following characteristics:

- **Size**: typically specified in developer time. For our purposes we chose weeks.
- **Priority**: using a rough scale of high, medium or low
- **Risk**: using a rough scale of high, medium or low
- **Date needed**: specified in future time. For our simulator date needed is specified in weeks from the date of proposal
- **Quality Assessment**: the quality-ratio measure of a completed task is determined by its priority. A quality weight factor is given to a task based on priority. If the task is completed on time and with the required resources it will receive the quality ratio equivalent to the quality weight factor. These weight factors are:
  - High priority = 1.5 quality factor
  - Medium priority = 1.0 quality factor
We also calculate the completed tasks contribution to the overall quality by multiplying the quality ratio by its original size.

- \( \text{QualityPoints} = \text{QualityRatio} \times \text{Size} \)

- Return on Investment (ROI); estimation or other representation of an expected monetary value
- Sequencing; how one task’s completion affects others
- Task state; consisting of finished, started but incomplete, or not yet started
- Time in queue; typically specified by time since added to queue

Selected characteristics and their definitions for the purpose of our research:

- Size- Specifically chosen as it will be the main factor for the CIO in deciding the ultimate cost to the customer. Cost to the customer is the amount of time taken times the hourly rate. While this is not modeled in the simulator it is mentioned here to denote why size is on the top of our list of characteristics. As we will see later, the size is not the only factor that determines the task’s time to completion but it is the most important. Our standard task uses a mean of 10 developer weeks size and a standard deviation of 2. This value is able to be adjusted by the user when operating the simulator.

We generate the size using samples from a Normal (or Gaussian) distribution. Choosing Gaussian distribution for size allows us to change the range of outputs by changing the variance or mean. If the data that we are simulating needs to have a wide spread, we can increase the variance to better match this quality.
This proves especially useful for companies that have tasks with a large variance in task size.

- **Priority** - This characteristic can have many different meanings depending on who is being asked. Customers may all believe their proposal should be top priority. For our research we look at priority from the CIO’s standpoint as a measure of how important it is in relation to the other tasks involved. Priority is described as low, medium or high level.

  For the simulation base case we chose to use a 40% low, 40% medium and 20% high distribution. The simulator translates these figures into a decimal value of 0.5 for low, 1.0 for medium or 1.5 for high which is then multiplied by the size to give us a needed resources value. This has the effect of adding or subtracting the resources needed for a particular task based on how important it is to the company.

- **Risk** - This characteristic describes how much is known about the task beforehand, as well as being a proxy for other things. For example, a task that uses hardware that is unknown or outside libraries that are new would be considered a risky task and consequently will have a higher risk characteristic. This characteristic is described as either low, medium or high and reflects the perceived level of difficulty the task poses.

  The distribution chosen for risk was 30% low, 50% medium and 20% high. The simulator translates these figures into a decimal value of 1.0 for low, 1.25 for medium or 1.5 for high which is then multiplied by the size. This value is multiplied by the priority adjusted size to form the adjusted size of the task. This
has the effect of potentially adding additional resources needed for a particular task based on how risky it is to the company.

- **Date needed** - This characteristic describes the time from when the task has been generated to when the customer specifies they need it. While the size of the task has not changed, an early date needed will require more resources to be used to achieve an on-time delivery. This can cause a lack of resources to be allocated if not accounted for in resource allocation. This can help the simulator determine the actual number of tasks that can be completed at a given quality level.

  Our standard task uses a mean of 7 weeks for date-needed and a standard deviation of 2. This value is able to be adjusted by the user when operating the simulator. We generate the date needed using a Gaussian distribution.

- **Task state** - Tasks can be considered to be in one of three states; completed, started but not finished and not started. For our analysis we are not adjusting priority based on age, so the knowledge of a task’s state is not beneficial in that respect. For tasks that are not completed in an iteration, their resources needed will have changed by the amount accomplished so far and will potentially affect their placement in the next priority queue. This is accomplished without needing to know if the task was previously started.

- **Quality Assessment** - We assign a completed task a quality-ratio of 1.0 if it is finished on or before its date needed provided it has been allocated its base resource requirements estimate. Base resource requirements are determined based off of the tasks size, priority and risk. This number could be greater or less than the size depending on the priority and risk level. Depending on policy
decisions regarding organizational priority, a task may have up to an extra 50% resource allocation. If completed promptly in this way, it will achieve a quality-ratio up to 1.5, corresponding to the resources consumed. A task which is not completed by its date needed receives a quality-ratio corresponding to the amount completed. These quality figures, through a weighted calculation, eventually form the portfolio’s aggregate quality estimate. The weighted sum of quality-ratio of the completed tasks is designated as the estimate of the overall software portfolio quality.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Distribution</th>
<th>Default Value</th>
<th>Range</th>
<th>User Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Normal</td>
<td>10 weeks</td>
<td>1-50 weeks</td>
<td>Yes</td>
</tr>
<tr>
<td>Priority</td>
<td>Piecewise Constant</td>
<td>40%, 40%, 20%</td>
<td>Low, Medium, High</td>
<td>Yes</td>
</tr>
<tr>
<td>Risk</td>
<td>Piecewise Constant</td>
<td>30%, 50%, 20%</td>
<td>Low, Medium, High</td>
<td>Yes</td>
</tr>
<tr>
<td>Date needed</td>
<td>Normal</td>
<td>7 weeks</td>
<td>1-50 weeks</td>
<td>Yes</td>
</tr>
<tr>
<td>Task State</td>
<td>N/A</td>
<td>Variable</td>
<td>*</td>
<td>No</td>
</tr>
<tr>
<td>Quality Assessment</td>
<td>N/A</td>
<td>Calculated Value</td>
<td>0-1.5</td>
<td>No</td>
</tr>
</tbody>
</table>

* tasks can be in one of three states: completed, started but not finished or not started

Table 2.1 Task Definition Summary

Figure 2.1 below demonstrates the task characteristic inputs available to the user of MoneyWare.

Figure 2.1 Screen Shot for MoneyWare task characteristic input
The following characteristics were not chosen:

- **ROI** - describes the expected value of the task to the company. Using the formula below we see the need for more parameters, adding to the complexity but not affecting the accuracy of the simulator.

\[ ROI = \frac{\text{Gain from Investment} - \text{Cost from Investment}}{\text{Cost of Investment}} \]

This characteristic was initially included but proved too difficult to accurately convey a standardized meaning. ROI tends to be evaluated differently by various companies and in any case can be more considered a result of completion of a task instead of a characteristic.

- **Sequencing** - this characteristic is crucial for most software development but after careful consideration was not chosen for our task generation. For our purpose a task is considered the entire project consisting of all smaller tasks needed for completion. This characteristic is typically seen in PERT charts and in determining correct order of tasks. While this is an important characteristic and one that would be needed eventually, it was not included at this time.

- **Time in queue** - this characteristic describes how long the task has been queued. This characteristic was the most debatable for inclusion as a task characteristic. This characteristic can be used to adjust priority based on the amount of time a task has been sitting in a queue. To avoid an additional level of complexity, it was decided to leave this characteristic out.

We chose to adhere to the saying “As simple as possible but no simpler”. For this reason we had to choose carefully what to include and what to exclude. The characteristics chosen are:
- Size
- Priority
- Risk
- Date needed
- Quality assessment
- Task state

Table 2.2 shows a sample task list generated using the default parameters.

<table>
<thead>
<tr>
<th>Size</th>
<th>Risk Level</th>
<th>Risk</th>
<th>Priority Level</th>
<th>Priority</th>
<th>Adjusted Size</th>
<th>QualityRatio</th>
<th>DateNeeded</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Low</td>
<td>1</td>
<td>High</td>
<td>1.5</td>
<td>12</td>
<td>1.5</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>Low</td>
<td>1</td>
<td>Med</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>Med</td>
<td>1.25</td>
<td>Low</td>
<td>0.5</td>
<td>7.5</td>
<td>0.5</td>
<td>7</td>
</tr>
<tr>
<td>14</td>
<td>Low</td>
<td>1</td>
<td>Low</td>
<td>0.5</td>
<td>7</td>
<td>0.5</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>Low</td>
<td>1</td>
<td>Low</td>
<td>0.5</td>
<td>4.5</td>
<td>0.5</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>Med</td>
<td>1.25</td>
<td>Low</td>
<td>0.5</td>
<td>5.63</td>
<td>0.5</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>Med</td>
<td>1.25</td>
<td>Med</td>
<td>1</td>
<td>12.5</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>Med</td>
<td>1.25</td>
<td>Med</td>
<td>1</td>
<td>16.25</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>High</td>
<td>1.5</td>
<td>Med</td>
<td>1</td>
<td>12</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>Med</td>
<td>1.25</td>
<td>High</td>
<td>1.5</td>
<td>16.88</td>
<td>1.5</td>
<td>14</td>
</tr>
</tbody>
</table>

Table 2.2 Sample Task Characteristics

2.2 Tasks as a Workload Stream

Having determined how to properly define our tasks, how will they appear as an incoming stream for the CIO and the technical development staff?

The number of tasks which arrive in a sprint of four weeks is modeled in MoneyWare as a variable number of tasks with Poisson distribution and mean \( k \) per iteration. Here \( k \) is a parameter set by the user.

For the standard task stream \( k = 4 \). As the rate \( (k) \) becomes higher (as the occurrence of the thing we are watching becomes more common), the center of the curve moves
toward the right. Poisson is chosen because, while it may not be common for no new
tasks to appear for an iteration, it still has the possibility to happen.

Tasks that are not completed in one iteration are then placed back into the backlog. The CIO can check the backlog queue after each iteration to see what has completed and what tasks are left for the next iteration. Incomplete tasks that are placed in the backlog queue will now reflect the amount of work that has been done in previous iterations.

2.3 Final Observations before Choosing a Scheduling Policy

Ideally, the user will examine their previous projects and extract data for all relevant tasks and perform a statistical analysis to determine the parameters to be used when creating tasks. With this knowledge the parameters can be changed to match a given company’s data. This technique was applied using data gathered at STC to obtain the parameters used in testing. In chapter 5 we examine data gathered from a real company and adjust MoneyWare to match. We then analyze the output from MoneyWare and the real data for consistency.

Now that we have a specific method for creating tasks they must be stored somewhere. As tasks are created they are added to an unsorted queue. We refer to this queue as the backlog of tasks. This queue will include all tasks that are newly created along with any task that was not completed during the previous iteration and has not passed its date-needed yet. The CIO will examine the backlog queue and determine what scheduling policy to use to order the tasks for the development personnel.
Chapter 3 Scheduling Policies

An IS Director has various ways to choose which tasks to undertake and what level of resources to allocate to them. Tasks will most likely be high priority to the customer but it is up to the IS Director to determine the task’s priority as it relates to other competing tasks in the queue. Priorities are often set by the IS Director whose job is to reconcile competing needs, and to help line divisions understand that not everyone can have everything they might want. But IS Directors are inevitably subject to the influences of the political power and charisma of the customers. Their choices may not correspond to any optimization among the possible alternatives [1].

Given the task stream and resources, how does the simulation handle such issues as scheduling sequence and priority assignment? The resources available are unlikely to be sufficient to meet all demands made of development staff. A IS Director must then reconcile the various competing needs. He or she must certainly decide what is to be developed and in what order. At the same time, decisions must be made about the priority and resources to be allocated to a project, and in the process, the quality of the output. This chapter examines some basic scheduling policies and establishes a baseline for task completion and portfolio quality.

In order to study the prospective benefits of various policies intended to optimize quality, it is helpful to develop a baseline which reflects the consequences of common approaches. In the case of MoneyWare, we begin the examination with three basic policies: First-in, first-out (FIFO), Shortest Job First (SJF) and Important Task First (ITF). For the purpose of examining each policy compared to the others, all parameters were kept the same and multiple runs used while averaging the results.
Section 3.1 *Scheduling the Tasks* looks at the three different policies using different resources available, with all other parameters kept the same. Each subsection describes a specific policy’s results for the specified resources.

Section 3.2 *Analysis* compares the three different scheduling policies and how they performed as resources were added or removed. We take note of the short-comings of each policy and how this continues to lead to bottlenecks in software development. Finally, we address the issue of the reality of trying to convince IS managers that change can be beneficial.

Section 3.3 *Comparison of All Three Policies* looks at the three policies using fewer resources than needed. We establish baselines which we will use in the next chapter to compare the “better” policies we attempted.

### 3.1 Scheduling the Tasks

In practice, many scheduling policies may be implemented. For this analysis three common policies are chosen. Next chapter we explore two attempts at a “better” solution.

The Shortest-Job-First and First-In-First-Out algorithms were chosen to provide a benchmark to compare the other algorithms against the quality, backlog and completed tasks achieved by the benchmarks. The last basic policy and one commonly chosen by companies is the Important-Task-First algorithm.

#### 3.1.1 Tasks with Resource Supply Equal to Task Demand

We first examine the behavior of the different scheduling policies with available resources equal to the expected number of resources needed. The resources available
value is set slightly higher than the expected resources needed because some tasks require more resources than just their size would indicate. By setting this value slightly higher we are accounting for the adjusted size of the tasks. Running the simulator for “equal resources” provides a basis for the amount of degradation that occurs when resources are removed. It also can allow the CIO to test different scheduling policies based off of technical staff present or technical staff needed.

**First-In-First-Out Policy (FIFO)**

Tasks are always undertaken in the order in which they are requested. This has the appearance of fairness and neutrality but it means that tasks are undertaken without comparison of their relative value to the organization. The results are not likely to be optimal, as they respect history, even though the context may change. This policy does, however, demonstrate the expected completed number of tasks over time, and at what quality level, when no special judgment is applied. As such, the approach also provides one base-line evaluation of the sustainable quality of the portfolio.

FIFO is the most basic algorithm and the one with the least potential to produce high quality software. If a time-consuming task with low priority and high risk comes into the queue early then this can lead to a starvation issue. Starvation occurs when larger tasks continually use up all available resources preventing some tasks from ever completing. This task will consume many of the resources but will not contribute much to the overall portfolio quality given its low priority level. Nonetheless, this algorithm provides a benchmark with known limitations.

Figure 3.1 shows the results of choosing FIFO with sufficient resources for the expected amount of tasks in queue.
Standard Task Parameter List

For this and all future trial runs we designate a standard parameter list below. This list is kept constant with the only exception of resources available. This single parameter changes for the different trials and is noted when changed. This number was derived experimentally after many trials, to match resources available with expected resources needed.

The simulator was run using the standard parameters of:

- Number of tasks: mean of 4
- Number of iterations: 13
- Resources available: 11 (value chosen based on adjusted resources needed)
- Date needed: mean of 7
- Weeks in Iteration: 4
- Time/Size: mean of 10
- Risk: 40% Low, 40% Medium, 20% High
- Priority: 30% Low, 50% Medium, 20% High
Examining the figure above shows no unexpected results. We do notice the jumps in backlog which tend to occur at the start due to having abundant resources and no backlog. After the first iteration the backlog and tasks completed stable out. With resources available roughly matching the amount needed, the backlog will not grow and task completion should be fairly steady. We do see some peaks and valleys in quality ratio which is expected due to tasks arriving and being scheduled in no particular order. This will still allow large tasks to starve small tasks in some iterations. These starved out tasks are usually completed in later iterations, as the lack of a growing backlog indicates.

**Shortest-Job-First Policy (SJF)**

A second simple policy which gives a baseline result, is to order queued tasks in terms of size. Resources are then allocated to the smallest, then the next largest, until the available resources have been exhausted. This approach allows the IS Director and staff
to demonstrate a larger number of completed requests. However, it does not maximize the value to the organization of the work done. The fundamental problem is that it allows the quick to displace the important.

The Shortest-Job-First Algorithm (SJF) sorts the tasks according to the length remaining until it is complete. This allows for a task that has been partially completed in one iteration to be slotted into the position that it should occupy with its remaining length.

![Graph showing SJF with Resource Supply Equal to Task Demand](image)

**Figure 3.2 SJF with Resource Supply Equal to Task Demand (average of 10 trials)**

Figure 3.2 shows the results of choosing SJF with sufficient resources for the amount of tasks in queue. The simulator was run using our standard parameters listed above. One potential problem with Shortest Job First is if resources scarce, then longer tasks may never execute. In simulation this produces high numbers of tasks completed, but low quality numbers due to few high priority tasks being completed. Comparing SJF to
FIFO shows that SJF completes a little higher average number of tasks (.1 task per iteration more) but does so with a slightly lower quality ratio (.1%). With resources available equal to resources needed this is only a small difference. Businesses require that high priority tasks be executed in a timely fashion which this policy does not take into account. If an optimized solution can complete the same number of tasks as SJF and satisfy the business requirements of selecting higher priority tasks first then it truly is a better solution.

With resources available equal to resources needed, SJF performs fairly well. Even though it eschews priority in favor of size, with enough resources most if not all tasks are completed each iteration. Iterations that contain several large tasks will still cause a drop in quality but are completed in later iterations, causing a few fluctuations in quality ratio.

**Important-Task-First Policy (ITF)**

With this policy, we order tasks by their designated organizational priority, and undertake them in that order. This has the advantage that the IS Director can report the completion of tasks which are deemed the most important. Tasks with a high priority may be allocated additional resources in order to be completed with high quality. A risk is that some quick and easy jobs may not be done.

ITF sorts the tasks according to the priority attribute created when the task is created. By choosing this algorithm, the CIO is following a known pattern that produces an adequate level of software quality. ITF does not take into account due date or size, it allows some tasks to starve others reducing overall software quality. This policy, while following plausible business logic, fails to complete as many tasks as SJF.
It can be argued, that in the business context that this is a trade-off that must occur. Unfortunately it completes noticeably fewer tasks than SJF and FIFO. Task completion dropped by 3 tasks per year, while quality increased by 15%. This occurs due to ITF always completing high priority tasks first. With enough resources ITF starts to complete less important and thus lower priority tasks. Important tasks require more resources to guarantee their completion. As such, ITF will schedule tasks that require more resources at the first of the queue. This causes less tasks to be completed but the ones that are have a high quality-ratio.

Figure 3.3 shows the results of choosing ITF with sufficient resources for the amount of tasks in queue. The simulator was run using our standard listed above. Our long term goal is to find a solution that combines the tasks completed of SJF and the priority completed of ITF policies. The quality-ratio is noticeably higher using ITF than
the two previous policies. While this may look good, the number of tasks completed per iteration is much lower, consequently the backlog continues to grow.

3.1.2 Tasks with Resources Less than Task Demand

We now examine the results produced using 20% fewer resources than needed. This allows us to examine how each of the policies react when resources are at a premium. This allows us to see the growth of task backlog, task completion rate and quality level. Figures 3.4, 3.5 and 3.6 show the results of the simulator for each of the scheduling policies FIFO, SJF, and ITF using our standard parameters described above, except that the resources available are reduced to 9.

![Resource Supply Less than Task Demand](image)

Figure 3.4 FIFO with Resource Supply Less than Task Demand (average of 10 trials)

From the figure above we notice quality ratio has dropped considerably from .90 to under .80 with 20% fewer resources when compared to equal resources above. The task
backlog is growing quickly (ending with almost 13 tasks in the backlog), while tasks completed per iteration is down about 25%. This is not unexpected once resources have been limited. Of note here is using an average of 10 trials we still get a task completed graph with lots of peaks and valleys. This is unavoidable using the FIFO policy as it is completely dependent on the random order of task arrival.

First-in, first-out development tends to result in a build-up of task backlog, although not as much as ITF. This build-up has the potential to starve out smaller tasks.

Figure 3.5 SJF with Resource Supply Less than Task Demand (average of 10 trials)

SJF appears to be a good choice at first glance. There is still a large number of tasks completed per trial (averaging 41 total), barely completing fewer than when resources were abundant. Backlog is greater than earlier but less than other policies. These are the strengths of SJF, but its weakness is the lack of quality it achieves. Quality has dropped by roughly 20%. This represents the lowest quality ratio of any policy tested. Quality
drops significantly due to SJF ordering the queue based on size. Size is partially
determined by priority, low priority tasks decrease the amount of resources needed, thus
SJF completes more low priority tasks and ones that achieve a lower quality ratio.

Figure 3.6 ITF with Resource Supply Less than Task Demand (average of 10 trials)
An interesting occurrence appears when resources available are limited using ITF. What
appears as an anomaly can be explained quite easily. From figure 3.3 above we see that
quality ratio has actually improved even though we have fewer resources available. This
happens because with resources limited ITF only completes important tasks. This means
tasks which are low priority are not completed and therefore do not factor into the
quality ratio. This would be similar to a company only completing a few tasks but
making sure they are completed extremely well.
3.1.3 Tasks with Surplus Resources

Finally, we consider what happens when there are abundant resources for the amount of tasks present. This is mainly done to show how the task completion and quality ratio change as resources are removed/added. To further simulate how a company may deal with excess resources, the simulator checks the resources needed versus the resources available at the start of every iteration. If there is a surplus of resources greater than 5, then these resources are applied to any task with a medium or less priority. This simulates technical staff that are not currently working on a task (the surplus of resources), adding extra help to unfinished tasks. This has the effect of raising the quality ratio when we have a surplus of resources. Figures 3.7, 3.8 and 3.9 show the results of the simulator for each of the scheduling policies using our standard parameters described earlier, with the exception of resources available increased to 12. This number is roughly a 10% surplus in resources available.

Figure 3.7 FIFO with Resource Supply Greater than Task Demand (average of 10 trials)
We see a similar task completion and backlog to figure 3.3 above; we do see quality improve slightly as the extra resources available in some iterations are applied to lower priority tasks. This has the effect of raising the quality for those tasks and hence a slight rise in quality ratio. A 10% surplus in resources available over what is needed allows most iterations to complete all tasks. The backlog is for the most part non-existent and averaged less than one task at any given point.

Figure 3.8 SJF with Resource Supply Greater than Task Demand (average of 10 trials)

With resources abundant SJF matches the other policies in task completion and quality. SJF’s quality ratio is dependent on resources available where task completion stays consistent throughout.
We see a similar result with ITF as with the other two policies. With plenty of resources quality tends to average the same between the three policies. This is an obvious occurrence since they are completing all the tasks with enough resources to ensure high quality.

3.2 Analysis

With the results from the above sections we can now establish a baseline on which to compare our “better” scheduling policies. Our “better” policies need to match or exceed SJF in terms of task completion but follow the business guidelines of executing higher priority tasks earlier. As Richard Feynman remarked “I learned that innovation is a very difficult thing in the real world”. [3] He was referring to how difficult it is to get people to change their habitual views and methodologies. Such is the case with convincing
management that there may be a better way to schedule tasks. Giving them numeric proof with the quality metric we have created can only help in this pursuit.

### 3.2.1 FIFO Analysis

Using table 3.1 we can start to examine the pitfalls of using a FIFO scheduling policy. The most noticeable issue with FIFO is its unpredictability. Depending on when the tasks come in there may be large, low priority tasks starving all the others. This will lead to low quality for that iteration. This shows up in Figure 3.4 as a spike either up or down. The task completed growth is completely dependent on the order in which tasks come in. This lead to a large discrepancy in tasks completed, backlog and quality between each trial run. If a large task comes in first it will completely starve all others resulting in few if any tasks completed that iteration. As expected the quality ratio is lower due to fewer tasks being completed and the ones that are completed having no guarantee of higher quality. Unfortunately there are companies scheduling tasks this way and with fewer resources available.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Total Tasks Completed</th>
<th>Tasks Completed/Iteration</th>
<th>Backlog</th>
<th>Quality Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.8</td>
<td>2.8</td>
<td>1.2</td>
<td>0.8</td>
</tr>
<tr>
<td>2</td>
<td>5.4</td>
<td>2.6</td>
<td>2.5</td>
<td>0.78</td>
</tr>
<tr>
<td>3</td>
<td>8.5</td>
<td>3.2</td>
<td>3.4</td>
<td>0.82</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>3.4</td>
<td>4</td>
<td>0.81</td>
</tr>
<tr>
<td>5</td>
<td>14.5</td>
<td>2.6</td>
<td>5.4</td>
<td>0.82</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>3.4</td>
<td>6</td>
<td>0.79</td>
</tr>
<tr>
<td>7</td>
<td>21.2</td>
<td>3.2</td>
<td>6.8</td>
<td>0.8</td>
</tr>
<tr>
<td>8</td>
<td>24</td>
<td>2.8</td>
<td>8</td>
<td>0.8</td>
</tr>
<tr>
<td>9</td>
<td>27.6</td>
<td>3.6</td>
<td>8.4</td>
<td>0.79</td>
</tr>
<tr>
<td>10</td>
<td>30.8</td>
<td>3</td>
<td>9.4</td>
<td>0.81</td>
</tr>
<tr>
<td>11</td>
<td>34</td>
<td>3.4</td>
<td>10</td>
<td>0.8</td>
</tr>
<tr>
<td>12</td>
<td>36.5</td>
<td>2.6</td>
<td>11.4</td>
<td>0.78</td>
</tr>
<tr>
<td>13</td>
<td>39.4</td>
<td>2.8</td>
<td>12.5</td>
<td>0.79</td>
</tr>
</tbody>
</table>

Table 3.1 FIFO Comparison by Iteration (average of 10 trials)
While having equal number of resources to resources needed is highly unlikely, it is helpful to run the simulation with these parameters. This verifies that task completion and quality will improve proportionately. It can also show if the policy is operating close to max efficiency with the given amount of resources. Since some iterations contain more or less tasks than the mean we do get a backlog at times but these tasks will be finished in subsequent iterations. With no emphasis on higher priority tasks, longer tasks with low priority will still starve out potentially higher quality achieving tasks.

<table>
<thead>
<tr>
<th>One Year Comparison (average of 10 trials)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks Completed</td>
</tr>
<tr>
<td>Equal Resources</td>
</tr>
<tr>
<td>Scarce Resources</td>
</tr>
<tr>
<td>Surplus Resources</td>
</tr>
</tbody>
</table>

Table 3.2 FIFO Comparison using average of multiple trials

Lastly, we consider what happens when resources are greater than tasks by 10%. This is mainly done to show how the task completion and quality ratio change as resources are removed/added. This gives a quantifiable measure to how well a scheduling policy will degrade as resources fluctuate.

3.2.2 SJF Analysis

The most noticeable result with SJF is the amount of tasks completed. Tasks completed set the benchmark for all other policies. Unfortunately it does so without regard to business logic or quality. SJF task completion is used as a goal for an optimized solution which takes priority into account. Table 3.3 below summarizes the results using MoneyWare multiple times with the SJF scheduling policy. Tasks completed are higher than either of our other two “basic” policies but quality ratio is well below. This occurs
because SJF will order the tasks by resources needed, which is a factor of size and priority. The lower the priority the less resources allocated to it, thereby having the effect of ordering low priority tasks mostly at the front of the queue. This has the effect of completing many tasks but at a low quality, as it should. Completing a large number of low priority tasks should lower the overall portfolio quality. If all tasks are of the same priority then SJF would be an ideal scheduling policy.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Total Tasks Completed</th>
<th>Tasks Completed/Iteration</th>
<th>Backlog</th>
<th>Quality Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1.1</td>
<td>0.88</td>
</tr>
<tr>
<td>2</td>
<td>5.9</td>
<td>2.9</td>
<td>2.3</td>
<td>0.89</td>
</tr>
<tr>
<td>3</td>
<td>9.1</td>
<td>3.2</td>
<td>3.2</td>
<td>0.86</td>
</tr>
<tr>
<td>4</td>
<td>12.4</td>
<td>3.3</td>
<td>3.8</td>
<td>0.88</td>
</tr>
<tr>
<td>5</td>
<td>15.5</td>
<td>3.1</td>
<td>4.9</td>
<td>0.85</td>
</tr>
<tr>
<td>6</td>
<td>18.9</td>
<td>3.4</td>
<td>5.6</td>
<td>0.83</td>
</tr>
<tr>
<td>7</td>
<td>22.2</td>
<td>3.3</td>
<td>6.2</td>
<td>0.8</td>
</tr>
<tr>
<td>8</td>
<td>25.3</td>
<td>3.1</td>
<td>7.3</td>
<td>0.79</td>
</tr>
<tr>
<td>9</td>
<td>28.7</td>
<td>3.4</td>
<td>8.1</td>
<td>0.79</td>
</tr>
<tr>
<td>10</td>
<td>31.9</td>
<td>3.2</td>
<td>8.9</td>
<td>0.77</td>
</tr>
<tr>
<td>11</td>
<td>35.1</td>
<td>3.2</td>
<td>9.5</td>
<td>0.78</td>
</tr>
<tr>
<td>12</td>
<td>38.2</td>
<td>3.1</td>
<td>10.3</td>
<td>0.8</td>
</tr>
<tr>
<td>13</td>
<td>41.2</td>
<td>3</td>
<td>11.1</td>
<td>0.78</td>
</tr>
</tbody>
</table>

Table 3.3 SJF Comparison by Iteration (average of 10 trials)

Secondly, we consider the scenario of equal resources versus equal size of tasks in the queue. We see a marked improvement in quality at this point. With enough resources available most every task will be completed and quality ratio will be comparable with the other policies. Task completion improves slightly and quality ratio approaches the other scheduling policies.
Table 3.4 SJF Comparison using average of multiple trials

<table>
<thead>
<tr>
<th>Equal Resources</th>
<th>Tasks Completed</th>
<th>Backlog at End of Year</th>
<th>Average Quality Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scarce Resources</td>
<td>41.2</td>
<td>11.1</td>
<td>0.7</td>
</tr>
<tr>
<td>Surplus Resources</td>
<td>51.4</td>
<td>0.6</td>
<td>1.01</td>
</tr>
</tbody>
</table>

At this point SJF is starting to approach an optimum amount of task completion. This shows the amount of tasks our optimum approach should be able to achieve but not quality ratio, as SJF does not worry about which tasks might achieve a higher quality ratio.

3.2.3 ITF Analysis

While ITF satisfies business logic it does not optimize tasks completed and we see a large number of backlogged tasks. As we can see from Table 3.5 quality ratio is substantially higher than the other two policies. This should be expected with limited resources as the only tasks being completed are high priority, and at a high quality. This is easily explained considering important jobs tend to require more resources to guarantee they are finished on time. This effectively starves out a large number of small easily completed tasks leading to a large number of backlogged tasks. Our goal for optimization is to find the combination of SJF and ITF. Essentially trying to complete as many tasks as SJF will but adhering to the business logic and quality level of ITF.
Similar to SJF, when resources are abundant most if not all tasks will be completed. While task completion will go up, quality will go down. This happens because the lower priority tasks will be completed. Tasks having a low priority detract from the quality of the portfolio. The analysis of the portfolio needs to take into account both tasks completed and quality.

Table 3.5 ITF Comparison by Iteration (average of 10 trials)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Total Tasks Completed</th>
<th>Tasks Completed/Iteration</th>
<th>Backlog</th>
<th>Quality Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.5</td>
<td>2.5</td>
<td>2</td>
<td>1.1</td>
</tr>
<tr>
<td>2</td>
<td>5.3</td>
<td>2.8</td>
<td>3</td>
<td>1.08</td>
</tr>
<tr>
<td>3</td>
<td>7.9</td>
<td>2.6</td>
<td>3.8</td>
<td>1.1</td>
</tr>
<tr>
<td>4</td>
<td>10.6</td>
<td>2.7</td>
<td>4.7</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>13.5</td>
<td>2.9</td>
<td>6</td>
<td>1.05</td>
</tr>
<tr>
<td>6</td>
<td>16.6</td>
<td>3.1</td>
<td>7</td>
<td>1.1</td>
</tr>
<tr>
<td>7</td>
<td>19.6</td>
<td>3</td>
<td>7.8</td>
<td>1.04</td>
</tr>
<tr>
<td>8</td>
<td>22.2</td>
<td>2.6</td>
<td>9.1</td>
<td>1.08</td>
</tr>
<tr>
<td>9</td>
<td>24.7</td>
<td>2.5</td>
<td>10.1</td>
<td>1.1</td>
</tr>
<tr>
<td>10</td>
<td>27.5</td>
<td>2.8</td>
<td>11.4</td>
<td>1.07</td>
</tr>
<tr>
<td>11</td>
<td>29.9</td>
<td>2.4</td>
<td>13</td>
<td>1.09</td>
</tr>
<tr>
<td>12</td>
<td>32.7</td>
<td>2.8</td>
<td>14.4</td>
<td>1.06</td>
</tr>
<tr>
<td>13</td>
<td>35.1</td>
<td>2.4</td>
<td>16.1</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Table 3.6 ITF Comparison using average of multiple trials

<table>
<thead>
<tr>
<th>Resources</th>
<th>Tasks Completed</th>
<th>Backlog at End of Year</th>
<th>Average Quality Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal Resources</td>
<td>46.8</td>
<td>6.8</td>
<td>1.05</td>
</tr>
<tr>
<td>Scarc Resources</td>
<td>35.1</td>
<td>16.1</td>
<td>1.1</td>
</tr>
<tr>
<td>Surplus Resources</td>
<td>49.9</td>
<td>0.8</td>
<td>0.97</td>
</tr>
</tbody>
</table>

Of note here is the better quality achieved while completing less tasks. This mirrors what occurs in businesses every day. Worry only about the important tasks and forget about the rest, especially when resources are at a premium. Unfortunately it is not
achieving near the amount of tasks completed as SJF. While this may be acceptable for some companies, they are essentially losing money every day with this policy. In establishing a portfolio quality metric we will show in the following chapter how much more efficient an optimized scheduling policy can be. This directly translates into increased profits from more tasks completed at a higher quality and less resources wasted.

3.3 Comparison of all Three Policies

Now that we have a sufficient amount of data from three different policies, how do they compare to each other and is there a baseline that we can strive to better with our future attempts?

We compare the three policies using the results obtained with the resource supply less than the task demand. This allows us to see if and to what effect the scheduling policy has on task completion and quality.
Figure 3.10 Quality Ratio for FIFO, SJF and ITF

Figure 3.11 Comparison of Tasks Completed and Quality Ratio (average of 10 trials)
From the figures above we can see the relative strengths and weaknesses of our first three policies. ITF has a quality ratio that is far greater than the other two. This occurs due to the fact that ITF with limited resources will always choose the most important tasks and provide plenty of resources to complete them. This results in high quality but limited amount of tasks completed.

SJF completes on average 6 more tasks per year or 1.5 iterations of tasks more. This is equivalent to 6 more weeks of work. Unfortunately it does so without much regard to quality. Our assumption is that most CIOs would prefer to achieve a much higher level of quality from the software produced.

FIFO sits firmly in the middle of these two. Which is another way of saying it doesn’t really do anything very well. At best we can expect slightly better quality than SJF and a fair amount more tasks completed than ITF.
Another metric we can use to further compare the scheduling policies is quality points. From figure 3.12 we see ITF averages 2.7 points higher than the other two. Quality points are the aggregate measure of size and quality ratio of the completed tasks in an iteration. If the user values quality over all else then ITF has shown to be the more productive policy. This is one issue that we hope to improve on. Our improved policies should be able to complete more tasks than ITF and improve on its quality point value.

<table>
<thead>
<tr>
<th>Quality Points Comparison (average of 10 trials)</th>
<th>FIFO</th>
<th>SJF</th>
<th>ITF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Quality Points for one year</td>
<td>325.9</td>
<td>327</td>
<td>361.6</td>
</tr>
<tr>
<td>Average Quality Points Per Iteration</td>
<td>25.1</td>
<td>25.15</td>
<td>27.85</td>
</tr>
</tbody>
</table>

Table 3.7 Quality Points Comparison
Using figure 3.13 we can explore the backlog growth when resources are limited.

Backlog is the amount of unfinished tasks per iteration. For some companies, backlog can represent a source of lost revenue. Backlog will lead to missed deadlines and as such will cause companies to incur monetary penalties. Limiting the backlog growth is one metric that should be monitored closely. ITF has on average 4 more tasks in its backlog than the others. This represents an average iteration worth of tasks not being completed compared to the other two.

For our better policies we would like to see quality near ITF and tasks completed near SJF. Since both of these only focus on one metric, we should strive to approach both of these numbers. We have seen that as task completion goes up that quality will have to come down a little (we will be completing some of the lower quality tasks as well). If we
can focus on completing all the important tasks first and only use excess resources on the lower priority tasks we will guarantee our quality level will be sufficiently high.
Chapter 4 Scheduling for Optimized Quality

There is considerable complexity involved with task definition, scheduling, development and maintenance. In what ways can we reduce this bottleneck and ultimately achieve a higher software portfolio quality? If we can increase the tasks completed and follow the business logic of completing higher priority tasks, then we will effectively decrease this bottleneck while achieving a higher portfolio quality.

Having examined several common scheduling policies in the previous chapter we can now set our sights on improving our task completion and increasing portfolio quality. By examining the general definition of portfolio quality we can begin to determine what will lead to improvements.

Portfolio quality consists of the following factors:

- Completing tasks on time
- Completing tasks with the needed amount of resources
- Completing higher quality tasks

Fewer resources used or completing low priority tasks will both reduce the portfolio quality. It is reasonable that if a software portfolio consists mainly of low priority tasks, than we should consider it of less quality than one with higher priority tasks. If this were not true, then companies would simply focus on completing as many tasks as possible.

By evaluating the portfolio with tasks completed and quality we can determine better policies.

We also use quality points as a metric to further compare the different policies. This metric allows us to compare quality ratio with the size of the task. We need to develop policies that help to reduce the amount of starvation that occurs when large, high priority
tasks use up all the resources before the smaller but important tasks get a chance to be completed.

In this chapter we develop and quantify two of our own scheduling policies. We call our first attempt Priority-Then-Size (PTS). It modifies ITF to accomplish more completed tasks while keeping quality roughly the same. Our next method is a Heuristic-Based-Policy (HBP). It will further refine PTS by employing heuristics to provide thresholds for ordering purposes.

Section 4.1 *Scheduling the Tasks* follows our pattern from the previous chapter in examining the two “better” approaches using different resource levels.

Section 4.2 *Analysis* analyzes the two scheduling policies for portfolio performance using the criteria specified in the previous section.

Section 4.3 *Comparison of HBP and PTS Policies* looks at all policies using resource supply less than task demand. We compare the two better attempts at scheduling policies we proposed in this chapter with the three from last chapter.

### 4.1 Scheduling the Tasks

Last chapter we examined several different scheduling policies. In doing so we established some baseline measures against which to compare our more sophisticated attempts presented in this chapter. We start by looking at both of these “better” policies using resources equal to the task demand. Next we remove 20% of the available resources and examine how this affects the quality and task completion and compare to the earlier policies.
4.1.1 Tasks with Equal Resources

Our testing protocol remains the same as last chapter. We first examine the different scheduling policies with available resources equal to the resources needed. Running the simulator for equal resources also provides a basis for the amount of task completion and quality loss that occurs when resources are removed. It also can allow the CIO to test different scheduling policies based on technical staff present or technical staff needed.

**Priority-Then-Size Policy (PTS)**

The previous 3 policies each had its drawbacks. First-in-first-out is the simplest to implement but does not attempt to optimize any parameter. Shortest-job-first completes a large number of tasks, but does not attempt to optimize quality or take into account the priority of a task. Important-task-first optimizes priority but will allow large tasks to starve out smaller tasks. From the weaknesses of these policies we can introduce a “better” solution. Tasks are ordered by priority first and then by resources needed. From this ordering we complete smaller tasks with equal priority first. This attempts to better utilize resources while still focusing on achieving high quality. This has the advantage of reordering tasks every iteration based on two factors. Figure 4.1 shows the results from running the simulator with the standard task parameters outlined in chapter 3.
Figure 4.1 PTS with Resources Equal to Task Demand (average of 10 trials)

With equal resources we expect to see very little backlog growth, the growth represented in the graph above occurs mainly from tasks not completing in the last couple of iterations before ending the simulation. Examining the results by iteration shows the backlog growth at around .25 tasks per iteration. Quality ratio and tasks completed are similar to ITF as would be expected since both policies share similar methods and with enough resources available should exhibit similar results.

Heuristic-Based-Policy (HBP)

Finally, we introduce our second and best attempt at a better solution. Taking into account the strengths and weaknesses of the previous four policies we developed a policy that can match SJF in terms of task completion and ITF in terms of quality. Tasks are ordered by priority first and then by date needed. From this ordering we select any task within a given range (explained below) to be completed first. This allows tasks with slightly less priority but smaller size to be completed before large tasks with high
priority. We use a heuristic based on the size to determine this threshold. This is a fixed value determined by multiplying 1.5 times the user selected size. This has led to the greatest increase in both task completion and quality. Similar to PTS this has the advantage of reordering tasks every iteration based on multiple factors. Tasks that have the most potential to be achieved with high quality will be completed earlier. Figure 4.2 shows the results from running the simulator with the standard task parameters listed above.

![Resource Equal to Task Demand](image)

Figure 4.2 HBP with Resources Equal to Task Demand (average of 10 trials)

Task completion, quality ratio and backlog are in-line with the best of the previous policies. This indicates, unlike ITF, that the performance will increase in all three categories as resources increase.

4.1.2 Tasks with Resources Less than Task Demand
We now examine the results produced using 20% less resources than needed. We chose this resource level as it allows the biggest variations between the policies. This guarantees the incoming tasks will require more resources than are available, thus allowing us to examine how well each policy performs under pressure. We will use this resource level for comparisons in our analysis later this chapter. The user is able to adjust the level of resources to match what they have available. It also allows the user to simulate at a lower level of resources to see if the same task completion and quality can be achieved with fewer resources. Figures 4.3 and 4.4 show the results of the simulator for each of the scheduling policies using our standard parameters listed above with the exception of resources available now equal to 9.

Figure 4.3 PTS with Resources Less than Task Demand (average of 10 trials)

We expect to see a rapid increase in backlog when the resource supply is less than the task demand. How quickly this increase occurs is one feature we can use to gauge the differences of each policy. PTS shows a noticeable decrease in the backlog compared to
ITF, although it still has a slightly faster growing backlog than SJF. Quality ratio has improved versus itself with more resources. This is the same occurrence we noted with ITF in the previous chapter. With limited resources, only the important tasks are being completed.

![Resources Less than Task Demand](image)

Figure 4.4 HBP with Resources Less than Task Demand (average of 10 trials)

Examining the graph above shows a noticeable increase in tasks completed per iteration and a much smaller increase in backlog compared to the other policies. Quality ratio has remained high while completing a large number of tasks; something none of the other policies were able to accomplish.

### 4.1.3 Comparison against baseline results

One interesting result noticed is how HBP with fewer resources compares to the baselines with equal resources. In table 4.1 below we examine how HBP compares to the baselines with enough resources from the policies discussed in chapter 3.
Table 4.1 HBP with Fewer Resources Compared to Baseline Results

<table>
<thead>
<tr>
<th></th>
<th>Tasks Completed</th>
<th>Backlog at end of year</th>
<th>Average Quality Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>SJF</td>
<td>49.9</td>
<td>2</td>
<td>0.89</td>
</tr>
<tr>
<td>FIFO</td>
<td>49.8</td>
<td>2</td>
<td>0.9</td>
</tr>
<tr>
<td>ITF</td>
<td>46.8</td>
<td>6.8</td>
<td>1.05</td>
</tr>
<tr>
<td>HBP</td>
<td>45.5</td>
<td>7.4</td>
<td>1</td>
</tr>
</tbody>
</table>

While HBP does not equal the others, it compares favorably. This result shows that HBP while not able to complete as many tasks as SJF and FIFO it approaches their results but at a higher quality ratio. Any reduction in quality would be a cause for concern and one that must be considered carefully by the CIO. By choosing HBP in the above scenario resources are 20% less for the entire year, compared to the other three, or 26 developer weeks of time saved.

### 4.2 Analysis

While our previous chapter focused on creating a baseline performance on which to judge future policies, this chapter looks at two of those “better” policies. From the previous chapter we noted that Important-Task-First achieved a high quality performance but did so while completing much fewer tasks. Shortest-Job-First completed a high number of tasks, even when resources were scarce, but quality suffered greatly. This chapter focused on combining the performance from ITF and SJF, if possible.

#### 4.2.1 PTS Analysis

PTS took a better but still simple approach of ordering the task queue using two measures. Using table 4.2 we can start to examine the successes accomplished by PTS over previous policies. PTS completes more tasks than ITF and at a similar quality level.
This is encouraging but the number of tasks completed is still far below what SJF accomplished. The growth in task completion versus ITF is encouraging but still below what should be considered acceptable. High priority tasks will still starve out medium priority tasks especially if there is a large number of them. From a quality standpoint, we need to ensure medium priority tasks can be completed with the same frequency as high priority. With PTS this is not possible.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Total Tasks Completed</th>
<th>Tasks Completed/Iteration</th>
<th>Backlog</th>
<th>Quality Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.9</td>
<td>2.9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5.7</td>
<td>2.8</td>
<td>2</td>
<td>0.99</td>
</tr>
<tr>
<td>3</td>
<td>8.7</td>
<td>3.0</td>
<td>2.9</td>
<td>1.05</td>
</tr>
<tr>
<td>4</td>
<td>11.6</td>
<td>2.9</td>
<td>3.8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>14.7</td>
<td>3.1</td>
<td>5</td>
<td>1.05</td>
</tr>
<tr>
<td>6</td>
<td>17.8</td>
<td>3.1</td>
<td>5.9</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>20.7</td>
<td>2.9</td>
<td>7.1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>23.8</td>
<td>3.1</td>
<td>8</td>
<td>0.99</td>
</tr>
<tr>
<td>9</td>
<td>26.7</td>
<td>2.9</td>
<td>3.2</td>
<td>0.98</td>
</tr>
<tr>
<td>10</td>
<td>29.7</td>
<td>3.0</td>
<td>10.1</td>
<td>0.99</td>
</tr>
<tr>
<td>11</td>
<td>32.8</td>
<td>3.1</td>
<td>11.5</td>
<td>0.97</td>
</tr>
<tr>
<td>12</td>
<td>35.6</td>
<td>2.8</td>
<td>12.4</td>
<td>0.99</td>
</tr>
<tr>
<td>13</td>
<td>38.4</td>
<td>2.8</td>
<td>13.2</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Table 4.2 PTS Comparison by Iteration (average of 10 trials)

While having equal number of resources to resources needed is highly unlikely, it is helpful to run the simulation with these parameters. This verifies that task completion and quality will improve proportionately. It can also show if the policy is operating close to max efficiency with the given amount of resources. Examining table 4.3 below we see a large gap between task completions with less resources than the other two levels. This indicates this policy is not efficient in completing tasks. It suffers a similar fate as ITF did from the previous chapter. It improves on ITF but not enough for our satisfaction.
4.2.2 HBP Analysis

Heuristic-Based-Policy attempts to bridge the gap between task completion and task quality. We have shown that it is easy to achieve high numbers when attempting to maximize one of these metrics, but there has always been a tradeoff. From Table 4.4 below we see that HBP completes the same number of tasks as SJF with a quality ratio very close to that of ITF, successfully bridging the gap between the two.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Total Tasks Completed</th>
<th>Tasks Completed/Iteration</th>
<th>Backlog</th>
<th>Quality Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>3.5</td>
<td>0.9</td>
<td>1.01</td>
</tr>
<tr>
<td>2</td>
<td>7.5</td>
<td>4</td>
<td>1.5</td>
<td>0.98</td>
</tr>
<tr>
<td>3</td>
<td>10.8</td>
<td>3.3</td>
<td>2.2</td>
<td>1.02</td>
</tr>
<tr>
<td>4</td>
<td>14.3</td>
<td>3.5</td>
<td>2.9</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>17.8</td>
<td>3.5</td>
<td>3.6</td>
<td>0.99</td>
</tr>
<tr>
<td>6</td>
<td>21.1</td>
<td>3.3</td>
<td>4.1</td>
<td>1.03</td>
</tr>
<tr>
<td>7</td>
<td>24.6</td>
<td>3.5</td>
<td>4.7</td>
<td>0.99</td>
</tr>
<tr>
<td>8</td>
<td>28.1</td>
<td>3.5</td>
<td>5.1</td>
<td>0.99</td>
</tr>
<tr>
<td>9</td>
<td>31.8</td>
<td>3.7</td>
<td>5.8</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>35.3</td>
<td>3.5</td>
<td>6.2</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>38.8</td>
<td>3.5</td>
<td>6.5</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>42.1</td>
<td>3.3</td>
<td>5.9</td>
<td>0.99</td>
</tr>
<tr>
<td>13</td>
<td>45.5</td>
<td>3.4</td>
<td>7.4</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Table 4.4 HBP Comparison by Iteration (average of 10 trials)

What allows HBP to perform so well are the multiple selection criteria it incorporates.

We know that tasks that are medium or high priority if completed with sufficient
resources will improve the quality of the portfolio. HBP policy selects any task with either of these priorities that is small enough to complete quickly. This eliminates the starvation that tends to occur when using only one or two characteristics to schedule the tasks. Table 4.5 below shows an example of completed tasks using the HBP scheduling policy.

<table>
<thead>
<tr>
<th>HBP Example Completed Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>14</td>
</tr>
</tbody>
</table>

Table 4.5 HBP Example Completed Task Queue

Secondly, we consider the scenario of equal resources versus equal size of tasks in the queue. We do not see much improvement as we add resources, this shows that HBP is operating at near maximum ability with 20% less resources. From a CIO’s standpoint this would indicate either more tasks can be added to the queue or less resources are needed (for the employee’s sake we will hope they choose the former).

<table>
<thead>
<tr>
<th>One Year Comparison (average of 10 trials)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks Completed</td>
</tr>
<tr>
<td>Equal Resources</td>
</tr>
<tr>
<td>Scarce Resources</td>
</tr>
<tr>
<td>Surplus Resources</td>
</tr>
</tbody>
</table>

Table 4.6 HBP Comparison using average of 10 trials
4.3 Comparison of All Policies

Looking at the results obtained last chapter and comparing the results described above we were able to accomplish our goal of combining the best of SJF and ITF? We compare the five policies using the results obtained with the resource supply less than the task demand. This allows us to see if and to what effect the scheduling policy has on task completion and quality.

![Comparison of Quality Ratio and Task Completion](image)

Figure 4.5 Comparison of Tasks Completed and Quality Ratio (average of 10 trials)

From figure 4.5 above we notice immediately the large number of tasks completed by HBP. Using the scale on the right of the graph (each line is equivalent to the average number of tasks in an iteration); we can see the number of iterations worth of tasks gained using HBP. For example, HBP completed a full iteration more tasks than SJF and almost 2.5 more than ITF. This would allow a CIO to either schedule more tasks or
reduce resources needed. The big improvement is the combination of tasks completed while achieving a quality ratio at a 1.0 level.

<table>
<thead>
<tr>
<th>Task Comparison of Important Portfolio Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks Completed</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>FIFO</td>
</tr>
<tr>
<td>SJF</td>
</tr>
<tr>
<td>ITF</td>
</tr>
<tr>
<td>PTS</td>
</tr>
<tr>
<td>HBP</td>
</tr>
</tbody>
</table>

Table 4.7 Comparison of All Scheduling Policies (average of 10 trials)

After completing and examining many trials using all scheduling policies, achieving as close to a 1.0 quality ratio is the ideal level. This occurs due to the fact that the more tasks completed the better chance of completing lower priority tasks that will offset the higher quality numbers achieved with high priority tasks. Quality ratio numbers higher than a 1.0 indicate a policy that will mostly favor completing high priority tasks but not completing as many tasks.

![Quality and Tasks Completed Comparison](image)

Figure 4.6 Quality Comparison of all Five Policies (average of 10 trials)
Figure 4.7 Quality Ratio Comparison (average of 10 trials)

From figure 4.7 we can see that ITF consistently averages a higher quality ratio than the other policies. This would be an ideal choice if number of tasks completed was not an issue. ITF focuses strictly on completing only important tasks and doing so with high quality. Figure 4.8 clearly shows the discrepancy in quality ratio between SJF and HBP even though HBP completes more tasks.

Another metric useful for comparing the different policies is quality points. This metric is an aggregate of the individual tasks quality, accounting for size and quality of the task completed. Table 4.8 and figure 4.8 below shows the results for quality points from all policies.
We notice similar results with quality points as we did with task completion and quality ratio. Quality points give us a better indication of the portfolio as a whole. We notice HBP performs noticeably better than the other four. This can be attributed to completing medium and high priority tasks and completing a large number of them. This effectively shows how efficient HBP is at ordering the tasks in the queue.

<table>
<thead>
<tr>
<th>Quality Points Comparison (average of 10 trials)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>325.9</td>
</tr>
<tr>
<td>25.1</td>
</tr>
</tbody>
</table>

Table 4.8 Quality Points Comparison for all Scheduling Policies

Table 4.8 gives us a good indication of the overall ability of each scheduling policy. We do see ITF with a high number due to completing mostly high priority tasks. We would
still need to consider the other metrics when evaluating the overall merit of the scheduling policy.

Along with task completion and quality ratio, we judge each policy on the backlog growth. It is important that a policy can limit this growth as much as possible. Depending on the policy chosen this could be limiting a task from ever being completed. In the case of SJF, a large task may never be completed. Likewise in ITF, a low priority task will sit in the backlog indefinitely.

![Backlog Comparison](image)

Figure 4.9 Backlog Growth Comparison of all Five Policies (average of 10 trials)

With resource supply less than task demand, backlog growth will have to occur. We examine the number of tasks that accumulate in the backlog. From figure 4.9 above we can use this growth as a very good measure of one metric to compare the different scheduling policies. Not only does HBP end with a much smaller amount, the growth rate is substantially less as well.
For our better policies we would like to see quality near ITF and tasks completed near SJF. Since both of these only focus on one metric, we should strive to approach both of these numbers. We have seen that as task completion goes up, quality will have to come down a little (we will be completing some of the lower quality tasks as well). If we can focus on completing all the important tasks first and only use excess resources on the lower priority tasks we will guarantee our quality level will be sufficiently high.

<table>
<thead>
<tr>
<th>Schedule Policy Final Rankings</th>
<th>Tasks Completed Ranking</th>
<th>Backlog of Tasks Rankit</th>
<th>Average Quality Ratio Ranking</th>
<th>Quality Points</th>
<th>Final Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBP</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PTS</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>FIFO</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>ITF</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>SJF</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 4.9 Final Rankings for all Scheduling Policies

Table 4.9 above lists our ranking of each of the different scheduling policies. This is an arbitrary ranking based on our definitions of what we feel most important when scheduling tasks. There is no denying that HBP is the clear best choice for task completion, backlog growth and quality achieved. Our subjective rankings will mostly concern the other four policies. PTS while not completing as many tasks as SJF, is close enough and with a much better quality ratio. After the first two, each of the rest has a major limitation. Each of these policies is in use today so companies are able to work around the limitations. It is our hope that given the results produced by MoneyWare that more companies will explore the potential gains available using the HBP policy.
Chapter 5 Case Study

The original research produced by Dharani Murthy focused on a theoretical approach, likely due to the difficulty in trying to calibrate against an actual organization’s situation. Our interest was in taking ideas like hers and exploring what it would take to make a model more aligned with actual practice.

In this chapter we explore how MoneyWare performs using data obtained from a real world company. This company focuses on producing hardware and software for external clients. They mostly deal with government contracts and as such put an emphasis on high quality software. I was lucky enough to spend three months interning with this company. Many of the key ideas presented in this thesis such as software development for external customers and task sequencing were highly influenced by the company’s practices.

We discuss an in-depth look at a specific company’s data and see if the simulator can be used to explore potentially better scheduling solutions. The data collected comes from the real-world company discussed in chapter 1 STC. This data was collected using metrics stored in Jira over the course of 52 developer-weeks. The data is presented in table 5.1 below. Calculating their task completion percentage shows they completed 77% of their tasks. From interviews with technical staff present at the company, their quality ratio is roughly 90%. Since they did not have true software quality metrics available this was an educated guess by their technical staff.
Table 5.1 STC’s Task Characteristics for 52 weeks

Since MoneyWare is calculated to allow only an integer amount for tasks per iteration we chose to run the simulator with a mean of 3 tasks per iteration. This results in an average of 39 tasks per year. Using this information we focused not on the actual number of tasks completed but on the percentage.

STC adhere to a scheduling policy almost exactly the same as our Priority-Then-Size policy. With the data provided above we can adjust MoneyWare to closely match these parameters. From interviews with programmers at STC, they run slightly less than 20% below resources available to resources needed. We will adjust for this in the simulator and run using each of the scheduling policies presented in previous chapters. As we did before, all policies were run 10 times and the results averaged. The following parameters were used for all policies:

- Number of tasks- mean of 3 tasks per iteration
- Size- mean of 6 developer weeks, standard deviation of 1
- Priority- 40% low, 40% medium, 20% high
- Risk- 30% low, 50% medium, 20% high
- Date needed- mean of 7 weeks, standard deviation of 1
- Number of iterations- fixed at 13 (one year)
- Resources available- 4 developer weeks
5.1 Comparing the Scheduling Policies

Due to the tasks per iteration restrictions when using MoneyWare we need to look at percentages rather than actual numbers. From above we know that STC completed 77% of their tasks and at a quality ratio of roughly 90%. Using table 5.2 below we can compare how STC performed versus how they should expect to perform based on the simulator’s results.

<table>
<thead>
<tr>
<th></th>
<th>Tasks Completed</th>
<th>Backlog at end of year</th>
<th>Average Quality Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITF</td>
<td>21</td>
<td>18.2</td>
<td>1.1</td>
</tr>
<tr>
<td>PTS</td>
<td>30</td>
<td>9</td>
<td>0.93</td>
</tr>
<tr>
<td>HBP</td>
<td>32.2</td>
<td>6.4</td>
<td>0.99</td>
</tr>
<tr>
<td>STC Actual</td>
<td>26</td>
<td>6</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 5.2 Comparison of All Policies using STC’s Parameters

Checking the results for PTS (the policy STC uses), shows a task completion of 77% with slightly higher quality ratio then STC. This further validates the accuracy of MoneyWare, at least in relation to STC’s data. With this information we can now evaluate STC’s choices and if there is a potentially better alternative.

5.2 Observations

Using the provided data we can see that STC achieves a respectable amount of task completion and quality. This comes from experienced task schedulers using a fairly consistent and reliable amount of resources. Our findings above show that STC could produce, on average, two more tasks per year than they currently are accomplishing.
Is it worth the effort for two tasks per year? STC averages roughly $4 million in revenue per year. They also average about $100,000 revenue per task, so two tasks more would equate to roughly $200,000 gained. If we assume a 20% profit margin, their profit per year is $800,000. Gaining $200,000 per year or ¼ of their profit margin without adding extra human resources is significant. A significant parameter not taken into account for MoneyWare but essential in how STC functions is the backlog. Most contracts that STC has include a clause resulting in penalties for late delivery. The substantial decrease in backlog has the additional benefit of fewer penalties incurred. The idea of MoneyWare is to show how our attempt at a better policy can achieve a higher performance than current policies. It also allows the user to run hypothetical parameters to discover better resource allocations and/or number of tasks that can be completed.
Chapter 6 Conclusions and Future Direction

MoneyWare derives its name from Moneyball, a film which follows a baseball organization that exploits the idea of using statistical properties of development tasks to direct resource allocation and development priority. One of these ideas explained in the following passage from the book Mathletics by Wayne Winston is the main goal of MoneyWare [7]:

“The key fielding statistic is UZR (Ultimate Zone Rating). A player’s UZR is how many more runs a player’s fielding saved compared to an average fielder.

During 2007-8 the value of fielding was underappreciated, so by buying defense inexpensively the Tampa Bay Devil Rays were able to improve their fielding and gain twelve wins! This was a key reason the Rays were 2008 AL champs.”

The Devil Rays made use of a little known but important baseball statistic, to buy inexpensive players that resulted in more wins and used fewer resources. This led to a much improved level of success on a fraction of the budget of most other teams.

MoneyWare attempts to employ some of the same ideas that the team featured in Moneyball. This team made use of, what at the time were considered, minor baseball statistics to purchase undervalued players. This allowed them to compete favourably using less resources.

MoneyWare attempts something similar, the use of a new metric which we have called the quality ratio, to save resources all the while increasing the performance of the software portfolio.

MoneyWare is another area where this same idea can be applied. By scheduling tasks using SJF, FIFO or ITF we follow what has been done before. But is it possible to
accomplish the same or better results with fewer resources? By developing a more descriptive statistic, which we call the quality ratio, we can better quantify the consequences of the different policies. Once we can better measure the effects of the different policies, we can attempt to create higher performing ones. With HBP using a combination of ideas and policies we are able to complete more tasks while achieving a higher quality ratio. Not only that, but the quality ratio is less susceptible to peaks and valleys.

6.1 Conclusion

This thesis investigation has followed two themes. One is the development of a simulator which mimics the functionality of task management in an organization which develops software for external clients. The other an attempt to create better scheduling policies using the simulator to verify the results.

MoneyWare has several strengths that have been demonstrated throughout this research. By limiting the characteristics available we are able to simulate on a simple level an actual company’s task completion and backlog. This gives validation to the idea and initial work of this simulator. We were also able to quantify various scheduling policies using a combination of existing and new metrics. We were able to show how scheduling based on task size only or priority only scheduling can lead to less than ideal performance.

Yet there are still weaknesses to acknowledge. While our goal was to limit the complexity of MoneyWare to facilitate the completion of the research, it meant sacrificing some features needed for a marketable product. Task sequencing would be the top priority of features not included but needed for future growth. Allowing the user
to specify a percentage of resources available to apply to each task would help make the simulator more realistic.

### 6.2 Future Directions

We finish this chapter discussing the different areas that can be investigated in the future. We look at the many different areas that hold the most promise for future research. There are two main directions future research could take: improve the scheduling algorithms, or attempt to turn the simulator into a marketable product (although this would require much work.)

#### 6.2.1 Further Investigation of Scheduling Algorithms

We believe the area most ripe for improvement is applying better heuristics to the scheduling process. Here are several different ideas for moving forward.

**Investigate the formulation as knapsack problem**

The knapsack problem is a classic computer science problem that states: Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. From this definition we can see that our task scheduling problem can be formulated as a knapsack problem. However, there is an additional condition that tasks not scheduled in one iteration are moved to the next. This additional condition can be overlooked for the classic knapsack solution since it would try to solve each iteration using the parameters supplied for that particular iteration.
In their paper “Heuristic Algorithm For Single Resource Constrained Project Scheduling Problem Based on the Dynamic Programming” Stanimirovic et al. [6], introduce a heuristic method to solve a very similar problem to our scheduling policy. We believe that their research could be applied to this project area with great success.

**Learning Agents**

Learning agents have an advantage in that it allows the agents to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow. The most important distinction is between the "learning element", which is responsible for making improvements, and the "performance element", which is responsible for selecting external actions.

The learning element uses feedback from the "critic" on how the agent is doing and determines how the performance element should be modified to do better in the future. The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.

The last component of the learning agent is the "problem generator". It is responsible for suggesting actions that will lead to new and informative experiences.

By applying a learning process to the simulator it should be possible for the simulator to begin to predict future tasks and compute the best possible scheduling for the current iteration.

**6.2.2 Creation of a Marketable Product**

This research focused on the theoretical aspect of determining portfolio quality using different scheduling policies. We would be remiss if we did not embrace the idea and
potential for such a product in the project management field. The initial research conducted by Sindhu Dharani Murthy and Dr. David Erbach focused entirely on the theoretical approach while the continuation of this research we performed moved one step closer to an applied approach.

With that in mind we begin to look at where this product might ultimately find a market in the project manager’s set of tools. We acknowledge that for this product to go to market would require substantial modifications for both input and output. This would require a number of tests to determine what users want and how to best produce output that would be beneficial.

One change or addition that would be needed is that of considering all the small tasks that make up the large task we focused our research on. Although this might not make it into a first release it is an important issue that would need to be addressed. We also need to continue to add task characteristics to have them further resemble actual tasks. This would allow for the following features:

- The ability to add in a critical path type algorithm.
- The ability to complete tasks simultaneously.

While these features would turn this simulator into a viable product they would not change the results currently produced by the simulator.

### 6.3 Lessons Learned

This section addresses the reader who wishes to pursue this line of research further. By recounting the problems and solutions we encountered it is our hope that future research can expand on this without running into some of the same issues we did.
• Understand the differences between software quality and portfolio quality early. Make sure any audience understands this difference from the beginning. Since the concept for this simulator is likely to be unfamiliar, it is important to distinguish between these two concepts early and often.

• Limit the number of variables used in the simulator. Most variables can be accounted for in other ways and rarely lead to significant differences in output. Pressure from outside sources, especially from those looking for a true marketable product will move this from a theoretical pursuit to a more applied pursuit. While this is not necessarily a bad thing it should not be the focus from the beginning.

• The simulator is primarily one of optimizing a schedule with limited resources and limited time. Look to other areas for inspiration and best practices i.e. network scheduling, airport flight scheduling and so forth.
APPENDIX A: STATISTICAL DISTRIBUTIONS

Uniform Distribution:

The uniform distribution is defined as the distribution of a random variable in which each value has the same probability of occurrence. This is also known as rectangular or continuous distribution. The resulting graph resembles a rectangular object, where each element occurs at least once uniformly.

Normal Distribution:

Normal distribution is a continuous distribution for any random variable with equal and finite mean, median and mode. It is also called “Gaussian distribution,” which produces a bell shaped curve. The normal distribution is generated using the following formula:

\[ f(x) = \frac{1}{\sqrt{2\pi \sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \]

Where

\( \mu = \text{Mean}, \sigma^2 = \text{Variance}, e = 2.7182 \)

Poisson Distribution:

Poisson distribution is a discrete random variable distribution, which gives the probability of a number of independent events occurring in fixed time. The Poisson distribution arises when you count a number of events across time or over an area. The Poisson distribution is defined using the formula:

\[ f(k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \]

\( \lambda = \text{Mean number of success in a given time period} \)

\( k = \text{Number of success we are interested in} \)

\( e = 2.7182 \)
APPENDIX B: MONEYWARE USERS GUIDE

The simulator implements a graphical user interface (GUI) that allows a quick set-up and simulation of tasks. The user has the option of entering their values in the text-boxes for Number of Tasks, Date Needed, Number of Iterations, Weeks in Iteration and Resources Available. Next the user will select one of the following SJF, FIFO, ITF, PTS or HBP from a list of radio buttons. Finally the user selects Size, Risk and Priority Levels using sliders or checks the default value check-box for each.

Once these steps are completed the user clicks on the Run Simulation button. Once the simulator runs it also opens a file directory box prompting the user to pick a location and name to save the output file. This file is saved as an xml file for use in Microsoft Excel. The user may then open up the completed file to view the results in raw data form. They may also view the results in different graph and table form.
BIBLIOGRAPHY


