OPTI-SOFT+: A RECOMMENDER FOR OPTIMAL SOFTWARE FEATURE SELECTION AND RELEASE PLANNING

by

Fernando Boccanera
Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy in
Information Technology

Committee:

________________________________	Dr. Alexander Brodsky, Dissertation Director
________________________________	Dr. Edward Huang, Committee Member
________________________________	Dr. Thomas LaToza, Committee Member
________________________________	Dr. Daniel Menascé, Committee Member
________________________________	Dr. Deborah Goodings, Associate Dean
Date:____________________________	Summer Semester 2022 George Mason University Fairfax, VA

Opti-Soft+: A Recommender for Optimal Software Feature Selection and Release Planning

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in in Information Technology at George Mason University

by

Fernando Boccanera
Master of Software Engineering
George Mason University, 1998
Bachelor of Computer Science
Campinas State University, Brazil, 1978

Director: Alexander Brodsky, Professor
Department of Computer Science

Summer Semester 2022
George Mason University
Fairfax, VA

DEDICATION

I dedicate this dissertation to my wife, who encouraged me to pursue the PhD and who supported me throughout all the years of my doctorate journey.

ACKNOWLEDGEMENTS

I wish to thank my committee members, Dr. Thomas LaToza, Dr. Edward Huang and Dr. Daniel Menascé, for their guidance and suggestions.

A special thanks to Dr. Alexander Brodsky, Committee Chairman, for his countless hours of advising, reviewing and for his immensurate patience throughout the entire process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AM	Analytical Module
BSN	Business Service Network
DGAL	Decision Guidance Analytics Language
DGS	Decision Guidance System
FLWOR	For, Let, Where, Order by and Return
IFM	Incremental Funding Method
JSON	JavaScript Object Notation
NPC	Net Present Cost
NPV	Net Present Value
PM	Performance Model
SN	Service Network

ABSTRACT

OPTI-SOFT+: A RECOMMENDER FOR OPTIMAL SOFTWARE FEATURE SELECTION AND RELEASE PLANNING

Fernando Boccanera, Ph.D.

George Mason University, 2022

Dissertation Director: Alexander Brodsky

Many software development projects fail completely or partially because they do not deliver much business benefit, that is, the return on the investment in the software is either zero or not enough to justify the investment. Knowing the business value of a potential software investment up front is a real challenge.

Multiple approaches have been proposed to increase the return on a software investment. However, they (1) don't have a valuation model, but instead rely on external stakeholders to provide valuation estimations, (2) are inaccurate, (3) are not reusable from one case to another, (4) require a high level of effort, and (5) are inflexible to changes.

To bridge this gap for the class of information systems that reduce the cost of the operations of a business process, developed in this dissertation is a Decision Guidance framework, called *Opti-Soft+*, to recommend a release schedule of software features that: (1) minimizes the combined cost of software development and improved business operations over the investment time horizon; and (2) outperforms the existing approaches.

Opti-Soft+ is unique in that it (1) is based on an accurate formal modeling of the BSN in terms of net present value as a function of the BSN configuration, which is enabled by the synergetic effect of multiple interrelated software features; and (2) completelyeliminates manual, time-consuming and often inaccurate estimation of the BSN cost reduction.

In order to develop the *Opti-Soft+* framework, the contributions of this dissertation include: (1) a formal analytic model that accurately computes the Net Present Value of both the software development and the improved business process operation over the time horizon of the investment; (2) a decision guidance system and methodology that codifies the formal analytic model and its inputs, formulates an optimization problem, solves it, recommends an optimal release schedule of software features and guides a decision-maker through the steps to setup and run the system; (3) methods for sensitivity analysis; and 4) an evaluation through a case study.

1. INTRODUCTION

1.1 <u>The Problem</u>

Many software development projects fail because they do not deliver much business benefit. Pucciarelli and Wiklund, [33], in a highly publicized report by the International Data Corporation (IDC), estimate that 25% of projects fail and another 25% do not provide any return on investment (ROI). The Standish Group, in their 2011 Chaos Report [42] had similar findings: 29% of projects were successful, 49% were challenged and 22% failed. Nine years later, the Standish Group Chaos Report for 2020 [41] still found only marginal improvements: 31% of projects were successful, 50% were challenged and 19% failed. Standish [42] also found that of the projects that do not fail, 45% of the functionality is never used, resulting in zero business value.

Failure factors such as not providing any ROI, or delivering software that is largely unused, are two sides of the same issue, which is that many projects do not seem to take into consideration the value, aka benefit, of the delivered software to the business that invested in it. Although many approaches have been developed to address this, they all have strong limitations and imprecisions.

The problem then is how to design an approach that delivers value back to the business that is commensurate to the investment, is accurate and has less limitations than the current approaches. Such a value-oriented approach would need to have the following characteristics: 1) a comprehensive model for understanding the investment environment, including the valuation factors, 2) an analytical method for determining business value with precision and another method for optimizing it, 3) a value-oriented software engineering methodology that uses the model and the method in order to continuously and iteratively make delivery decisions that generate the highest possible business value, and 4) addresses the limitations of current approaches.

This dissertation provides a solution to this problem for a class of investments in information systems that reduce the operations cost of a Business Service Network (BSN), where a BSN is a network of service-oriented components that are linked together to produce a business product. The solution, called *Opti-Soft+,* possesses all the characteristics of the value-oriented approach described above. In addition, it completely eliminates the effort to manually estimate or re-estimate valuation point, significantly reduces the effort to conduct 'what if' or sensitivity analysis and guides a decision maker through the steps to utilize the framework.

1.2 Research Gap

As the previous section and statistics demonstrate, software projects that are completed successfully and deliver a product that returns considerable value to the business are not the norm.

It is clear that in order to increase the rate of project success, the resulting software needs to deliver substantial business benefits, which requires a value-oriented approach. Historically, the first attempt to address business benefits was to treat software as an asset and include it in an organization's capital budgeting process. This required projects to be evaluated *ex-ante* through a business case, that is, prior to the decision to invest, and led to the development of a strong body of knowledge and methods [45], [18], [24], [23]. In these business-case oriented approaches, after the investment period ends, an evaluation is conducted, *ex-post,* in order to verify whether the benefits from the business case were actually realized. However, Lin et al. [25] and Song & Letch [40] found that *ex-ante* business case and *ex-post* evaluations of software investments did not materially improve the realization of business value.

One reason that business benefits were not realized is that often, the project execution team is not driven by them; there is a disconnect between decisions made on software functionality during project execution and the business case, creating a benefit gap. In his seminal work, Boehm [8] observed that the value-neutral software engineering practices of the time, seriously degraded project outcomes, and suggested that a value perspective should be integrated into the practices.

In the past two decades, there has been considerable research to address the benefit gap by adopting Boehm's suggestion of embedding business value considerations into the software development life cycle, resulting in the creation of many value-oriented approaches [4], [1], [22], [15], [27], [43]. These value-oriented approaches can be categorized as stakeholder-based, non-financial-based and financial-based.

1.2.1 Value-oriented Approaches: Stakeholder and Non-financial Based

In stakeholder-based approaches, the interest of stakeholders is assessed to determine the value of software functionality. Bagnall et al. [4] assigns weights to customers based on their importance to the organization, with the goal of finding a subset of customers whose requirements are to be satisfied within the available cost of the software release. AlBourae et al. [1] proposed a light re-planning model to select the most promising software features with the goal of achieving higher stakeholder satisfaction. The main technique used was stakeholders' voting.

Non-financial-based approaches use a non-financial metric as a proxy for value. One example is the Benefit Points method proposed by Hannay et al. [22]. The method assigns benefit points to user stories in a way similar to the assignment of story points in Agile practices. Benefit points link the user story to the business case but they are relative, not absolute and consequently cannot be compared across investments.

Both stakeholder-based and non-financial-based approaches are subjective, lack consistency in assigning values and consequently are of limited use in realizing business value.

1.2.2 Value-oriented Approaches: Financial-based

Financial-based approaches are the most widely used value-oriented methods [15],]27], [43] and are discussed next. They draw from the field of finance and commonly adopt Net Present Value (NPV) as a proxy to measure benefits of the investment in software, because NPV is widely used in capital budgeting to analyze the profitability of potential investments. NPV is the difference between the Present Value of cash inflows and the Present Value of cash outflows over a period of time. Present Value is the current worth of a future sum of money, given a specified rate of return. The rate used in capital budgeting is called the hurdle rate, which is defined as the minimum rate that an entity expects to earn from the investment. A pre-requisite to the calculation of NPV is the estimation of cash flows over a number of accounting periods within the investment horizon. The cash flow estimation can be conducted at many levels of granularity, with the lowest level being at the software requirement level.

Financial-based approaches are best suited to projects that deliver software in increments called releases so that the business benefit can be harvested after each release. The business benefit of each software functionality is evaluated and then used to make decisions during the release planning phases of the software development life cycle. We use the Agile term feature as synonym for software functionality or capability. A feature is a group of requirements that when implemented, expose a piece of functionality, a capability, that is beneficial to an end user.

Incremental Funding Methodology

The most popular and highly influential financial-based approach is the Incremental Funding Methodology (IFM) proposed by Denne and Cleland-Huang [15], [14]. IFM delivers software features as early as possible in order to maximize their business value. It assumes a software development life cycle that delivers software continuously and iteratively, consequently it can be utilized within the framework of modern Agile methodologies like Scrum. IFM associates cash flows (revenue and cost) to each software feature, but it does not provide any guidance on how to estimate the cash flows, it simply assumes that they are given.

F-EVOLVE*

Another financial-based approach is F-EVOLVE*, proposed by Maurice et al. in [27]. F-EVOLVE* is an iterative and evolutionary approach that facilitates the involvement of stakeholders to achieve increments (releases) that result in the highest degree of satisfaction among different stakeholders. Multiple stakeholders estimate the revenue of each software feature. Stakeholders are assigned a relative importance weight and the final revenue of each feature is the weighted average of the stakeholder estimates for the feature.

The approach considers features, revenues, releases and development resources. It provides decision support for the generation and selection of release plan alternatives, by formulating an Integer Linear Programming (ILP) problem with the goal of offering the most profitable sequence of features. It requires that the revenue as well as the cost of the software be given by stakeholders as a single combined cash flow. Like IFM, it does not provide any guidance on how to estimate the cash flows, it simply assumes that they are given.

van den Akker et al.

In [43], van den Akker et al. proposes an approach and tool for the selection of requirements for the next release, based on solving an ILP problem. The approach is very similar to F-EVOLVE*, with two differences; 1) it does not recommend a release plan for the entire software development project, it just recommends features for the next release; and 2) it does not consider multiple stakeholders, each revenue has only one estimation point.

The approach assumes that a release's best set of requirements is the set that results in maximum projected revenue against available resources in a given time period. It takes into account the total list of requirements, whether or not requirements are dependent on one another, a requirement's projected revenue, and a requirements resource claim per development team.

1.2.3 Limitations of Existing Value-based Approaches

A major limitation of all the current value-based approaches is that they do not have a method to estimate value points, they just expect that someone, e.g., stakeholders, provide the data points. This leads to inconsistency, subjectivity, and imprecision.

A second limitation that leads to imprecision is the need to draw a direct correlation between a benefit, like an increase in revenue, and a software feature. Such correlation is not trivial and researchers have acknowledged this difficulty, e.g., Devaraj and Kohli [16] noted that "the principal issue encountered is whether we can isolate the effect of IT on firm performance. It does not have an easy answer, because it means disentangling the effect of IT from various other factors such as competition, economic cycle, capacity utilization, and many other context-specific issues.".

A third limitation is that each and every unit of benefit like revenue, has to be mapped to one and only one software feature. This is not a realistic assumption because often, realizing a business benefit does require the implementation of more than one software feature. The result of this assumption is imprecision, because in situations where the benefit is caused by a group of features, the practitioner is forced to divide the value among the features.

A fourth limitation is that the approaches do not consider the cost of maintaining the software once it is implemented, leading to a major imprecision.

A fifth limitation is that the approaches require a considerable effort to estimate a large amount of value points. In an example with 4 releases, 3 stakeholders and 50 features, the number of estimation points in IFM and F-EVOLVE* would be 200, while in van den Akker et al. would be 600.

A sixth limitation is that the approaches are not flexible; they demand manual recalculations when any factor change such as labor rates, investment period, discount rate, etc. This inflexibility also requires a significant manual effort to conduct 'what if' or sensitivity analysis.

Value-oriented approaches like stakeholder and non-financial based are of limited use, so we do not consider them further. For the financial-based approaches, Table 1 summarizes their limitations.

Table 1 - Limitations of Value-Oriented, Financial-based Approaches

Limitation	IFM	F-EVOLVE*	van den Akker et al.
Characteristics that lead to imprecision	1) lack of valuation model, 2) granularity, 3) one-to-one mapping 4) discounting method 5) software maintenance cost not included	1) lack of valuation model, 2) granularity, 3) one-to-one mapping 4) discounting method 5) software maintenance cost not included	1) lack of valuation model, 2) granularity, 3) one-to-one mapping 4) discounting method 5) software cost not included
Valuation effort to set the method up (f=features, r=releases, s=stakeholders, c=factors)	High (f * r* c) points manually calculated	High (f * r * s * c) points manually calculated	High (f * c) points manually calculated
Valuation effort when factors change	Very high Manual recalculations needed when any factor change such as labor rates, discount rate, investment period, etc…	Very high Manual recalculations needed when any factor change such as labor rates, discount rate, investment period, etc…	Very high Manual recalculations needed when any factor change such as labor rates, discount rate, investment period, etc…
Effort to conduct 'what if' or sensitivity analysis	High require manual recalculations	High require manual recalculations	High require manual recalculations
Reason discounting causes imprecision	Done at the end of the release	Done at the end of the release	Not done

Section 2 provides more details of the Financial-based approaches.

1.3 Research Challenges

The previous section examined the limitations of the current value-oriented, financial-based approaches, and these limitations lead to the following research challenges:

1. How do we develop a formal valuation method that accurately computes the net present value of both the software development and the improved BSN operation as a function of (1) a release schedule of software features, (2) an implicitly defined set of feasible BSN configurations that are enabled by implemented software features, (3) software features that are required to enable each atomic BSN service, (4) software features dependency graph, and (5) a range of cost and operation parameters; where the method

closes the research gap of imprecision and the inability to map a benefit to more than one feature?

2. How do we develop a Decision Guidance System (DGS) and methodology that recommends a release schedule of software features that minimizes the combined cost of software development and the BSN operations over the investment time horizon; where the DGS closes the research gap of high effort to estimate valuation points?
3. How do we develop a method for sensitivity analysis of the normalized cost per unit of production, for a recommended release plan and associated improved BSN, as a function of the BSN throughput, where the method closes the research gap of inflexibility and requiring manual re-evaluations?
4. How do we develop an evaluation that demonstrates that *Opti-Soft+* performs better than the current approaches in some realistic cases?

1.4 Thesis Statement and Summary of Contributions

The limitations of the current approaches have shown that no approach is able to address the above challenges. One reason is that the approaches don't have a cost/benefit model of the investment environment, they just assume that external stakeholders provide the cost/benefit estimates. Developing such a model is challenging if the scope encompasses all possible systems, but it is possible for a particular domain. In this dissertation, that is the strategy that we adopt, that is, we propose a cost/benefit model and address the limitations and inaccuracy of the current approaches for a specific class of systems.

The dissertation focuses on a class of information systems that reduces the cost a Business Service Network (BSN), where a BSN is a network of service-oriented components that are linked together to produce a business product, for example, a BSN composed of 5 services, each performed by a person, that take a grant application as input and produces, as output, a decision on the grant.

1.4.1 Thesis Statement

For a class of information systems that automate and reduce the cost of operation of a Business Service Network (BSN), it is possible to develop a Decision Guidance (DG) software framework, called *Opti-Soft+,* to recommend a release schedule of software features that minimizes the combined cost of software development and the BSN operations over the investment time horizon. It is also possible to demonstrate that the framework outperforms the current approaches in terms of added business value and required effort. In lifting current limitations of value-based software release approaches, *Opti-Soft+* is unique in that it (1) is based on an accurate formal modeling of the BSN in terms of net present value as a function of the BSN configuration, which is enabled by the synergetic effect of multiple interrelated software features; and (2) completely eliminates manual, time-consuming and often inaccurate estimation of the BSN cost reduction. *Opti-Soft+* comprises of (1) a formal analytic model that computes the combined cost of the BSN and software development, given a candidate release schedule and an implicitly defined space of BSN configuration alternatives and the software features that enable them; (2) a decision guidance system that recommends an optimal release schedule, which is produced by converting a formal analytic model and a library of components into a mixed-integer linear programming (MILP) problem and solving it using an MILP solver; and (3) a methodology to recommend software release schedules that does not require manual estimations and is flexible to change valuation factors and conduct 'what if' or sensitivity analysis. *Opti-Soft+* moves the software investment benefit estimation from a manual, subjective approach to a systematic, model-based methodology that we believe results in considerably higher return on the software investment.

1.4.2 Summary of Contributions

The key contributions of this dissertation are as follows:

1. Formal Analytic Model: The design and development of a formal analytic model that accurately computes the net present value of both the software development and the improved BSN operation as a function of (1) a release schedule of software features, (2) an implicitly defined set of feasible BSN configurations that are enabled by the implemented software features, (3) software features that are required to enable each atomic BSN service, (4) software features dependency graph, and (5) a range of cost and operation parameters.

The analytical model is novel and unique in terms of the following: (1) it completely eliminates manual, time-consuming and often inaccurate estimation of the BSN cost reduction, by exposing valuation factors as parameters that can be easily changed prior to using the DGS to re-compute the recommendation; (2) it provides the equations to precisely compute the cost of the software and the business benefit due to improved BSN efficiency instead of estimating them as in the current approaches; (3) it leverages the insight that there is a direct correlation between the implementation of a set of software features and the improvement in the BSN operational efficiency, and the degree of improvement can be accurately calculated; (4) instead of estimating BSN improvement value per feature, which may not be realistic, the model estimates the value at the level of the business process, for the synergetic effect of features on the improved BSN configuration and operation; and (5) it provides a more accurate and granular (daily) cash flow discounting.

2. Decision Guidance System (DGS) and Methodology: The design and development of a DGS and a methodology, collectively called *OptiSoft*, to recommend a release schedule of software features that minimizes the combined cost of software development and the BSN operations over the investment time horizon. The methodology guides a decision-maker through the steps to setup and run the DGS to get a recommendation. The DGS uses the formal analytic model and the input of (1) an implicitly defined set of feasible BSN configurations that are enabled by implemented software features, (2) a set of software features required to enable each atomic BSN service, (3) a software feature dependency graph, and (4) a variety of cost and operation parameters. The DGS machine-generates a Mixed Integer Linear Programming (MILP) optimization problem and solves it using a commercial solver (CPLEX or Bonmin) and recommends the optimized Release Schedule to stakeholders. The DGS is unique and novel in the following ways: (1) it is based on an extensible repository of component models, which are written in a high level query language (JSONiq) rather than in a lower-level mathematical programming modeling languages such as AMPL, yet achieves the performance of commercial MILP solvers (2) it is modular, reusable and extensible, allowing development of new components that can easily integrate with existing ones; and (3) it significantly reduces the effort to conduct 'what if' or sensitivity analysis, because different scenarios and sensitivities can be easily analyzed by changing one or more parameters, rerunning the DGS and comparing the results.

3. **Methods for Sensitivity Analysis:** The development of methods of sensitivity analysis of the normalized cost per unit of production, for a recommended release plan and associated improved BSN, as a function of the BSN throughput. The methods help stakeholders make more informed decisions when cost factors may change.

4. **Initial Evaluation** that demonstrates that *Opti-Soft+* performs better than the current approaches in common situations where a benefit is driven by more than one feature. Given two features *A* and *B*, in cases where *Benefit(A, B)>Benefit(A)+Benefit(B), Opti-Soft+* correctly includes *Benefit(A,B)* in the total benefit while the current methods will, at the most, include *Benefit(A)+Benefit(B)*. In this case, *Opti-Soft+* outperforms the current methods by *Benefit(A,B) - (Benefit(A)+Benefit(B))*.

Table 2 shows how *Opti-Soft+* contributions compare with the current approaches. Additional comparisons are discussed in Section 9.

Table 2 *Opti-Soft* Contributions Comparing to Current Approaches

	IFM	F-EVOLVE*	van den Akker et al.	*Opti-Soft+*
Recommend an entire release schedule	✓	✓	Only next release	✓
Includes combined SW cost and business benefits	✓	✓	- (excludes SW)	
Applicable to general SW projects	✓	✓	✓	- For BSN only
Accurate formal model of business benefit as a function of SW features (vs. external manual valuation)	-	-	-	✓
Associate synergetic business value with multiple features (as opposed to silo features)	-	-	-	✓
Includes software maintenance cost beyond the release where it is developed	-	-	-	✓
Low effort, high flexibility and scalability to conduct valuations	-	-	-	✓
Support sensitivity analysis of recommendations	-	-	-	✓
Based on formal optimization (math programming based)	-	-	-	✓

1.5 Organization of the Dissertation

The rest of the dissertation is organized as follows: Section 2 discusses the work related to our proposed framework; Section 3 explains the framework's model; Section 4 presents the formal analytical model; Section 5 describes the Decision Guidance System and the methodology; Section 6 shows the approach through an example, Section 7 shows the sensitivity analysis technique, Section 8 discusses an evaluation, Section 9 compares the framework to the current approaches, and Section 10 concludes the dissertation.

2. RELATED WORK

In this section we start with a discussion of Agile Release Planning as a foundation to discuss the related financial-based approaches and then cover some of them.

2.1 Agile Release Planning

Agile Release Planning is a technique used in Agile software development, which is an approach where self-organizing teams collaborate with the customer to conduct requirements analysis, design, development and implementation in short, timeboxed and iterative periods where working software is delivered at the end of each period. Agile software development has the potential to deliver software that is actually useful to the customer because the customer is an integral member of the development team. Due to the interactive participation of the customer, Agile is also geared towards rapid response to changes.

The term was coined and popularized in the Agile Manifesto [19], which codifies four values and twelve principles. Principle number one states that "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software". This principle adopts Boehm's [8] suggestion of embedding business value considerations into the software development life cycle.

In Agile, usually requirements are placed in a product backlog. Through a planning process, product backlog items are removed from the backlog and assigned to timeboxed iterations. Some Agile approaches like Scrum have two types of timeboxes: sprints and releases. A sprint has a fixed number of weeks (usually 1 to 4) and a release contains a fixed number of sprints. There are at least two types of items in the product backlog: features and user stories. Features are at the granularity of the business, that is, they are a self-contained piece of functionality that are identifiable by the customer to have business value. Features are broken down into user stories, which are small units that can be developed in a few hours.

There are two types of planning in Agile: Release and Sprint. Release Planning is the process of assigning features to particular releases while Sprint Planning is the process of assigning user stories to sprints.

Both features and user stories have their size estimated, usually in points. An Agile point is a proxy for effort, that is, the higher the effort, the higher the number of points. For each timebox (sprint or release), the capacity of the team is estimated in number of points. The team's capacity is a function of the duration of the timebox, the number of developers and the average productivity of the developers. For example, in a team of 5 developers that can produce, on average, 2 points per day, the team's capacity for a sprint of 3 weeks is 5x2x15 or 150 points. The release capacity is a multiplier of the sprint capacity, for example, if a release has 4 sprints, then the release capacity in our example is 600 points.

The estimation of capacity is central to Agile, because of the constraint of a fixed timebox. Release Planning is then constrained by the release capacity, that is, no release can have more feature points than the capacity of the release. After Release Planning, both the duration and scope of the release is fixed.

Because decisions on which functionality to develop is made at Release Planning, this stage is a prime target for methods that improve business value harvesting including *Opti-Soft+*. In Scrum, the responsibility for benefit analysis is delegated to the Product Owner [35]. In reality, the Product Owner faces enormous challenges in determining business values, and the related approaches below try to address that.

2.2 <u>Related Financial-based Approaches</u>

In this section we discuss the following value-oriented, financial-based approaches that are somewhat related to ours:

1. Incremental Funding Method
2. F-EVOLVE*
3. van den Akker et al.
4. Continuous Value-based IT Project Steering

Approaches other than the four above were not considered because they are not comprehensive. For example, Riegel and Doerr [34] developed heuristics that can be used to optimize requirements selection, but their cost metric only involves elicitation, not development. Hannay et al. [22] used benefit points as a metric for business value but did not propose a release scheduling approach. Elsaid et al. [17] used rule-based fuzzy logic to prioritize requirements but did not consider the development cost.

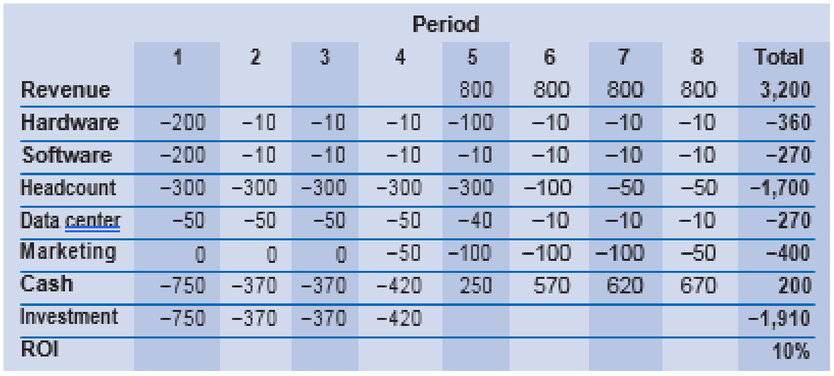
2.2.1 Incremental Funding Method

The first value-oriented, financial-based approach related to *Opti-Soft+* that we illustrate, is the highly popular and influential approach called Incremental Funding Methodology

(IFM). Proposed by Denne and Cleland-Huang [15], [14], [12], IFM's approach is to deliver software features as early as possible in order to maximize their business value or return. It assumes a software development life cycle that delivers software continuously and iteratively, consequently it can be utilized within the framework of modern Agile methodologies like Scrum. It also assumes that the business value of software, expressed in terms of cash flows, is provided externally by a market analysis that precedes the software development project, that is conducted as part of the business case.

In [15], the authors provide an example to demonstrate how the early delivery of software features increases the positive cash flow. Table 3 shows the cash flows in a traditional waterfall Software Development Life Cycle (SDLC) where the software is delivered at the end. The project lasts one year while the investment period is two years, and the accounting period is one quarter. The project requires an investment of $1.91M, it starts producing revenue after 4 quarters when it ends. At the end of the two years investment period, it returns $200K in undiscounted profit, with a ROI of 10%.

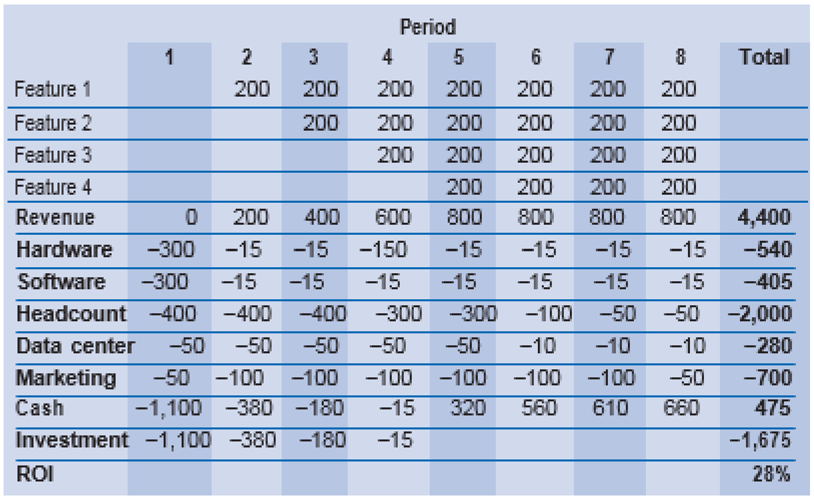
Table 3 - Cash flow analysis for a software development project using traditional feature delivery as presented in [15]



	Period								
	1	2	3	4	5	6	7	8	Total
Revenue					800	800	800	800	3,200
Hardware	−200	−10	−10	−10	−100	−10	−10	−10	-360
Software	−200	−10	−10	−10	−10	−10	−10	−10	-270
Headcount	−300	−300	−300	−300	−300	−100	−50	−50	-1,700
Data center	−50	−50	−50	−50	−40	−10	−10	−10	-270
Marketing	0	0	0	−50	−100	−100	−100	−50	-400
Cash	−750	−370	−370	−420	250	570	620	670	200
Investment	−750	−370	−370	−420					-1,910
ROI									10%

Table 4 shows the cash flows if the application is partitioned into four equally valuable features released independently during the first four quarters. Because revenue starts accruing after the first quarter, when Feature 1 is delivered, the resulting profit at the end of eight quarters is \$475K, the investment is \$1,675K and the ROI is 28%. The table clearly shows that from a value perspective, delivering software in increments provides significantly more financial benefits because value is harvested sooner than in the Waterfall SDLC.

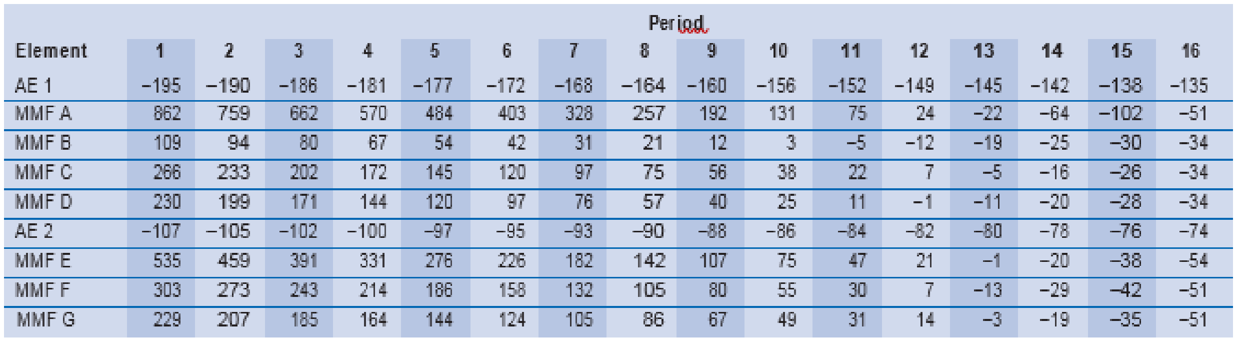
Table 4 - Cash flow analysis for the same software development project using incremental feature delivery as presented in [15]



	Period								
	1	2	3	4	5	6	7	8	Total
Feature 1		200	200	200	200	200	200	200	
Feature 2			200	200	200	200	200	200	
Feature 3				200	200	200	200	200	
Feature 4					200	200	200	200	
Revenue	0	200	400	600	800	800	800	800	**4,400**
Hardware	−300	−15	−15	−150	−15	−15	−15	−15	**−540**
Software	−300	−15	−15	−15	−15	−15	−15	−15	**−405**
Headcount	−400	−400	−400	−300	−300	−100	−50	−50	**−2,000**
Data center	−50	−50	−50	−50	−50	−10	−10	−10	**−280**
Marketing	−50	−100	−100	−100	−100	−100	−100	−50	**−700**
Cash	−1,100	−380	−180	−15	320	560	610	660	**475**
Investment	−1,100	−380	−180	−15					**−1,675**
ROI									**28%**

The IFM calls business features Minimum Marketable Features (MMFs) and technical features are Architectural Elements (AEs) and it takes into account dependencies among MMFs and AEs. It estimates the software development cost and the feature financial benefit separately and then combines them into a single stream of cash flows, that is, the software development cost has to be apportioned among the set of features. It then discounts the cash flow of each feature and calculates the total NPV. Next, it takes each feature total NPV and adjusts it to each accounting period. This is called Sequence Adjusted NPV (SANPV). Table 5 shows the example from [15], for a project with a four year investment period. In the example, AE1 is a precursor to MMF A, C and D, MMF A precedes MMF B, AE2 is a precursor to MMF E and F, and MMF G has no dependency. Also, MMF A requires two periods to be implemented while all others require only one period. If MMF A is developed in quarter 1, its SANPV is $862K, while if it is developed in quarter 8, it is only 257K.

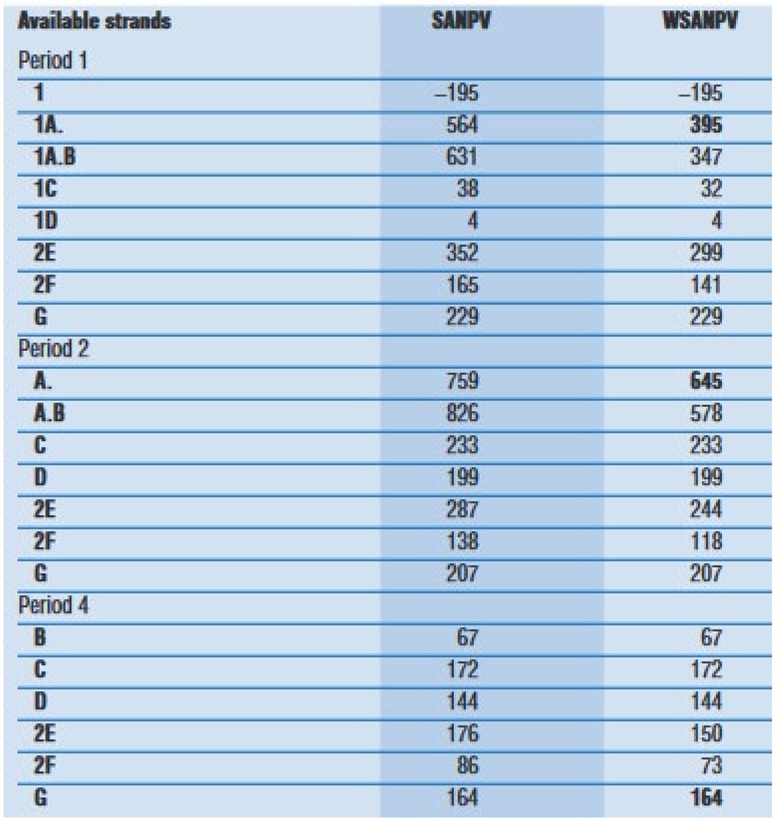
Table 5 - Sequence Adjusted NPV Example as presentedin [15]



	Period															
Element	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
AE 1	−195	−190	−186	−181	−177	−172	−168	−164	−160	−156	−152	−149	−145	−142	−138	−135
MMF A	862	759	662	570	484	403	328	257	192	131	75	24	−22	−64	−102	−51
MMF B	109	94	80	67	54	42	31	21	12	3	−5	−12	−19	−25	−30	−34
MMF C	266	233	202	172	145	120	97	75	56	38	22	7	−5	−16	−26	−34
MMF D	230	199	171	144	120	97	76	57	40	25	11	−1	−11	−20	−28	−34
AE 2	−107	−105	−102	−100	−97	−95	−93	−90	−88	−86	−84	−82	−80	−78	−76	−74
MMF E	535	459	391	331	276	226	182	142	107	75	47	21	−1	−20	−38	−54
MMF F	303	273	243	214	186	158	132	105	80	55	30	7	−13	−29	−42	−51
MMF G	229	207	185	164	144	124	105	86	67	49	31	14	−3	−19	−35	−51

Once the SANPV is determined, the IFM then chooses the sequence of features. It employs a heuristic algorithm that uses a simple look-ahead approach. The algorithm views sequencing options as strands, which are sequences of features linked by dependencies. In step 1, it starts with strands that reflect the dependency graph with the first feature of each strand being a feature that has no dependency. In the example, the sequence 1A., 1A.B, A.B, 1C, 1D, 2E, 2F and G, where the dot after A means that MMF A requires two periods to be developed. It then calculates the combined SANPV of the step 1 strands for period 1; for strand 1A.B (feature AE 1 is developed in period 1, A in periods 2 and 3, B in period 4) , the SANPV=-\$195K+\$759K+\$67K=\$631K. Next it calculates the weighted SANPV with the formula WSANPV=SANPV(1-WF(P-1)) where WF is the weighting factor and P is the number of periods needed to develop the strand. The reason is that earlier developed strands leave more time to develop other valuable features. WF is based on an empirical study. For sequence 1A.B, the WSANPV is (\$631K(1-(0.15(4-1)) or \$347K. Then it chooses the strand with the highest NPV for period 1, which according to Table 6, is 1A., which means that technical feature AE1 is developed in period 1.

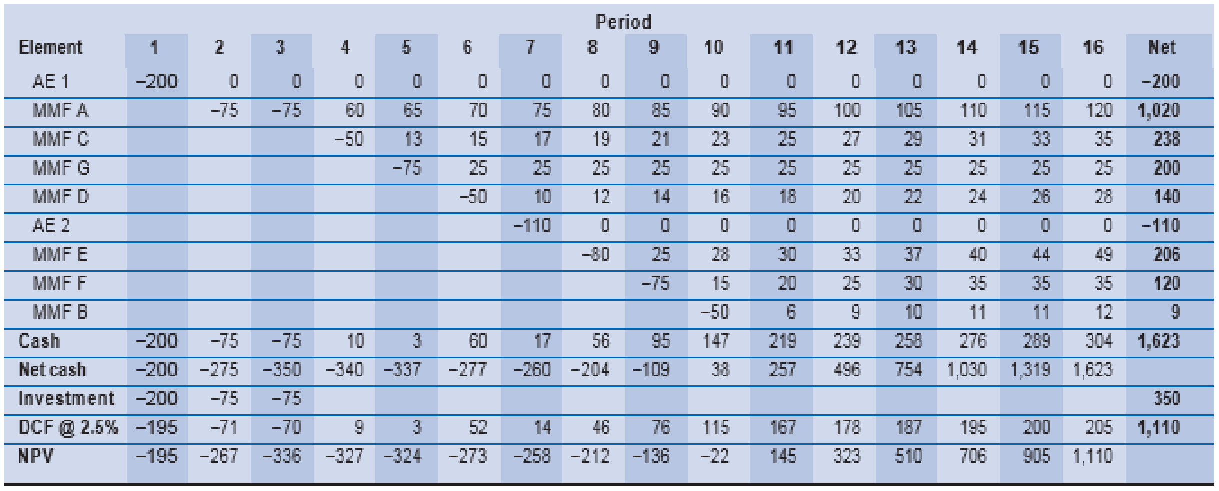
Table 6 – WSANPV for Strands for Periods 1, 2 and 4



Available strands	**SANPV**	**WSANPV**
Period 1		
1	–195	–195
1A.	564	**395**
1A.B	631	347
1C	38	32
1D	4	4
2E	352	299
2F	165	141
G	229	229
Period 2		
A.	759	**645**
A.B	826	578
C	233	233
D	199	199
2E	287	244
2F	138	118
G	207	207
Period 4		
B	67	67
C	172	172
D	144	144
2E	176	150
2F	86	73
G	164	**164**

In step 2, it removes "1" from the step 1 strands and produces the second set of strands: A., A.B, C, D, 2E, 2F, G. It then chooses the strand with the highest WSANPV, which is A., that is, MMF A is developed in periods 2 and 3. The process is repeated until the sequence contains all features. In the example, the heuristic selects sequence 1ACGD2EFB and Table 7 shows the NPV of the selected sequence, which is $1.1M. Note that an IFM sequence is basically a release plan, where each letter represents the name of the feature, and the letter position is the release (period) where the feature is developed.

Table 7 - NPV of the sequence 1ACGD2EFB as presentedin [15]



	Period																
Element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Net
AE 1	−200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**−200**
MMF A		−75	−75	60	65	70	75	80	85	90	95	100	105	110	115	120	**1,020**
MMF C				−50	13	15	17	19	21	23	25	27	29	31	33	35	**238**
MMF G					−75	25	25	25	25	25	25	25	25	25	25	25	**200**
MMF D						−50	10	12	14	16	18	20	22	24	26	28	**140**
AE 2							−110	0	0	0	0	0	0	0	0	0	**−110**
MMF E								−80	25	28	30	33	37	40	44	49	**206**
MMF F									−75	15	20	25	30	35	35	35	**120**
MMF B										−50	6	9	10	11	11	12	**9**
Cash	−200	−75	−75	10	3	60	17	56	95	147	219	239	258	276	289	304	**1,623**
Net cash	−200	−275	−350	−340	−337	−277	−260	−204	−109	38	257	496	754	1,030	1,319	1,623	
Investment	−200	−75	−75														350
DCF @ 2.5%	−195	−71	−70	9	3	52	14	46	76	115	167	178	187	195	200	205	**1,110**
NPV	−195	−267	−336	−327	−324	−273	−258	−212	−136	−22	145	323	510	706	905	1,110	

The IFM innovated because it 1) combined the revenue and the cost of software development at the feature level, 2) provided an algorithm for selecting a release plan and 3) demonstrated that the earlier a feature is released, the higher the NPV. However, its application to projects with a large number of features is problematic because the number of calculations grows exponentially. Also, IFM does not guarantee that the optimal sequence is produced by the method.

2.2.2 F-EVOLVE*

In [36], Ruhe and Ngo-The propose EVOLVE*, an iterative and evolutionary approach that facilitates the involvement of stakeholders to achieve increments (releases) that result in the highest degree of satisfaction among different stakeholders. In [27], Maurice et al. extend EVOLVE* by proposing F-EVOLVE*. In F-EVOLVE*, instead of voting, stakeholders are asked to estimate the financial value of each software feature.

F-EVOLVE* provides decision support in the generation and selection of release plan alternatives. The approach formulates an Integer Linear Programming (ILP) problem with the goal of offering the most profitable sequence of features.

Given a set of features $\{F_{1,\dots} F_n\}$, a set of stakeholders $\{S_{1,} \dots S_q\}$, relative importance λ_p for each S_p, and K releases, *NPV(i,k,p)* as the net present value of a cost estimate for feature *i* in release *k* by S_p, the objective function is

$$\sum_{i=1}^{n}\sum_{k=1}^{K}\sum_{p=1}^{q} \lambda_p \, NPV(i,k,p)\, x(i,k)$$

Where: *x(i,k)* is 1 if F_i is selected in release *k* or 0 otherwise

F-EVOLVE* has the following characteristics:

1. Multiple stakeholders estimate the revenue of each software feature. Stakeholders are assigned a relative importance weight and the final revenue of each feature is the weighted average of the stakeholder estimates for the feature.
2. Revenue estimates are projected over a fixed time horizon after the corresponding feature is implemented. Changes in the time horizon would require re-estimating each revenue.
3. The ILP formulation:
    - Considers features, revenues, releases and development resources
    - Maximizes the aggregate NPV
    - Uses binary decision variables *x(i,k)* to map features to releases
    - Produces a release plan by instantiating the binary decision variables, subject to the following constraints:

        - Development team capacity
        - Dependencies among features

4. Provides a decision support system that:
    - Solves the ILP problem
    - Generates a set of alternative solutions from which the stakeholder can make a decision.

2.2.3 van den Akker et al. Approach

In [43], van den Akker et al. propose an approach and tool for the selection of requirements for the next release based on solving an ILP problem. The approach is very similar to F-EVOLVE*, with two differences; 1) it does not recommend a release plan for the entire software development project, it just recommends features for the next release; and 2) it does not consider multiple stakeholders, each revenue has only one estimation point.

The approach assumes that a release's best set of requirements is the set that results in maximum projected revenue against available resources in a given time period. It takes into account the total list of requirements, whether or not requirements are dependent on one another, a requirement's projected revenue, and a requirement's effort to develop. Given a set of requirements {$R_1, R_2, \dots R_n$} and their corresponding revenues {$v_1, v_2, \dots v_n$}, the ILP formulation is shown below:

$$max \sum_{j=1}^{n} v_j x_j$$

$$subject\ to \sum_{j=1}^{n} a_j x_j \leq d(T)Q$$

$$x \in \{0,1\}$$
$$x_j \leq x_k \; if \; R_j \; is \, dependent \; on \; R_k$$

Where:

x_j is 1 if R_j is selected or 0 otherwise

a_j is the number of man-days to implement R_j

T is the development period

d(T) is the number of working days in the development period.

Q is the number of persons working in the development team

d(T)Q is the development team capacity in man-days.

The recommendation is a set of requirements for the next release that optimizes the total revenue. In [44], the authors extend the original model to cover revenue-based dependencies, i.e., a case where a set of requirements increase or decrease the revenue only when they are combined in a package. Because the individual revenue of each requirement in the package have already been considered, the extension reflects only the remaining value. The extension models the case of increasing revenue separately from the case of decreasing revenue. It adds one term to the objective function for the increase in revenue and another for the decrease. It also adds two additional constraints, one for the increase case and another for the decrease case.

The problem with this extension to handle revenue-based dependencies is that "if part of the additional revenue is already obtained when a subset of the package is implemented, the model becomes much more complex a.o. because we would have to define variables for all subsets. If the packages are not disjoint, the additional revenue might be reduced if two 'overlapping' packages are selected, which again complicates the model.” The lack of independency between packages leads to imprecision because of double counting.

2.2.4 Continuous Value-based IT Project Steering

The Continuous Value-based IT Project Steering approach by Fridgen et al. [21] has the following steps:

1. Definition of project objectives
2. Derivation of specific requirements
3. Ex-ante evaluation and aggregation of cash flows for each requirement.
4. Continuous project success measuring and controlling
5. Ex-post measurement of cash flow realizations

The approach assumes that in step 3, cash flows *cf(it)* are initially evaluated for each one of the requirements i in each period t with $t = 0 \ldots T$. The method uses an interval-based scheme for the evaluation of each cash flow. Because cash flows are assumed to be normally distributed, the approach is able to derive expected value μ_{it} and standard deviation σ_{it} for each requirement r_t in each period t, which is then used to calculate the distribution parameters of the Net Present Value for each requirement. The overall value of the project is the aggregate NPV of all requirements. The standard deviation is used to measure risk. Using these parameters, various methods of an integrated risk/return can be applied to make an investment decision.

In step 4, at the end of every development period, the NPV is recomputed to account for the fact that the cash flow for the requirements just implemented are not random variables anymore because they are known with certainty. The approach then calculates a metric called Project Success Measuring (PSM), which can be used to investigate the deviations from the *ex-ante* business case.

In step 5, at the end of the project, the *ex-post* measurement of the total benefit is compared with the *ex-ante* expected value, to gain insights that can be used in future projects.

The authors claim that "treating cash flows as random variables clearly eases their estimation, as no decision maker has to commit to exact values". The approach might provide a better treatment of the cash flows estimation, but they are still very subjective. Also, it does not provide requirement prioritization, so its usefulness is largely diminished by not influencing the order of the requirements. The PSM metric can be used to stop the project if all high-value requirements have already been implemented, but this is the only value-based decision allowed by the approach during project execution.

2.2.5 Comparison of Related Approaches

Table 8 summarizes the characteristics of each approach described above.

Table 8 - Characteristics of the Related Approaches

Characteristic	IFM	F-EVOLVE*	van den Akker et al.	4. Continuous Value-based IT Project Steering
Business benefit metric	Total NPV	Total NPV	Total Revenue	NPV
Built in valuation model	No, value estimates are given	No, value estimates are given	No, value estimates are given	No, value estimates are given
Value metric	Projected Cash Flow	Projected Revenue	Projected Revenue	Projected Cash Flow
Number of value estimation points	#features * #releases	#features * #stakeholders * #releases	#features	#features * #releases

Characteristic	**IFM**	**F-EVOLVE***	**van den Akker et al.**	**4. Continuous Value-based IT Project Steering**
Granularity of value estimation	Feature	Feature	Feature	Feature
Mapping of feature to an estimation point	One-to-one	One-to-one	One-to-one Many-to-many is handled in a complex way	One-to-one
Value estimation incorporates software development	Yes	Yes	No	No
Time horizon is built into estimations	Yes	Yes	Yes	Yes
Discounting method	Discount at the end of each release period	Discount at the end of each release period	None	Discount at the end of each release period
Release schedule recommendation	For the entire software project	For the entire software project	Only for the next release	Does not recommend a schedule.
Recommendation method	Heuristics	ILP	ILP	Total NPV is used ex-ante to make a binary investment decision
Supports feature dependencies	Yes	Yes	Yes	No
Release size is constrained by the seize of feature	No more than 1 feature per release	Yes	Yes	N/A

Note that some of the characteristics in the previous table are actually limitations, for example, the second row shows that no approach has a built-in valuation model, they all assume that value estimates are given.

Table 9 Summarizes the limitations.

Table 9 - Summary of Limitations of Related Approaches

Limitation	IFM	F-EVOLVE*	van den Akker et al.	Continuous Value-based IT Project Steering
Characteristics lead to imprecision	1) lack of valuation model, 2) granularity, 3) one-to-one mapping 4) discounting method 5) software maintenance cost not included	1) lack of valuation model, 2) granularity, 3) one-to-one mapping 4) discounting method 5) software maintenance cost not included	1) lack of valuation model, 2) granularity, 3) one-to-one mapping 4) discounting method 5) software cost not included	1) lack of valuation model, 2) granularity, 3) one-to-one mapping 4) discounting method 5) software cost not included
Valuation effort to set the method up (f=features, r=releases, s=stakeholders, c=factors, p=processes)	High, (f * r* c) points manually calculated	High, (f * r * s * c) points manually calculated	High, (f * c) points manually calculated	High, (f * r * c) points manually calculated
Valuation effort when factors change	Very high. Manual recalculations needed when any factor change such as labor rates, discount rate, investment period, etc…	Very high. Manual recalculations needed when any factor change such as labor rates, discount rate, investment period, etc…	Very high. Manual recalculations needed when any factor change such as labor rates, discount rate, investment period, etc…	Very high. Manual recalculations needed when any factor change such as labor rates, discount rate, investment period, etc…
Effort to conduct 'what if' or sensitivy analysis	High, require manual recalculations	High, require manual recalculations	High, require manual recalculations	N/A
Reason discounting causes imprecision	Done at the end of the release	Done at the end of the release	Not done	Done at the end of the release

As the above table shows, a major limitation of all the current value-based approaches is that they do not have a method to estimate value points, they just expect that someone, e.g., stakeholders, provide the data points. This leads to inconsistency, subjectivity, and imprecision.

A second limitation that leads to imprecision is the need to draw a direct correlation between a benefit, like an increase in revenue, and a software feature. Such correlation is not trivial and researchers have acknowledged this difficulty, e.g., Devaraj and Kohli [16] noted that "the principal issue encountered is whether we can isolate the effect of IT on firm performance. It does not have an easy answer, because it means disentangling the effect of IT from various other factors such as competition, economic cycle, capacity utilization, and many other context-specific issues.".

A third limitation is that each and every unit of benefit like revenue, has to be mapped to one and only one software feature. This is not a realistic assumption because often, realizing a business benefit does require the implementation of more than one software feature. The result of this assumption is imprecision, because in situations where the benefit is caused by a group of features, the practitioner is forced to divide the value among the features.

A fourth limitation is that the approaches do not consider the cost of maintaining the software once it is implemented, leading to a major imprecision.

A fifth limitation is that the approaches require a considerable effort to estimate a large amount of value points. In an example with 4 releases, 3 stakeholders and 50 features, the number of estimation points in IFM and F-EVOLVE* would be 200, while in van den Akker et al. would be 600.

A sixth limitation is that the approaches are not flexible; they demand manual recalculations when any factor change such as labor rates, investment period, discount rate, etc. This inflexibility also requires a significant manual effort to conduct 'what if' or sensitivity analysis.

In the subsequent sections, we propose a framework to address the limitations of the current related approaches.

3. *OPTI-SOFT*+ MODEL OVERVIEW

This section is an overview of the proposed approach hypothesized in the Thesis Statement. The approach is called *Opti-Soft*+, and we show that it addresses the challenges described in Section 1.2.

The accuracy of a method to recommend an optimal software delivery schedule can be increased if the method is geared towards a class of software projects where the benefit, as measured by the cash flows, can be isolated from other factors that influence the financial performance of the organization. Such an isolation is possible in information systems that improve the efficiency of a business process. A business process consumes inputs and produces a product as its output, and the cost of a unit of the product is mostly a function of the labor rates of the workstation workers and the time that each workstation takes to produce the product.

Taking this concept one step further, the financial benefit of a software feature that improves a business process can be determined by calculating the cost of the business process before and after the software implementation. This means that there is a direct correlation between a software feature and the business process improvement, and the degree of improvement can be accurately calculated.

Together, the isolation and correlation properties above of information systems basically say that the implementation of software features leads to more efficient business processes, that is, the efficiency is a direct consequence of the availability of software features. This efficiency is gained due to a reduction of the time a worker spends, or the elimination of a portion of the process, or the utilization of workers with a lower labor rate.

The proposed framework, *Opti-Soft+,* recognizes the isolation and correlation properties and takes advantage of them in order to calculate the potential business value of software investment alternatives, and recommend a release plan, among many candidates, that maximizes the investment

The uniqueness of the *Opti-Soft+* framework is its accurate computation of the value of the improvement, by formally modeling the Business Process Network (BPN) and its associated costs over the investment time horizon, as a function of the software release schedule. in *Opti-Soft+,* the estimation of the business value of software features is not external, but instead, is at the heart of its cost/benefit model. The framework:

1. Has a built in, formal analytical model that encompasses all aspects of the investment domain (business process, software development process, space of candidate release plans).
2. Provides the equations to precisely compute the cost of the software and the business benefit, instead of estimating them as in the current approaches.
3. Addresses the challenge of allocating value units by allocating value not at the level of a feature, but at the level of a process.
4. Guides a decision maker through the steps to get a recommendation
5. Formulates and solves the MILP

6. Recommends an optimal release plan and business process configuration that maximizes the net present value of the business cost aggregated with the software development cost

Characteristics 1, 2 and 3 are novel and unique. Also novel and unique are the following benefits:

- Significantly improves the precision of valuation points.
- Completely eliminates the effort to manually estimate or re-estimate valuation points by exposing valuation factors as parameters that can be easily changed and recomputed
- Significantly reduces the effort to conduct 'what if' or sensitivity analysis. Different scenarios and sensitivities can be easily analyzed by changing one or more parameters, rerunning the DGS and then comparing the results.
- Improves the precision of valuations even further by discounting monies on the day that cash flows are incurred, instead of at the end of the release period. This is accomplished by using a pay schedule.

We now provide an overview of the framework.

3.1 Business Cost

In order to maximize the business value from a software investment, we need to estimate the cost of software development as well as the business benefit of the implementation of the software. For a class of software projects that implement information systems that improve a business process, the benefit of the software is the cost savings of the improvement.

A business process consumes input flows, e.g., applications, and produces output flows, e.g., grants. The cost associated with the business process is a function of the cost drivers such as labor rates and time spent. This means that the benefit (savings) of an investment in a software feature that improves a business process can be determined by the difference in the cost of the process before and after the improvement induced by the utilization of the software feature.

The above insight, that the implementation of a software feature allows the adoption of more efficient business process networks (BPN) is key to *Opti-Soft+*, because each new, more efficient BPN configuration can be identified, and its cost measured with precision. In Figure 1, we have an initial BPN configuration, called BPN_0 that can benefit from automation and has a Net Present Cost $NPC(BPN_0)$. A cash investment $NPC(SW_1)$ is made to implement software features SW_1 in the first release (r=1). After release 1, the availability of the software features SW_1 allows process improvements, so BPN_0 transitions to BPN_1, which has $NPC(BPN_1)$, which is lower than $NPC(BPN_0)$. The procedure continues iteratively, with each investment $NPC(SW_r)$ in release r causing the BPN_{r-1} to transition to BPN_r, resulting in a lower $NPC(BPN_r)$.

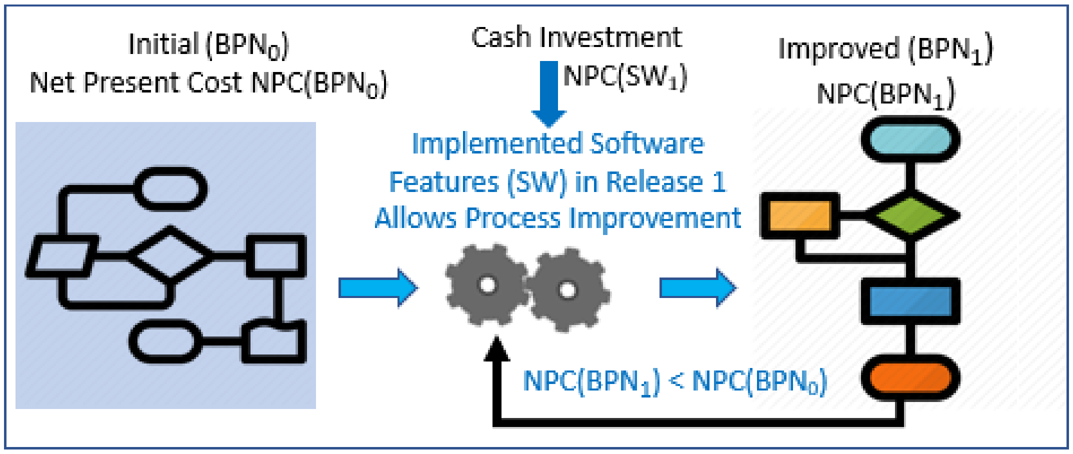

Initial (BPN_0)
Net Present Cost $NPC(BPN_0)$
Cash Investment
$NPC(SW_1)$
Implemented Software Features (SW) in Release 1 Allows Process Improvement
Improved (BPN_1)
$NPC(BPN_1)$
$NPC(BPN_1) < NPC(BPN_0)$

Figure 1 - BPN Cost Reduction due to the Investment in Software Features

In order to calculate and optimize the cost savings, we need to model the BNP transitions as well as the software features that enable them. In the following sections we explain the underlying model for the BPN, its cost/benefit model and the cost/benefit model for the software development.

3.2 BPN Modeling

To explain the BPN model, let's start with the mechanics of a business process. Intuitively, a business process is like a factory, where raw materials are incrementally transformed into a finished product. A business process consumes inputs and produces a product and the cost of a unit of the product is a function mostly of the labor rates of the workstation workers and the time that each intermediate workstation takes to produce intermediate products. This means that the cost, or cash outflow, of a BPN can be accurately determined by computing its labor and any other non-labor cost.

We are interested in modeling the financial benefit, using cash flow as a proxy, of a business process. The total benefit is the delta between the cash flow of the business process before the system, the As-Is, and after, the To-Be. Because we want to influence the software release schedule, that is, the sequence that the software features are implemented, we need to consider the cash flow not at the end of the project, but at the end of each release. This is because different release schedules produce different cash flows.

Consider a business process that takes as input a paper application and produces as output an application decision. This simple process takes one single input and produces one single output, although more generically, processes can have multiple inputs and outputs. In this case, the process is driven by the input and there is a one-to-one relationship between the input and the output. The unit cost, or cost outflow, is easily computed by multiplying the labor rate by the time it takes to process a single application and adding any other non-labor cost.

In the area of finance, the analysis of investment alternatives usually employs NPV as the benchmark metric. The NPV of the As-Is BPN can be easily calculated by multiplying the unit cost of each process by the number of times the process is executed, summing and discounting. The same calculation applies to the To-Be BPN.

Now that we know how to accurately compute the NPV of a process, let's examine the link between a business process and the software features that make possible the process to exist. In an incremental SDLC, each software feature is assigned to a specific release and each release implements the assigned features and delivers them in the form of business functionality. When a release is deployed, it can have a positive impact in the cost of the business process. By positive impact we mean a reduction of the average labor rate and/or a reduction of the time to produce one unit of the product or the use of workers with a lower labor rate.

The *Opti-soft+* approach uses the reduction in the labor cost to precisely calculate the financial benefit of a software feature, as opposed to previous approaches that assume the value of the benefit is given. To leverage this labor reduction, the initial BPN, the As-Is, needs to change at the end of each release to take advantage of the software features just implemented. This means that the BPN goes through several configurations until it reaches the final, To-Be configuration.

The above insight, that the implementation of software features allows the usage of more efficient business process networks is key to *Opti-Soft+,* because each new BPN configuration can be modeled, and its cost measured and associated to a set of features. Note that in this approach, there is no need to estimate the cost of individual features, a feature is just a mechanism that triggers a change in the BPN configuration, while cost is precisely calculated at the level of the BPN.

3.2.1 BPN Transition Based on Software Feature Implementation

We utilize a Service Network representation, as described in [11], to model the BPN. A Service Network is a "network of service-oriented components that are linked together to produce products". Services can be composite, that is, have subservices or atomic, that is, indivisible. We use the term Business Service Network (BSN) to refer to the representation of a BPN as a Service Network.

We now show intuitively how processes called services, are modeled in a generic way that allows them to be combined into a service network that can evolve with the implementation of software features. We use an example of a BSN that takes a user application as input, adjudicates the application and produces either a notice or a letter. For this example, Figure 2 shows the initial BSN, the As-Is, which is composed of three services in a sequence: AA, BA, CA. We call this configuration BSN_0 and its Net Present Cost (NPC) is $NPC(BSN_0)$.

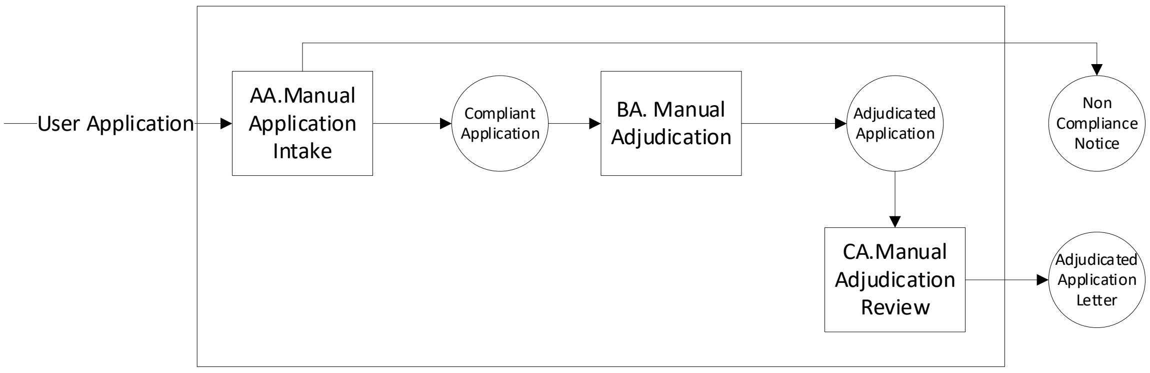


Figure 2 – Initial Application Adjudication BSN_0

Suppose also that feature F1 allows process, aka service, AA (Manual Application Intake) to transition to AB (Electronic Application Intake), F2 allows BA (Manual Adjudication) to transition to BB (Electronic Adjudication), F3 allows CA (Manual Adjudication Review) to transition to CB (Electronic Adjudication Review) and F4 allows AB to transition to AC (Self-service Application Intake). These dependencies between services and features are summarized in Table 10.

Table 10 - Required Features to Activate a Service

Service	Required Feature
AA	
AB	F1
AC	F4
BA	
BB	F2
CA	
CB	F3

If F1, F3, F2 and F4 are implemented in releases 1, 2, 3 and 4 respectively, then after release 1, BSN_0 (AA, BA, CA) transitions to BSN_1 with subprocesses AB, BA, CA and NPC(BSN_0) changes to NPC(BSN_1). After release 2, BSN_1 transitions to BSN_2 with subprocesses AB, BA, CB and NPC(BSN_1) changes to NPC(BSN_2). After release 3, BSN_2 transitions to BSN_3 with subprocesses AB, BB, CB and NPC(BSN_2) changes to NPC(BSN_3). After release 4, BSN_3 transitions to BSN_4 with subprocesses AC, BB, CB and NPC(BSN_3) changes to NPC(BSN_4). These transitions are shown in Table 11.

Table 11 - Application Adjudication BSN Configuration After Each Release

Release	BSN configuration	BSN subprocesses	Implemented Feature	NPC
Before Rel 1 (As-Is)	BSN_0	AA, BA, CA		NPC(BSN_0)
After Rel 1	BSN_1	AB, BA, CA	F1	NPC(BSN_1)
After Rel 2	BSN_2	AB, BA, CB	F3	NPC(BSN_2)
After Rel 3	BSN_3	AB, BB, CB	F2	NPC(BSN_3)
After Rel 4 (To-Be)	BSN_4	AC, BB, CB	F4	NPC(BSN_4)

Because each subsequent BSN is more efficient than the previous one, each NPC(BSN$_r$) is less than its predecessor NPC(BSN$_{r-1}$). The final BSN, the To-Be, is AC, BB, CB, which is illustrated in Figure 3.

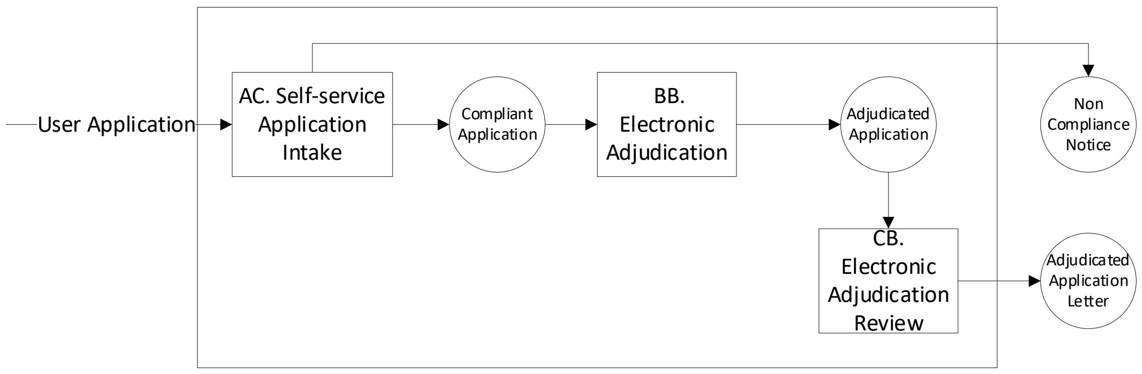
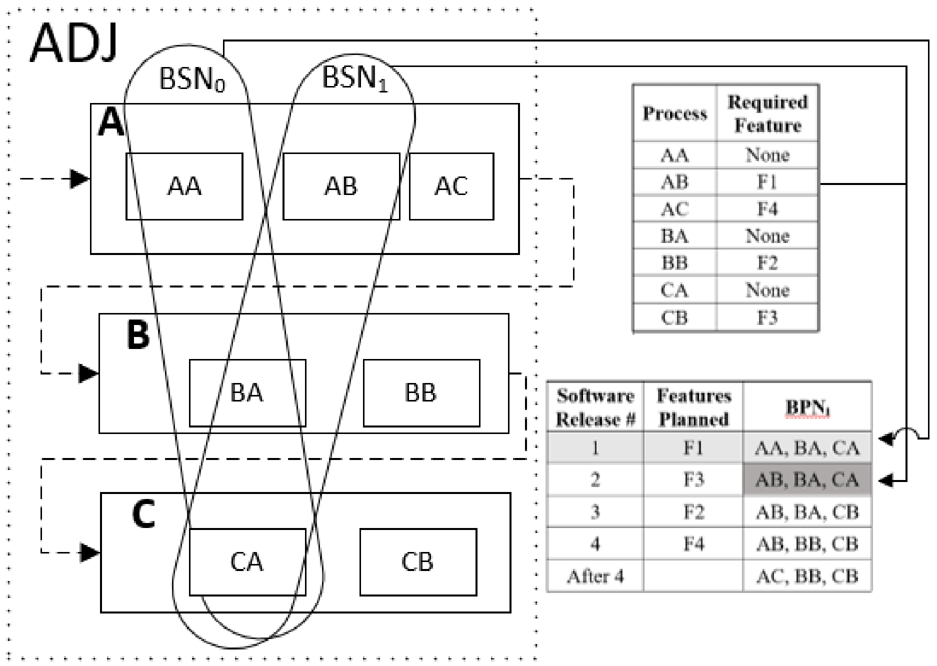

User Application
AC. Self-service Application Intake
Compliant Application
BB. Electronic Adjudication
Adjudicated Application
Non Compliance Notice
CB. Electronic Adjudication Review
Adjudicated Application Letter

Figure 3 - Final Application Adjudication BSN after 4 Releases

Figure 4 shows a diagram of the BSN transitions and the dependencies between services and features. The diagram demonstrates that BSN$_1$ is enabled by F1, which is developed in release 1 and consequently available during release 2.



Process	Required Feature
AA	None
AB	F1
AC	F4
BA	None
BB	F2
CA	None
CB	F3

Software Release #	Features Planned	BPN$_i$
1	F1	AA, BA, CA
2	F3	AB, BA, CA
3	F2	AB, BA, CB
4	F4	AB, BB, CB
After 4		AC, BB, CB

Figure 4 - BSN Transitions and Dependencies Between Services and Features

Note that there is no need to estimate the benefit, that is, the NPC of each feature; the benefit is determined at the BSN level. By not estimating the NPC at the feature level, the end result is more precise and it also eliminates a key assumption of previous approaches, which is that every dollar of benefit can be allocated to one and only one feature. In *Opti-Soft+* a BSN transition may require multiple features, which is a more realistic assumption. It is also simpler, because it eliminates the need to estimate cash flow at the feature level, which is really challenging and prone to subjectivity. In this section we use NPC as a proxy for business value, but in future sections we show that the metric actually used by *Opti-Soft+* is NPV. The NPC is the total cost of the BSN discounted by the hurdle rate while the NPV is the cash flow, that is, NPC with a negative sign.

3.2.2 Generalized BSN Model

We now show how to model a BSN. Continuing with the example in the previous section, we generalize the entire BSN by modeling it as a top-level service called ADJ (adjudication) whose subservices are A, B and C, shown in Figure 4. Service ADJ requires subservices A and B and C, consequently the relationship between ADJ and its subservices is of the type AND. Let's now model service A. Figure 4 shows that service A can have configurations AA or AB or AC, consequently the relationship between A and its subservices is of the type OR. Services B and C are also of the type OR. Service B is the parent of subservices BA and BB and service C is the parent of subservices CA and CB. Services ADJ, A, B and C are composite, that is, they are composed of subservices, while AA, AB, AC, BA, BB, CA, CB are atomic, that is, indivisible. Table 12 shows the generalized model for the Application Adjudication BSN, as well as the required feature for each service.

Table 12 - Generalized Service Model for the Application Adjudication BSN

Service	Type	Subservices	Required Feature
ADJ	AND	A, B, C	
A	OR	AA, AB, AC	
B	OR	BA, BB	
C	OR	CA, CB	
AA	Atomic		
AB	Atomic		F1
AC	Atomic		F4
BA	Atomic		
BB	Atomic		F2
CA	Atomic		
CB	Atomic		F3

Note that the subservices of a composite service can also be composite. This recursion makes the model very flexible because it supports as many hierarchy levels as needed in a consistent way.

For composite services, we model the generalization relationship as a binary function that is associated with each service. In an AND service, the value of the binary function is 1 for every subservice while in an OR service, the value is 1 for one and only one service and 0 for the others. For atomic services, we use the same binary function to model their participation in a particular BSN configuration.

For the Application Adjudication example, the instantiation of the binary function for the initial configuration AA, BA, CA is shown in Table 13. Note that A, B and C are instantiated as 1 because ADJ is a type AND service that requires all three to be present. Also note that the only atomic services instantiated are those that exist in the AA, BA, CA configuration, that is, *P(AA)=P(BA)=P(CA)=1* while all others are zero. The instantiation also satisfies the OR type for services A, B and C because for each one of them, there is only one activated atomic service.

Table 13 - Instantiation of the Binary Function for the As-Is BSN

Service ***s***	**Participation function** ***P(s)***
ADJ	1
A	1
B	1
C	1
AA	1

Service *s*	**Participation function** *P(s)*
AB	0
AC	0
BA	1
BB	0
CA	1
CB	0

Now that we covered how the BPN/BSN is modeled, let's discuss how the BSN cost and the software development cost are modeled.

3.3 BSN Cost/Benefit Model

The original framework, called *Opti-Soft+,* supported only labor costs because usually they are the largest cost driver, if not the only one. *Opti-Soft+* extends the original framework by supporting non-labor costs.

The types of business service costs supported by *Opti-Soft+* are: labor, non-labor fixed and non-labor variable. These costs are first computed at the atomic service level for each day of each release, and according to the formal analytical model described in Section 4, are captured in *CostPerDay(s,r),* where *s* is a service and *r* is the release. Note that each release corresponds to a particular configuration of the BSN and because the configuration does not change within a release, the cost for each day of a release is the same.

3.3.1 Labor Service Cost

As explained above, usually the largest, if not the only, cost component is labor. Each atomic service of the BSN is performed by workers with well-defined roles. Each role has a labor rate and each input processed or output produced by the role has a set duration. Given a service *s,* where a worker of labor role *l* has to process *n* inputs per day, and spends *t* hours for each input, the daily labor cost is:

$$LaborCostPerDay(s,r) = LaborRate(l) * t * n$$

LaborCostPerDay(s,r) is a component of *CostPerDay(s,r)*, which is the total daily cost of service *s* for release *r*.

3.3.2 Non-labor Fixed Service Cost

Fixed non-labor costs are not driven by inputs or outputs, instead they are driven by the services themselves. An example of a fixed cost associated with a particular service is rent. In the formal model, parameter *ServiceCostPerDay(s,r)* captures the daily cost for each atomic service *s*. It is a component of *CostPerDay(s,r)*.

3.3.3 Non-labor Variable Service Cost

Variable, non-labor costs are associated with the amount of work produced by an atomic service, that is, it is driven by the inputs or by outputs and are similar to the calculation of labor costs.

Parameters *CostPerInput(s,i)* and *CostPerOutput(s,o)* capture the non-labor variable costs for each input and output. These parameters are multiplied by the number of flows (inputs or outputs) per day to compute *FlowCostPerDay(s,r)*, which is a component of *CostPerDay(s,r)*.

3.3.4 Aggregated Cost of the BSN

The aggregated, total daily cost of each atomic service *s,* for release *r, CostPerDay(s,r)*, is:

$$LaborCostPerDay(s,r)+FlowCostPerDay(s,r)+ServiceCostPerDay(s,r)$$

The *CostPerDay(c,r)* for a composite service *c* is the sum of the costs for the subservices:

$$CostPerDay(c,r) = \sum_{s\in c} CostPerDay(s,r)$$

The highest level of the BSN hierarchy is the root service, so the aggregated total daily cost of the BSN for each release *r* is:

$$BSNCostPerD(r) = CostPerDay(root,r)$$

In order to calculate the NPV, we need to know the cash flow for each day *d* of the investment period. To do that, we transform the BSN cost per day for each release into a cash flow for each day, taking into consideration the start and end day of each release and a payment schedule. For an investment time horizon *th*, the BSN cash flow is captured in the metric below:

$$BSN.CashFlow(d) \quad \forall\, d \in [1..th]$$

3.4 <u>SDLC Cost/Benefit Model</u>

Opti-Soft+ assumes an Agile Software Development Life Cycle and Release Planning. Similarly to the BSN cost/benefit model, the largest component of the SDLC cost is labor. Another component is fixed cost such as office equipment. This section covers these two types of costs which are summed up to compute the SDLC cost per day, called *SWCostForDay*. This section starts by discussing release planning, which is a technique to organize development over time.

3.4.1 Release Planning

Opti-Soft+ requires a SDLC that incrementally delivers functionality in releases and where release planning is conducted at the feature level. Such approach is well documented in the Agile practice literature [13] and is called feature-driven. In a feature-driven approach, the product backlog contains two types of items at different levels of granularity: features and user stories. Initially features are identified, estimated, and added to the backlog; later on, they are broken down in user stories which are also identified, estimated and added to the backlog.

In a feature-driven approach, release planning is conducted at the feature level, that is, features are removed from the product backlog and assigned to releases. Usually, some kind of prioritization is utilized to drive release planning; in our case, we use the NPV of the benefit of the feature as the prioritization technique; the higher the NPV, the higher the priority.

Features, like user stories, are customarily estimated in points, which are based on the perceived effort to implement the features. There are many techniques to estimate user story/feature points, but the proposed approach does not require the adoption of any one in particular.

One important practice in Agile frameworks like Scrum, is the adoption of capacity as a means of determining how many points can fit in a particular timebox (Sprint or Release). Capacity is a function of the duration of the timebox, the size of the developer's team and the productivity of each developer.

Release Planning is the process of fitting features in a release in a way that it does not exceed capacity. The size of each feature is estimated in effort points consequently the total size of features assigned to a release, called the release size, cannot exceed the release capacity. Release planning is at the core of the Mixed-integer Linear Programming formulation because one of the results of solving the MILP problem is a release schedule.

3.4.2 SDLC Labor Cost

Once the release plan is known, the release size RS is calculated by adding the size of each feature in the release. Given the release duration RD, the developer cost per point DC, the development cost per day for each release r is:

$$devCostPerDay(r) = \frac{RS(r)}{RD(r)} \times DC$$

To make the framework more generic, the operations cost of the software, once the release is deployed, is also considered. We address the cost of operations by estimating the operations cost per point per day, multiplying it by the size of the operational software. This is captured in $opsCostPerDay(r)$. Next, we add the $devCostPerDay(r)$ to the $opsCostPerDay(r)$ to get the total labor cost for the release and then transform it into a cash flow for each day, taking into consideration the start and end day of each release and a payment schedule. For an investment time horizon th, the labor cash flow of software development is captured in the metric below:

$$LaborCashFlow(d)\ \forall\ d \in\ [1..th]$$

3.4.3 Non-labor Cost of the SDLC

Some non-labor costs can be experienced during software development. They are incurred by resources such as a hardware server, a software tool, etc…Every feature may require a set of resources. The full set of resources required by a feature f needs to be available prior to the start of the release that implements f. A resource might be paid in the release that implements f or in a prior release. We assume that resource costs are paid on the first day of each release, consequently on the first day of a release, all resources needed by all features in the release must be paid.

To be flexible, we allow multiple features to require the same resource, establishing dependencies among features. Resource dependencies are handled by a Dependency Graph. Assuming the following parameters:

- $\boldsymbol{ResSet}$ is the set of all non-labor resources
- $\boldsymbol{FeatureRes}$ maps features to resources
- $\boldsymbol{ResCost}$ maps a resource to its cost

The non-labor cost of software development resources is:

$$ResCashFlow(d)\quad \forall\ d \in\ [1..th]$$

3.4.4 SDLC Aggregated Cost

The aggregated cost of software development is:

$$SWD.CashFlow(d) = LaborCashFlow(d) + ResCashFlow(d) \ \forall \ d \in [1..TH]$$

3.5 Combining BSN and Software Cost

Section 3.3 computed the cost of the BSN while section 3.4 computed the cost of the software. Now we combine both into the metric

$$CombinedCashFlow(d) = BSN.Cashflow(d) + SWD.CashFlow(d)$$

The final metric, the $TimeWindowNPV(t)$, is the sum of the $CombinedCashFlow(t)$ for each day from 1 to *t*, divided by the discount rate.

3.6 Overview of the Mixed Integer Linear Program

The main goal of *Opti-Soft+* is to recommend an optimal release plan and corresponding BSN configuration, among many candidates. The recommendation is the result of solving a Mixed Integer Linear Program (MILP) that maximizes the business value of the investment, as measured by the *TimeWindowNPV* described above. Note that *TimeWindowNPV*, being a cash outflow, is a negative number and consequently maximizing the NPV is the same as minimizing the Net Present Cost. Section 4 describes the formal analytical model and MILP in detail but in general terms, the MILP is as follows:

Given a set of decision variables,

Given a set of parameters such as the services in the BSN, their relationships, the required BSN throughput, the features, number of releases, labor rates, etc.,

Maximize *TimeWindowNPV*

Subject to *Constraints*

Solving the MILP produces a recommendation by instantiating the decision variables. The recommendation has the following parts:

1. A release plan where each feature is assigned to a particular release in an optimal sequence.
2. An optimal BSN configuration to be adopted after each release
3. The required throughput of each service in the optimal BSN configurations

The decision variables that make up the recommendation are:

1. *On(s,r)* – a binary indicating whether service *s* is activated in the BSN configuration during the development of release *r*.
2. *IBF(r,f)* - a binary indicating whether business feature *f* is implemented in release *r*.
3. *ITF(r,f)* - a binary indicating whether technical feature *f* is implemented in release *r*.
4. *InputThru(s,i,r)* – a real, indicating the number of inputs of type *i* that must go through atomic service *s* during the development of release *r,* in order to satisfy the required BSN throughput, which is a parameter called *Demand.*

The MILP constraints are:

- *FeatureSetsForReleasesArePairwiseDisjoint*
    - A feature cannot be implemented in more than one release

- *DependencyGraphIsSatisfied*
    - If feature *F2* depends on *F1,* then *F1* has to be developed in the same release as *F2* or in a prior release
- *BSNDemandIsSatisfied*
    - The number of flows of input *i* through the root service is greater or equal *Demand(i),* which is a given parameter
- *BSNSupplyIsSatisfied*
    - The number of flows of output *o* through the root service is greater or equal *Demand(o),* which is a given parameter
- *AllSubServicesAreActivated*
    - All subservices of a composite service *c* of type AND have to be activated, that is, *On(s,r)=1* for every subservice *s* of *c* and for every release *r*
- *OnlyOneServiceIsActivated*
    - Only one subservice of a composite service *c* of type OR can be activated, that is, if *On(a,r)=1* then *On(s,r)=0* for every subservice *s* of *c, s≠a,* and for every release *r*
- *RootMustBeActivated*
    - The root service must activated, that is, *On(BSN.rootID,r)=1* for every release *r*
- *TotalSupplyMatchesTotalDemand*
    - For composite services, the total flows through the BSN have to balance, that is, for each flow *f,* the total supply of *f* is equal to the total demand of *f*

- *FeatureDependencyIsSatisfied*
    - If an atomic service *a* is activated in release *r*, that is, *On(a,r)=1,* then every feature *f* that is required by *a* has to be implemented in a release *n*, $n \leq r$
- *DeactivatedServicesIsSatisfied*
    - If an atomic service *a* is deactivated in release *r*, that is, *On(a,r)=0,* then nothing can flow through it, that is, the number of flows *f* through *a* in *r* has to be zero, for every *f*
- *ReleaseSizeCannotExceedCapacity*
    - The sum of the sizes of all features in a release *r* cannot exceed the capacity of the team on release *r*

Now that we intuitively explained the modeling of the BSN, the cost/benefit model, the decision variables of the MILP and the constraints, we are ready to cover the formal analytical model.

4. *OPTI-SOFT+* FORMAL ANALYTICAL MODEL

4.1 Model Introduction

In this section, we formally model the release scheduling problem as a set of tuples $t\langle Parameters,\ DecisionVariables,\ Computation,\ Constraints,\ InterfaceMetrics\rangle$ where *t* is the tuple name.

- *Parameters* are given
- *DecisionVariables* are the variables that control the performance of the model.
- *Computation* defines the equations to calculate the *DecisionVariables, Constraints and InterfaceMetrics*.
- *Constraints* is the set $\{C_1(P, DV),\ \dots C_n(P, DV)\}$, where $C_1(P, DV),\ \dots C_n(P, DV)$ are predicates, expressed as a function of parameters P and decision variables DV, that need to be satisfied.
- *InterfaceMetrics* is the set $\{M_1(P, DV),\ \dots,\ M_k(P, DV)\}$, where $M_1(P, DV)$, $\dots,\ M_k(P, DV)$ represent metrics that can be interfaced among tuples. Interface metrics are calculated in the *Computation* element.

Due to the complexity of the Release Scheduling problem, we model it as a hierarchy of component tuples in a top-down fashion as shown in Figure 5. The hierarchy establishes an inheritance relationship between the formalizations. A formalization *A* at level *n* establishes a parent-child relationship to formalization *B* at level *n+1* if *A* uses some result of *B* like an interface metric. For example, the *ReleaseScheduling* formalization uses

BSN's CashFlow interface metric, consequently all *BSN* tuples are available in *ReleaseScheduling*. In addition, because *BSN* is a child of *ReleaseScheduling*, *BSN* inherits its *Parameters*, *DecisionVariables* and *InterfaceMetrics* and consequently can refer to them.

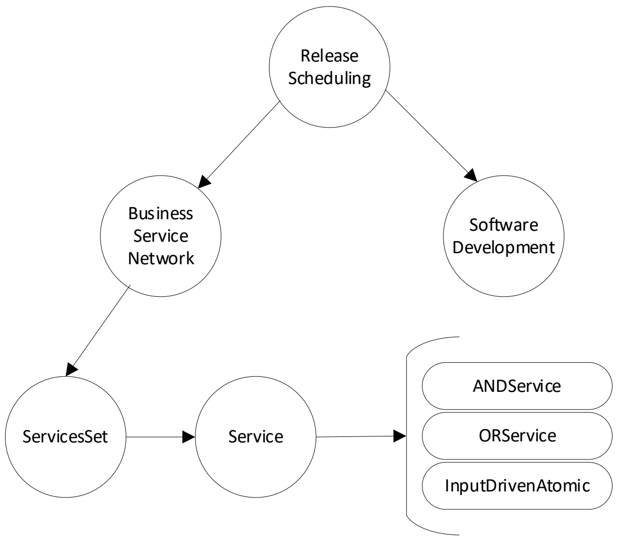


Figure 5 - Hierarchy of the Components of the Formal Model

The components of the formal model are:

1. *ReleaseScheduling* is at the top of the hierarchy. It aggregates results from the other components in order to compute the *TimeWindowNPV(d)* for each day *d* in the time horizon. Its main parameters describe the features, their dependencies, sizes, time horizon, number of releases and release duration. The decision variables are *IBF(r,f)* and *ITF(r,f)*.

2. *BusinessServiceNetwork* computes the BSN's *CashFlow(d)*, which is a term in the computation of *TimeWindowNPV(d)*. Its main parameters are the labor rates, payment schedule and the set of services.
3. *Service* is a container for the various types of services. Every service is identified by an id and has a type.
4. *ANDservice* describes a composite services of type AND. It aggregates the *CostPerDay(id,r)* of all subservices, which is then used as a term in the computation of *BSN.CashFlow(d)*. The parameters are the id, type, set of inputs, set of outputs and the set of subservices. The decision variable is *On(s,r),* a binary indicating whether the service is activated or not.
5. *ORservice* is similar to *ANDservice*; the only difference is that the relationship with its subservices is of type OR.
6. *InputDrivenAtomicService* models an atomic, (indivisible) service which's throughput is driven by the number of inputs that it needs to consume. Its main parameters are the ratio of inputs to outputs, the time spent to produce one output and the set of features required for the service to be activated. The decision variables are *On(s,r),* a binary indicating whether the service is activated or not, and *InputThru(s,i,r),* the required throughput. It computes the *CostPerDay(id,r),* which is aggregated in the composite services.
7. *OutputDrivenAtomicService* models an indivisible service which's throughput is driven by the number of outputs that it needs to consume. Its main parameters are the ratio of outputs to inputs, the time spent to produce one output and the set of

features required for the service to be activated. The decision variables are *On(s,r),* a binary indicating whether the service is activated or not, and *OutputThru(s,i,r),* the required throughput. It computes the *CostPerDay(id,r),* which is aggregated in the composite services.

8. *SoftwareDevelopment* computes *SWD.CashFlow(d),* which is a term in the calculation of *TimeWindowNPV(d)*. Its main parameters are the size of the team, productivity, and the cost per unit. It uses feature point as a unit of cost.

Figure 6 shows the roll up of the cost calculations from the bottom of the hierarchy to the top.

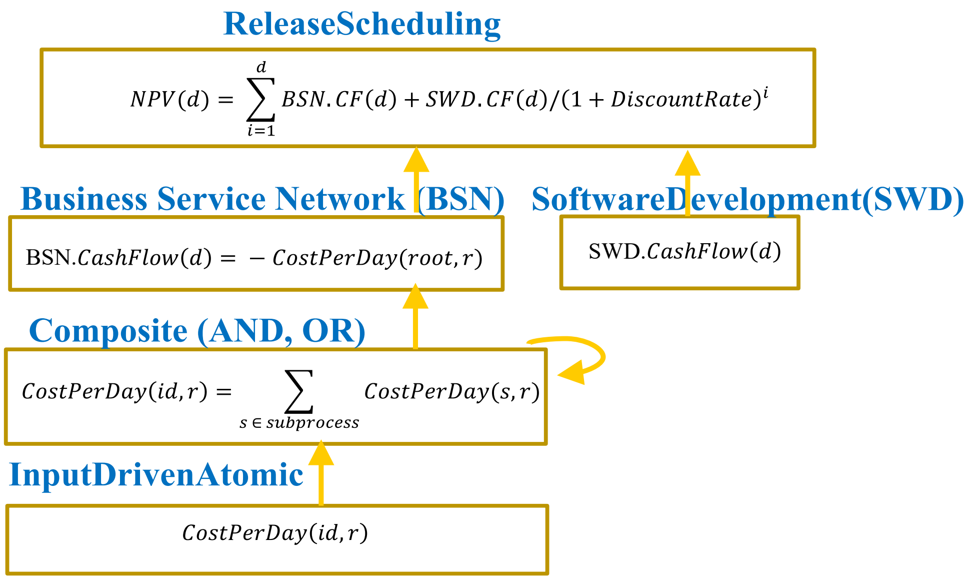

ReleaseScheduling
$NPV(d) = \sum_{i=1}^{d} BSN.CF(d) + SWD.CF(d)/(1 + DiscountRate)^i$
Business Service Network (BSN)
SoftwareDevelopment(SWD)
$\mathrm{BSN}.CashFlow(d) = -CostPerDay(root, r)$
$\mathrm{SWD}.CashFlow(d)$
Composite (AND, OR)
$CostPerDay(id,r) = \sum_{s \in subprocess} CostPerDay(s,r)$
InputDrivenAtomic
$CostPerDay(id,r)$

Figure 6 - Roll up of cost calculations from the bottom to the top

In the next sections we describe each formalization component in detail.

4.2 Release Scheduling Formalization

ReleaseScheduling (RSch) formalization is a tuple ⟨*Parameters, DecisionVariables, Computation, Constraints, InterfaceMetrics*⟩

where:

Parameters, also denoted **Parm,** is a tuple ⟨*Features, TH, DiscountRate, ReleaseInfo, RestSet, ResCost, FeatureRes, BSN.Parameters, SWD.Parameters*⟩

Where *Features* is a tuple ⟨*BF, TF, DG, FS* ⟩ where:

- ***BF*** is a set of business features
- ***TF*** is a set of technical features, such that

  $BF \cap TF = \emptyset$
- ***DG***, (Dependency Graph), is a partial order over $F = BF \cup TF$, $(f_1, f_2) \in DG$ also denoted $f_1 \prec f_2$, means that f_2 is dependent on f_1, that is, feature f_1 is a pre-requisite for feature f_2.
- $\boldsymbol{FS}: F \rightarrow \mathbb{R}^+$ is a function described as follows: $(\forall f \in F), FS(f)$ gives the size, in effort point, of each feature f.
- ***TH*** is the time horizon for analysis in days
- ***DiscountRate*** is the daily rate to discount cash flows.
- ***ReleaseInfo*** is a tuple ⟨*NR, RD* ⟩, where:
  - ***NR*** is the number or releases

- $\boldsymbol{RD} : [1..NR] \rightarrow \mathbb{R}^+$ is a function described as follows: $(\forall\, r \in [1..NR]), RD(r)$ gives the maximum duration in days for release r.
- $\boldsymbol{ResSet}$ is a set of non-labor resources that have a fixed cost
- $\boldsymbol{ResCost}$: $ResSet \rightarrow \mathbb{R}^+$ is a function described as follows: $(\forall\, e \in ResSet)$, $ResCost(e)$ gives the non-labor fixed cost for resource e.
- $\boldsymbol{FeatureRes}$: $F \rightarrow 2^{ResSet}$is a function described as follows: $(\forall f \in BF \cup TF, \forall e \in ResSet)$, $FeatureRes(f)$ gives a set of resources e required by feature f.
- ***BSN.Parameters*** is defined in section 4.3
- ***SWD.Parameters*** is defined in section 4.9

DecisionVariables, also denoted **DV,** is a tuple $\langle IBF, ITF, BSN.DecisionVariables, SWD.DecisionVariables\rangle$

where:

- $\boldsymbol{IBF} : [1..NR] \rightarrow 2^{BF}$ is a function described as follows: $(\forall\, r \in [1..NR]), IBF(r)$ gives a set of business features planned to be implemented in release r.
- $\boldsymbol{ITF} : [1..NR] \rightarrow 2^{BF}$ is a function described as follows: $(\forall\, r \in [1..NR]), ITF(r)$ gives a set of technical features planned to be implemented in release r.
- $\boldsymbol{BSN.DecisionVariables}$ is defined in section 4.3.
- $\boldsymbol{SWD.DecisionVariables}$ is defined in section 4.9.

Computation

1.Let $SoFarIBF: [1..NR + 1] \rightarrow 2^{BF}$ be a function described as follows: $(\forall\ r \in [1..NR + 1])$, $SoFarIBF(r)$ gives the set of all business features implemented up to release r or the period after the last release, computed as follows:

$$SoFarIBF(r) = \bigcup_{i=1}^{r-1} IBF(i)$$

2.Let $CombinedCashFlow: [1..TH] \rightarrow \mathbb{R}$ be a function described as follows: $(\forall\ d \in [1..TH])$, $CombinedCashFlow(d)$ gives the combined income/expenditure of both the Business Service Network and the Software Development, $(\forall\ d \in [1..TH])$, computed as follows:

$$CombinedCashFlow(d) = BSN.IM.Cashflow(d) + SWD.IM.CashFlow(d)$$

where:

• ***BSN.IM.CashFlow*** is defined in section *BSN.InterfaceMetrics* of section 4.3

• ***SWD.IM.CashFlow*** is defined in section *Software.InterfaceMetrics* of section 4.9.

Note that a negative cash flow means that it is a cash outflow.

3.Let $TimeWindowNPV: [1..TH] \rightarrow \mathbb{R}$ be a function described as follows: $(\forall\ d \in [1..TH])$, $TimeWindowNPV(d)$ gives the Net Present Value (NPV) of the *CombinedCashFlow* for the time investment window $[1..d]$, computed as follows:

$$TimeWindowNPV(d) = \sum_{i=1}^{d} \frac{CombinedCashFlow(i)}{(1 + DiscountRate)^i}$$

4.Let $F = BF \cup TF$

5.Let $IF(r) = IBF(r) \cup ITF(r), (\forall r \in [1..NR])$

6.*FeatureSetsForReleasesArePairwiseDisjoint* constraint is:

$$IF(i) \cap IF(j) = \emptyset \ \ (\forall\, i, j, \in [1..NR], i \neq j)$$

7.*DependencyGraphIsSatisfied* constraint is:

$$(\forall r \in [1..NR])(\forall\, f_1, f_2 \ \in \ F),$$

$$(f_1 \prec f_2 \wedge f_2 \in IF(r)) \rightarrow (f_1 \in \bigcup_{i=1}^{r} IF(i))$$

Constraints

1. ***FeatureSetsForReleasesArePairwiseDisjoint*** is defined in computation #6 above.

2. ***DependencyGraphIsSatisfied*** *is defined in computation #7 above.*

3. ***BSN.Constraints*** is defined in section 4.3.

4. ***SWD.Constraints*** is defined in section 4.9.

InterfaceMetrics, also denoted **IM,** is a tuple

$\langle SoFarIBF, CombinedCashFlow,\ TimeWindowNPV,\ BSN.InterfaceMetrics,$

$SWD.InterfaceMetrics\ \rangle$,

where:

- $\boldsymbol{CombinedCashFlow}$ is defined in computation #2 above.
- $\boldsymbol{TimeWindowNPV}$ is defined in computation #3 above.
- ***BSN.InterfaceMetrics*** is defined in section 4.3
- ***SWD.InterfaceMetrics*** is defined in section 4.9

4.3 Business Service Network Formalization

BusinessServiceNetwork formalization, also denoted **BSN,** is a tuple ⟨*Parameters, DecisionVariables, Computation, Constraints, InterfaceMetrics*⟩, where:

Parameters, also denoted **Parm,** is a tuple ⟨*LaborRates, LaborPaySched, BSNDemand, ServicesSet, rootID*⟩,

where:

- ***LaborRates*** is a tuple ⟨*LR, Rate*⟩ where*:*
- ***LR*** is a set of labor roles
- $\boldsymbol{Rate}: LR \to \mathbb{R}^+$ is a function described as follows: $(\forall\, l\ \in\ LR), Rate(l)$ gives the daily rate for labor role l.
- ***LaborPaySched,*** the labor cost payment schedule, is a tuple $\langle NLP, LaborPayDays\rangle$, where:
  - $\boldsymbol{NLP} \in \mathbb{R}^+$is the number of labor payments over the entire time horizon
  - $\boldsymbol{LaborPayDay}: [1..NLP] \to\ [1..TH]$ is a function described as follows: $(\forall\, p \in [1..NLP),\ LaborPayDay(p)$ gives the day, relative to the first day of the time horizon, on which a payment p is made.
- ***BSNDemand,*** is a tuple $\langle BSNI, BSNO, Demand\rangle$,

  where:

  - ***BSNI*** is a set of input flow ids that have to be processed by the Service Network.
  - ***BSNO*** is a set of output flow ids that have to be produced by the Service Network.

- $\boldsymbol{Demand}: BSNI \cup BSNO \rightarrow \mathbb{R}^{+}$ is a function described as follows: $(\forall\, j \in BSNI \cup BSNO)$, $Demand(j)$ gives for every flow j, the required processing throughput per day.
- ***ServicesSet*** is the set of all services in the Service Network, defined in section 4.4.
- $\boldsymbol{rootID}$ is the *id* of the Service, in the *ServicesSet,* which is designated to be the "root". The definition of a Service is given in section 4.4.

DecisionVariables is the set $\{s.DecisionVariables \mid s \in ServicesSet\}$. See section 4.4.

Computation

1. Let *root* be a Service in *ServicesSet* with *id=rootID*
2. $BSNDemandIsSatisfied$ constraint:

   $(\forall\, i \in BSNI)\ (\forall\, r \in [1..NR+1])$,

   $Service.IM.InputThru(rootID, i, r) \geq Demand\ (i)$

   - $Service.IM.InputThru(rootID, r)$ is defined in section 4.4.
3. $BSNSupplyIsSatisfied$ *constraint:*

   $(\forall\, o \in BSNO)\ (\forall\, r \in [1..NR+1])$,

   $Service.IM.OutputThru(rootID, o, r) \geq Demand\ (o)$

   - $Service.IM.OutputThru(rootID, o, r)$ is defined in section 4.4.
4. Let $BSNCostForDay : [1..TH] \rightarrow \mathbb{R}^{+}$ be a function described as follows: $(\forall\, d \in [1..TH])$, $BSNCostForDay(d)$ gives the service network labor cost accrued for day d computed as:

   $BSNCostForDay(d) = Service.IM.CostPerDay(rootID, r)$

Where:

- r is the release period (or period after the last release) where day d appears, i.e.,

$$SWD.IM.firstDay(r) \leq d \leq SWD.IM.lastDay(r)$$

- $Service.IM.CostPerDay(rootID, r)$ is defined in section 4.4.
- $SWD.IM.firstDay$ and $SWD.IM.lastDay$ are defined in section 4.9.

5. Let $BSNPayment: [1..NLP] \rightarrow \mathbb{R}$ be a function described as follows: $(\forall\, p \in [1..NLP])$, $BSNPayment(p)$ gives the service network labor payment in dollars, for each scheduled payment p, computed as:

$$BSNPayment(p) = \begin{cases} \displaystyle\sum_{d=1}^{LaborPayDay(p)} BSNCostForDay(d) & \forall p = 1 \\ \displaystyle\sum_{d=LaborPayDay(p-1)+1}^{LaborPayDay(p)} BSNCostForDay(d) & \forall p = [2..NLP] \end{cases}$$

6. Let $CashFlow : [1..TH] \rightarrow \mathbb{R}^{+}$ be a function described as follows: $(\forall\, d \in [1..TH])$, $CashFlow(d)$ gives the cash flow for the entire Business Service Network for day d, computed as follows:

$$CashFlow(d) = \begin{cases} -BSNPayment(p)\ if\ \exists p | d = LaborPayDay(p) \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad Otherwise \end{cases}$$

Constraints

1. $\boldsymbol{BSNDemandIsSatisfied}$(see Computation #2)
2. $\boldsymbol{BSNSupplyIsSatisfied}$ (see Computation #3)
3. $\boldsymbol{Service.IM(rootID,r).Constraints}$ (See section 4.4)

InterfaceMetrics, also denoted **IM,** is a tuple $\langle CashFlow \rangle$, where:

• $\boldsymbol{CashFlow}$ is defined in computation #6 above.

4.4 <u>Service Formalization</u>

ServicesSet formalization is a set of **Service**, where:

Service is a tuple ⟨*Parameters, DecisionVariables, Computation, Constraints, InterfaceMetrics*⟩, defined separately for each $ServiceType \in \{ANDservice, ORservice, InputDrivenAtomicService, OutDrivenAtomicService\}$

Every service has an ***id*** and a ***ServiceType***. We denote by $service(id)$ the service with identifier id. **Service** is a container for the service types below and inherent their tuples.

• ***ANDservice*** type is defined in section 4.5.

• ***ORservice*** type is defined in section 4.6.

• ***InputDrivenAtomicService*** type is defined in section 4.7.

• ***OutputDrivenAtomicService*** type is defined in section 4.8.

4.5 <u>ANDservice Formalization</u>

Intuitively, an ANDservice is a composite service, that is, an aggregation of sub-services such that all sub-services are activated.

ANDservice formalization is a tuple ⟨*Parameters, DecisionVariables, Computation, Constraints, InterfaceMetrics*⟩

where:

Parameters, also denoted **Parm,** is a tuple ⟨*id, ServiceType(id), I(id), O(id), Subservices(id)*⟩

where:

- ***id*** is the Service id, which must be unique across all services in the *ServicesSet.*
- ***I(id)*** is a set of inputs
- ***O(id)*** is a set of outputs
- ***Subservices(id)*** is a set of the ids of the sub-services.
- ***ServiceType(id)*** is *ANDservice.*

DecisionVariables, also denoted **DV**, is a tuple $\langle On(id) \rangle$

where:

- $\boldsymbol{On(id)}: [1..NR+1] \rightarrow \{0,1\}$ is a function that determines whether the Service *id* is activated or not, for a particular release, i.e., $(\forall\, r \in [1..NR+1]), On(id)(r)$, also denoted by *On(id,r)* is as follows:

$$On(id,r) = \begin{cases} 1\ if\ service\ id\ is\ activated\ in\ release\ r \\ 0\ otherwise \end{cases}$$

Computation

1.*AllSubservicesAreActivated* constraint:

Let *n* be the cardinality of *Subservices(id)*. Then the constraint is:

$$\sum_{i\,\in\,Subservices(id)} On(i,r)\ = n * On(id,r), \qquad \forall\, r \in [1..NR+1]$$

2.Let $SetAllIO(id)$ be a set of inputs and outputs, computed as follows:

$$SetAllIO(id) = I(id) \bigcup O(id) \cup \left(\bigcup_{i \in Subservices(id)} I(i) \right)$$

$$\cup \left(\bigcup_{i \in Subservices(id)} O(i) \right)$$

3.Let $InternalSupply(id): SetAllIO \times [1..NR+1] \rightarrow \mathbb{R}$ be a function described as follows:($\forall j \in SetAllIO(id), \forall r \in [1..NR+1]$) $InternalSupply(id)(j,r)$, also denoted $InternalSupply(id,j,r)$, gives the internal supply of flow j during release r (and the period after the last release), computed as follows:

$$InternalSupply(id,j,r) = \begin{cases} \sum_{s \in Subservices(id)} OutputThru(s,j,r) & if\ j \in O(s) \\ 0 & otherwise \end{cases}$$

4.Let $InternalDemand(id): SetAllIO \times [1..NR+1] \rightarrow \mathbb{R}$ be a function described as follows:($\forall j \in SetAllIO(id), \forall r \in [1..NR+1]$)$InternalDemand(id)(j,r)$, also denoted $InternalDemand(id,j,r)$, gives the internal demand of flow j during release r (and the period after the last release), computed as follows:

$$InternalDemand(id,j,r) = \begin{cases} \sum_{s \in Subservices(id)} InputThru(s,j,r) & if\ j \in I(s) \\ 0 & otherwise \end{cases}$$

5.Let $InputThru(id): I(id) \times [1..NR+1] \rightarrow \mathbb{R}^{+}$ be a function described as follows: ($\forall\ i \in I\ (id), \forall\ r \in [1..NR+1]$), $InputThru(id)(i,r)$, also denoted

$InputThru(id, i, r)$, gives the throughput of i (or quantity per day) during release r or the period after the last release, computed as

$\forall i \in I(id), \forall\, r \in [1..NR+1]$,

$$InputThru(id, i, r) = InternalDemand(id, i, r) - InternalSupply(id, i, r)$$

6.Let $OuputThru(id): O(id) \times [1..NR+1] \rightarrow \mathbb{R}^{+}$ be a function described as follows: $(\forall\, o \in O\ (id), \forall\, r \in [1..NR+1])$, $OutputThru(id)(o, r)$, also denoted $OutputThru(id, o, r)$, gives the throughput of o (or quantity per day) during release r or the period after the last release, computed as

$\forall\, o \in O(id), \forall\, r \in [1..NR+1]$,

$$OutputThru(id, o, r) = InternalSupply(id, o, r) - InternalDemand(id, o, r)$$

7.Let $TotalDemand(id): SetAllIO \times [1..NR+1] \rightarrow \mathbb{R}$ be a function described as follows: $(\forall j \in SetAllIO(id), \forall r \in [1..NR+1])\ TotalDemand(id)(j, r)$, also denoted $TotalDemand(id, j, r)$, gives the total demand of flow j during release r (and the period after the last release), computed as follows:

$$TotalDemand(id, j, r) = \begin{cases} OutputThru(id, j, r) + InternalDemand(id, j, r) & if\ j \in O(id) \\ InternalDemand(id, j, r) & otherwise \end{cases}$$

8.Let $TotalSupply(id): SetAllIO \times [1..NR+1] \rightarrow \mathbb{R}$ be a function described as follows: $(\forall j \in SetAllIO(id), \forall r \in [1..NR+1])\ TotalSupply(id)(j, r)$, also denoted $TotalSupply(id, j, r)$, gives the total supply of flow j during release r (and the period after the last release), computed as follows:

$$TotalSupply(id, j, r) = \begin{cases} InputThru(id, j, r) + InternalSupply(id, j, r) & if\ j \in I(id) \\ InternalSupply(id, j, r) & otherwise \end{cases}$$

9. *TotalSupplyMatchesTotalDemand* constraint is:

$\forall\, j \in SetAllIO(id), \forall\, r \in [1..NR+1],$

$$TotalSupply(id,j,r) = TotalDemand(id,j,r)$$

10. Let $CostPerDay(id)$: $[1..NR+1] \rightarrow \mathbb{R}$ be a function described as follows: ($\forall\, r \in [1..NR+1]$), $CostPerDay(id)(r)$, also denoted $CostperDay(id,r)$, gives the total dollar cost per day during period *r* and the period after the last period, computed as:

$$CostPerDay(id,r) = \sum_{i \in Subservices(id)} Service.IM.CostPerDay(i,r)$$

11. *RootMustBeActivated* constraint is:

$$On(BSN.rootID,r) = 1 \qquad \forall\, r \in [1..NR+1]$$

Constraints are as follows:

1.***AllSubservicesAreActivated*** (see computation #1)

2.***TotalSupplyMatchesTotalDemand*** **(**see computation # 9)

3.***RootMustBeActivated*** **(**see computation # 11)

InterfaceMetrics, also denoted **IM,** is a tuple ⟨*CostPerDay(id), InputThru(id), OutputThru(id)*⟩

where:

- $\boldsymbol{CostPerDay}(id)$ is defined in computation #10 above.
- $\boldsymbol{InputThru}(id)$ is defined in computation #5 above.
- $\boldsymbol{OutputThru}(id)$ is defined in computation #6 above.

4.6 ORservice Formalization

Intuitively, an ORservice is a composite service, that is, an aggregation of sub-services such that only one sub-services is activated.

ORservice formalization is a tuple ⟨*Parameters, DecisionVariables, Computation, Constraints, InterfaceMetrics*⟩

where:

Parameters, also denoted **Parm,** is a tuple ⟨*id, ServiceType(id), I(id), O(id), Subservices(id)*⟩

where:

- ***id*** is the Service id, which must be unique across all services in the *ServicesSet.*
- ***I(id)*** is a set of inputs
- ***O(id)*** is a set of outputs
- ***Subservices****(id)* is a set of the ids of the sub-services.
- ***ServiceType****(id)* is *ORservice.*

DecisionVariables, also denoted **DV,** is a tuple $\langle On(id)\rangle$

where:

- $\boldsymbol{On}(id)\colon [1..NR+1] \to \{0,1\}$ is a function that determines whether the Service *id* is activated or not, for a particular release, i.e., $(\forall\, r \in [1..NR+1]), On(id)(r)$, also denoted by *On(id,r)* is as follows:

$$On(id,r) = \begin{cases} 1 \ if\ service\ id\ is\ activated\ in\ release\ r \\ 0\ otherwise \end{cases}$$

Computation

1.*OnlyOneServiceIsActivated* constraint:

$$\sum_{i \in Subservices(id)} On(i,r) = On(id,r), \qquad \forall\, r \in [1..NR+1]$$

2. Same as ANDservice computation #2
3. Same as ANDservice computation #3
4. Same as ANDservice computation #4
5. Same as ANDservice computation #5
6. Same as ANDservice computation #6
7. Same as ANDservice computation #7
8. Same as ANDservice computation #8
9. Same as ANDservice computation #9
10. Same as ANDservice computation #10
11. Same as ANDservice computation #11

Constraints are as follows:

1. ***OnlyOneServiceIsActivated*** (see computation #1)
2. ***TotalSupplyMatchesTotalDemand*** **(**see computation # 9)
3. ***RootMustBeActivated*** **(**see computation # 11)

InterfaceMetrics, also denoted **IM,** is a tuple ⟨*CostPerDay(id), InputThru(id), OutputThru(id)*⟩

where:

- $\boldsymbol{CostPerDay}(id)$ is defined in computation #10 above.
- $\boldsymbol{InputThru}(id)$ is defined in computation #5 above.
- $\boldsymbol{OutputThru}(id)$ is defined in computation #6 above.

4.7 <u>InputDrivenAtomicService Formalization</u>

Intuitively, an InputDrivenAtomicService is an indivisible, atomic service which's throughput is driven by the number of inputs that it needs to consume, for example, a process that receives applications and adjudicates them.

InputDrivenAtomicService formalization is a tuple ⟨*Parameters, DecisionVariables, Computation, Constraints, InterfaceMetrics*⟩

Parameters, also denoted **Parm,** is a tuple $\langle id, ServiceType(id),\ I(id),\ O(id),\ RBF(id),\ ServiceRoles(id),\ IOthruRatio(id),\ RoleTimePerIO(id),\ ServiceCostPerDay,\ CostPerInput, CostPerOutput\rangle$

where:

- ***id*** is the Service id.
- ***I(id)*** *is a set of inputs*
- ***O(id)*** *is a set of outputs*
- $\boldsymbol{RBF}(id) \subseteq ReleaseScheduling.Parm.BF$ is a set of business features required by Service *id*
- $\boldsymbol{ServiceRoles}(id) \subseteq BSN.Parm.LR$ is a set of roles involved in the business service.

- $\boldsymbol{IOthruRatio}(id){:}\, I(id) \times O(id) \to \mathbb{R}^+$ is a function described as follows: $\left(\forall\, i \in I(id)\right)$, $\left(\forall\, o \in O(id)\right)$, $IOthruRatio(id)(i,o)$ also denoted as $IOthruRatio(id,i,o)$, gives for input i and output o, the ratio of output throughput based on the input throughput.
- $\boldsymbol{RoleTimePerIO}(id){:}\, ServiceRoles(id) \times (I(id) \cup O(id)) \to \mathbb{R}^+$ is a function described as follows: $(\forall\, l \in ServiceRoles(id),\ \forall\, j \in I(id) \cup O(id))$, $RoleTimePerIO(id)(l,j)$, also denoted as $RoleTimePerIO(id,l,j)$, gives the amount of time, in hours, that role l spends per flow j.
- $\boldsymbol{ServiceCostPerDay}{:}\, ServicesSet \to \mathbb{R}^+$ is a function described as follows: $(\forall\, s \in ServicesSet)$, $ServiceCostPerDay(s)$ gives the non-labor fixed cost of service *s* for each day.
- $\boldsymbol{CostPerInput}(id){:}\, I(id) \to \mathbb{R}^+$ is a function described as follows: $(\forall\, i \in I(id))$, $CostPerInput(id)(i)$, also denoted as $CostPerInput(id,i)$, gives the non-labor fixed cost for each input i processed by the service *id.*
- $\boldsymbol{CostPerOutput}(id){:}\, O(id) \to \mathbb{R}^+$ is a function described as follows: $(\forall\, o \in O(id))$, $CostPerOutput(id)(o)$, also denoted as $CostPerInput(id,o)$, gives the non-labor fixed cost for each output *o* processed by the service *id.*
- ***ServiceType****(id)* is *InputDrivenAtomicService*

DecisionVariables, also denoted **DV,** is a tuple $\langle On(id), InputThru(id)\rangle$

where:

- $\boldsymbol{On}(id)$: $[1..NR+1] \rightarrow \{0,1\}$ is a function that determines whether the Service id is activated or not, for a particular release, i.e., $(\forall\, r \in [1..NR+1]), On(id)(r)$, also denoted by *On(id,r)* is as follows:

$$On(id,r) = \begin{cases} 1\ if\ service\ id\ is\ activated\ in\ release\ r \\ 0\ otherwise \end{cases}$$

- $\boldsymbol{InputThru}(id)$: $I(id) \times [1..NR+1] \rightarrow \mathbb{R}^{+}$ is a function described as follows: $(\forall\, i \in I\ (id), \forall\, r \in [1..NR+1])$, $InputThru(id)(i,r)$, also denoted $InputThru(id,i,r)$, gives the throughput of i (or quantity per day) during release r or the period after the last release.

Computation

1. *FeatureDependencyIsSatisfied* constraint:

$$On(id, r) = 1 \rightarrow RBF(id) \subseteq RSch.IM.SoFarIBF(r) \quad \forall\, r \in [1..NR + 1]$$

2. *DeactivatedServicesIsSatisfied* constraint:

$\forall\, i \in I(id), \forall r \in [1..NR + 1]$,

$On(id,r) = 0 \rightarrow InputThru(id, i, r) = 0$

3. Let $OuputThru(id): O(id) \times [1..NR + 1] \rightarrow \mathbb{R}^+$ be a function described as follows: $(\forall\, o \in O\,(id), \forall\, r \in [1..NR + 1])$, $OutputThru(id)(o, r)$, also denoted $OutputThru(id, o, r)$, gives the throughput of o (or quantity per day) during release r or the period after the last release, computed as

$\forall\, o \in O(id), \forall\, r \in [1..NR + 1]$,

$$OutputThru(id, o, r) = \sum_{i \in I(id)} (IOthruRatio(id, i, o) \times InputThru(id, i, r))$$

4. Let $TimePerDay(id): [1..NR + 1] \times ServiceRoles(id) \rightarrow \mathbb{R}^+$ be a function described as follows: $(\forall\ l\ \in ServiceRoles(id), r \in [1..NR + 1])$, $TimePerDay(id)(l, r)$, also denoted $TimePerDay(id, l, r)$, gives the total duration per day for role l and release r (and the period after the last release), computed as:

$$\begin{aligned} TimePerDay(id, l, r) \\ &= \sum_{j \in I(id)} (RoleTimePerIO(id, l, j) \times InputThru(id, j, r)) \\ &+ \sum_{j \in O(id)} (RoleTimePerIO(id, l, j) \times OutputThru(id, j, r)) \end{aligned}$$

5. Let $LaborCostPerDay(id)\colon [1..NR+1] \rightarrow \mathbb{R}$ be a function described as follows:($\forall\, r \in [1..NR+1]$), $LaborCostPerDay(id)(r)$, also denoted $LaborCostperDay(id,r)$, gives the total labor cost per day during release *r*, computed as follows:

$$LaborCostPerDay(id,r) = \sum_{l \in ServiceRoles} (BSN.Parm.Rate(l) \times TimePerDay(id,l,r))$$

6. Let $FlowCostPerDay(id)\colon [1..NR+1] \rightarrow \mathbb{R}$ be a function described as follows:($\forall\, r \in [1..NR+1]$), $FlowCostPerDay(id)(r)$, also denoted $FlowCostperDay(id,r)$, gives the total non-labor cost per day for all input and output flows processed during release *r*, computed as follows:

$$FlowCostPerDay(id,r) = \sum_{j \in I(id)} (CostPerInput(id,j) \times InputThru(id,j,r)) + \sum_{j \in O(id)} (CostPerOutput(id,j) \times OutputThru(id,j,r))$$

7. Let $CostPerDay(id)\colon [1..NR+1] \rightarrow \mathbb{R}$ be a function described as follows:($\forall\, r \in [1..NR+1]$), $CostPerDay(id)(r)$, also denoted $CostperDay(id,r)$, gives the total cost per day during release *r*, computed as follows:

$$CostPerDay(id,r) = LaborCostPerDay(id,r) + FlowCostPerDay(id,r) + ServiceCostPerDay(id) \times On(id)$$

Constraints are as follows:

1. ***FeatureDependencyIsSatisfied*** (see computation #1)
2. ***DeactivatedServicesIsSatisfied*** (see computation #2)

InterfaceMetrics, also denoted **IM,** is a tuple ⟨*CostPerDay(id), InputThru(id), OutputThru(id)*⟩

where:

- $\boldsymbol{CostPerDay}(id)$ is defined in computation #7.
- $\boldsymbol{InputThru}(id)$ is defined in DecisionVariables.
- $\boldsymbol{OutputThru}(id)$ is defined in computation #3.

4.8 OutputDrivenAtomicService Formalization

Intuitively, an OutputDrivenAtomicService is an indivisible, atomic service which's throughput is driven by the number of outputs that it needs to produce, for example, a service that produces a report.

OutputDrivenAtomicService formalization is a tuple ⟨*Parameters, DecisionVariables, Computation, Constraints, InterfaceMetrics*⟩

Parameters, also denoted **Parm,** is a tuple $\langle id, ServiceType(id),\ I(id),\ O(id),\ RBF(id),\ ServiceRoles(id),\ OIthruRatio(id),\ RoleTimePerOI(id)\rangle$

where:

- ***id*** is the Service id.
- ***I(id)*** *is a set of inputs*

- $\boldsymbol{O(id)}$ *is a set of outputs*
- $\boldsymbol{RBF}(id) \subseteq ReleaseScheduling.Parm.BF$ is a set of business features required by Service *id*
- $\boldsymbol{ServiceRoles}(id) \subseteq BSN.Parm.LR$ is a set of roles involved in the business service.
- $\boldsymbol{OIthruRatio}(id){:}\,O(id) \times I(id) \rightarrow \mathbb{R}^{+}$ is a function described as follows: $\big(\forall\, o \in O(id)\big)$, $\big(\forall\, i \in I(id)\big)$, $OIOthruRatio(id)(i,o)$ also denoted as $OIthruRatio(id,i,o)$, gives for output o and input i, the ratio of input throughput based the output throughput.
- $\boldsymbol{RoleTimePerOI}(id){:}\,ServiceRoles(id) \times (O(id) \cup I(id)) \rightarrow \mathbb{R}^{+}$ is a function described as follows: $(\forall\, l \in ServiceRoles(id),\ \forall\, j \in O(id) \cup I(id))$, $RoleTimePerOI(id)(l,j)$, also denoted as $RoleTimePerOI(id,l,j)$, gives the amount of time, in hours, that role l spends per flow j.
- $\boldsymbol{ServiceCostPerDay}{:}\,ServicesSet \rightarrow \mathbb{R}^{+}$ is a function described as follows: $(\forall\, s \in ServicesSet)$, $FixedCostPerDay(s)$ gives the non-labor fixed cost of service *s* for each day.
- $\boldsymbol{CostPerInput}(id){:}\,I(id) \rightarrow \mathbb{R}^{+}$ is a function described as follows: $(\forall\, i \in I(id))$, $CostPerInput(id)$ gives the non-labor fixed cost for each input *i* processed by the service *id.*
- $\boldsymbol{CostPerOutput}(id){:}\ O(id) \rightarrow \mathbb{R}^{+}$ is a function described as follows: $(\forall\, o \in O(id))$, $CostPerOutput(id)$ gives the non-labor fixed cost for each output *o* processed by the service *id.*
- ***ServiceType****(id)* is *InputDrivenAtomicService*

DecisionVariables, also denoted **DV,** is a tuple $\langle On(id), OutputThru(id)\rangle$

where:

- $\boldsymbol{On}(id)$: $[1..NR+1] \rightarrow \{0,1\}$ is a function that determines whether the Service *id* is activated or not, for a particular release, i.e., $(\forall\, r \in [1..NR+1]), On(id)(r)$, also denoted by *On(id,r)* is as follows:

$$On(id,r) = \begin{cases} 1 \; if \; service \; id \; is \; activated \; in \, release \; r \\ 0 \; otherwise \end{cases}$$

- $\boldsymbol{OutputThru}(id)$: $O(id) \times [1..NR+1] \rightarrow \mathbb{R}^{+}$ is a function described as follows: $(\forall o \in O(id), \forall r \in [1..NR+1])$, $OutputThru(id)(o,r)$, also denoted $OutputThru(id,o,r)$, gives the throughput of o (or quantity per day) during release r or the period after the last release.

Computation

1.FeatureDependencyIsSatisfied constraint:

$$On(id,r) = 1 \rightarrow RBF(id) \subseteq RSch.IM.SoFarIBF(r) \quad \forall\, r \in [1..NR+1]$$

2.DeactivatedServicesIsSatisfied constraint:

$\forall\, o \in O(id), \forall r \in [1..NR+1]$,

$On(id,r) = 0 \rightarrow OutputThru(id,o,r) = 0$

*3.*Let $InputThru(id)$: $I(id) \times [1..NR+1] \rightarrow \mathbb{R}^{+}$ be a function described as follows: $(\forall\, i \in I(id), \forall\, r \in [1..NR+1])$, In$putThru(id)(i,r)$, also denoted $InputThru(id,i,r)$, gives the throughput of i (or quantity per day) during release r or the period after the last release, computed as

$\forall\, i \in I(id), \forall\, r \in [1..NR+1]$,

$$InputThru(id,i,r) = \sum_{o\in O(id)} (OIthruRatio(id,o,i) \times OutputThru(id,o,r))$$

*4.*Let $TimePerDay(id): [1..NR+1] \times ServiceRoles(id) \to \mathbb{R}^+$ be a function described as follows: $(\forall\ l\ \in ServiceRoles(id), r \in [1..NR+1])$, $TimePerDay(id)(l,r)$, also denoted $TimePerDay(id,l,r)$, gives the total duration per day for role l and release r (and the period after the last release), computed as:

$$TimePerDay(id,l,r)$$
$$= \sum_{j\in I(id)} (RoleTimePerOI(id,l,j) \times InputThru(id,j,r))$$
$$+ \sum_{j\in O(id)} (RoleTimePerOI(id,l,j) \times OutputThru(id,j,r))$$

*5.*Let $LaborCostPerDay(id): [1..NR+1] \to \mathbb{R}$ be a function described as follows:$(\forall\ r \in [1..NR+1])$, $LaborCostPerDay(id)(r)$, also denoted $LaborCostperDay(id,r)$, gives the total labor cost per day during release *r*, computed as follows:

$$LaborCostPerDay(id,r)$$
$$= \sum_{l\in ServiceRoles} (BSN.Parm.Rate(l) \times TimePerDay(id,l,r))$$

*6.*Let $FlowCostPerDay(id): [1..NR+1] \to \mathbb{R}$ be a function described as follows:$(\forall\ r \in [1..NR+1])$, $FlowCostPerDay(id)(r)$, also denoted $FlowCostperDay(id,r)$, gives the total non-labor cost per day for all input and output flows processed during release *r*, computed as follows:

$$\begin{aligned}FlowCostPerDay(id,r) &= \sum_{j\in I(id)} (CostPerInput(id) \times InputThru(id,j,r)) \\ &+ \sum_{j\in O(id)} (CostPerOutput(id) \times OutputThru(id,j,r))\end{aligned}$$

*7.*Let $CostPerDay(id)\colon [1..NR+1] \to \mathbb{R}$ be a function described as follows:($\forall\, r \in [1..NR+1]$), $CostPerDay(id)(r)$, also denoted $CostperDay(id,r)$, gives the total cost per day during release *r*, computed as follows:

$$\begin{aligned}CostPerDay(id,r) &= LaborCostPerDay(id,r) + FlowCostPerDay(id,r) \\ &+ ServiceCostPerDay(id) * On(id)\end{aligned}$$

Constraints are as follows:

1. ***FeatureDependencyIsSatisfied*** (see computation #1)
2. ***DeactivatedServicesIsSatisfied*** (see computation #2)

InterfaceMetrics, also denoted **IM,** is a tuple ⟨*CostPerDay(id), InputThru(id), OutputThru(id)*⟩

where:

- $\boldsymbol{CostPerDay}(id)$ is defined in computation #7.
- $\boldsymbol{InputThru}(id)$ is defined in DecisionVariables.

$\boldsymbol{OutputThru}(id)$ is defined in computation #3.

4.9 Software Development Formalization

SoftwareDevelopment formalization, also denoted **SWD**, is a tuple ⟨*Parameters, DecisionVariables, Computation, Constraints, InterfaceMetrics*⟩

where:

Parameters, also denoted **Parm**, is a tuple ⟨*TS, DP, DC, OC, SS, SWPaySched*⟩,

where:

- $\boldsymbol{TS} : [1..NR] \rightarrow \mathbb{R}^{+}$ is a function that gives the team size, in full time equivalents, for each release.
- $\boldsymbol{DP} : [1..NR] \rightarrow \mathbb{R}^{+}$ is a function that gives the developer productivity for each release in effort points per day.
- $\boldsymbol{DC} \in \mathbb{R}^{+}$ is the developer cost in dollars per effort point.
- $\boldsymbol{OC} \in \mathbb{R}^{+}$ is the operations cost in dollars per effort point per day.
- $\boldsymbol{SS} \in \mathbb{R}^{+}$ is the size, in effort points, of the As-Is system (prior to development).
- ***SWPaySched***, the software cost payment schedule, is a tuple $\langle NSP, SWPayDays \rangle$, where:
    - $NSP \in \mathbb{R}^{+}$is the number of payments to the software team over the entire time horizon.
    - $SWPayDay: [1..NSP] \rightarrow [1..TH]$ is a function, i.e. $(\forall\, p \in [1..NSP])$, $SWPayDay(p)$ gives the day (relative to the first day of the software development project) where payment p is made.

DecisionVariables, also denoted **DV**, is an empty tuple**.**

Computation:

1. Let $RS : [1..NR] \rightarrow \mathbb{R}^{+}$ be a function described as follows: $(\forall\, r \in [1..NR]), RS(r)$ gives the actual size, in effort points, of release r, once features are assigned to it. The computation is as follows:

$$RS(r) = \left(\sum_{j \in RSch.DV.IBF(r)} RSch.Parm.FS(j) + \sum_{j \in RSch.DV.ITF(r)} RSch.Parm.FS(j) \right)$$

2. Let $RC : [1..NR] \rightarrow \mathbb{R}^{+}$be a function described as follows: $(\forall\, r \in [1..NR]), RC(r)$ gives the maximum capacity, in effort points, for release r computed as:

$$RC(r) = \begin{cases} TS(r) \times DP(r) \times RSch.Parm.RD(r) & if\ RS(R) > 0 \\ 0 & if\ RS(R) = 0 \end{cases}$$

3. $ReleaseSizeCannotExceedCapacity$ constraint:

$$0 \leq RS(r) \leq RC(r) \quad \forall\, r \in [1..NR]$$

4. Let $firstDay : [1..NR+1] \rightarrow \mathbb{R}^{+}$be a function described as follows: $(\forall\, r \in [1..NR+1])$, $firstDay(r)$ gives the day when release r actually starts, computed as:

$$firstDay(r) = \begin{cases} 1 \quad r = 1 \\ Rsch.Parm.RD(r) + firstDay(r-1) \forall\, r = [2..NR+1] \end{cases}$$

5. Let $lastDay : [1..NR+1] \rightarrow \mathbb{R}^{+}$ be a function described as follows: $(\forall\, r \in [1..NR+1])$, $lastDay(r)$ gives the day when release r ends, computed as:

$$lastDay(r) = \begin{cases} firstDay(r+1) - 1 & r = [1..NR] \\ RSch.TH & (r = NR + 1) \end{cases}$$

6. Let $devCostPerDay : [1..NR + 1] \rightarrow \mathbb{R}^{+}$ be a function described as follows: ($\forall\, r \in [1..NR + 1]$), $devCostPerDay(r)$ gives the dollar cost of development per day for release r, computed as:

   $\forall\, r = [1..NR + 1]$,

$$devCostPerDay(r) = \begin{cases} \left(\dfrac{RC(r)}{RSch.Parm.RD(r)} \times DC\right) & (\ \forall r = [1..NR]) \\ 0 \quad (r = NR + 1) \end{cases}$$

7. Let $opsCostPerDay : [1..NR + 1] \rightarrow \mathbb{R}^{+}$ be a function described as follows: ($\forall\, r \in [1..NR + 1]$), $opsCostPerDay(r)$ gives the dollar cost of operations per day for release r, and the period after the last release, computed as:

$$opsCostPerDay(r) = \begin{cases} (SS \times OC) & r = 1 \\ ((\sum_{i=1}^{r-1} RC(i)) + SS) \times OC & \forall\, r = [2..NR + 1] \end{cases}$$

8. Let $SWCostForDay : [1..TH] \rightarrow \mathbb{R}^{+}$be a function described as follows: ($\forall\, d \in [1..TH]$), $SWCostForDay(d)$ gives the software cost accrued for each day d in the time horizon, computed as:

   $SWCostForDay(d) = devCostPerDay(r) + opsCostPerDay(r)$

where *r* is the release period (or period after the last release), where day *d* appears, i.e.,

$$firstDay(r) \leq d \leq lastDay(r)$$

9. Let $SWPayment\colon [1..NSP] \rightarrow \mathbb{R}$ be a function described as follows: $(\forall\, p \in [1..NSP])$, $SWPayment(d)$ gives the software payment in dollars, for each scheduled payment p, computed as follows:

$$SWPayment(p) = \begin{cases} \displaystyle\sum_{d=1}^{SWPayDay(p)} SWCostForDay(d) & p = 1 \\ \displaystyle\sum_{d=SWPayDay(p-1)+1}^{SWPayDay(p)} SWCostForDay(d) & p = [2.NSP] \end{cases}$$

10. Let $LaborCashFlow : [1..TH] \rightarrow \mathbb{R}^{+}$, be a function described as follows: $(\forall\, d \in [1..TH])$, $LaborCashFlow(d)$ gives the cash flow of software labor cost for day d, is computed as:

$$LaborCashFlow(d) = \begin{cases} -SWPayment(p)\ if\ \exists p | d = SWPayDay(p) \\ 0 \qquad\qquad\qquad\qquad\qquad Otherwise \end{cases}$$

11. Let $RelRes\colon [1..NR] \rightarrow\ 2^{ResSet}$ be a function described as follows: $(\forall\, r \in [1..NR])$, $RelRes(r)$ gives the set of all resources required by release r, computed as:

$$RelRes(r) = \bigcup_{f \in IBF(r) \cup ITF(r)} RSch.Parm.FeatureRes(f) \quad \forall\, r \in [1..NR]$$

12. Let $CumRelRes\colon [1..NR] \rightarrow\ 2^{RSch.Parm.ResSet}$ be a function described as follows: $(\forall\, r \in [1..NR])$, $CumRelRes(r)$ gives the cumulative set of resources required by all releases up to r, computed as:

$$CumRelRes(r) = \begin{cases} \emptyset & if\ r = 0 \\ \displaystyle\bigcup_{i=1}^{r} RelRes(i) & if\ r > 0 \end{cases}$$

13. Let $NewRelRes: [1..NR] \rightarrow 2^{ResSet}$ be a function described as follows: $(\forall r \in [1..NR])$, $NewRelRes(r)$ gives the set of new resources required by release r that were not paid in a previous release, computed as:

$$NewRelRes(r) = RelRes(r) - CumRelRes(r-1) \quad \forall r \in [1..NR]$$

14. Let $ResCostPerRel : [1..NR] \rightarrow \mathbb{R}^{+}$be a function described as follows: $(\forall r \in [1..NR])$, $ResCostPerRel(r)$ gives the cost of all resources that need to be paid in release *r* and were not paid in a previous release, computed as:

$$ResCostPerRel\ (r) = \sum_{e \in NewRelRes(r)} RSch.Parm.ResCost(e)\ \forall r \in [1..NR]$$

15. Let $ResCashFlow : [1..TH] \rightarrow \mathbb{R}^{+}$be a function described as follows: $(\forall d \in [1..TH])$, $ResCashFlow(d)$ gives the resource cash flow for each day d in the time horizon, computed as follows:

Let *r(d)* be a release during which day d occurs, i.e.,*firstDay(r(d))≤d≤lastDay(r(d))*

$$ResCashFlow(d) = \begin{cases} -ResCostPerRel(r) & where\ d = firstDay(r(d)) \\ 0 & Otherwise \end{cases}$$

$\forall d \in [1..TH]$

16. Let $CashFlow : [1..TH] \rightarrow \mathbb{R}^{+}$, be a function described as follows: $(\forall d \in [1..TH])$, $CashFlow(d)$ gives the total cash flow of software cost for day d, is computed as:

$$CashFlow(d) = LaborCashFlow(d) + ResCashFlow(d) \qquad \forall d \in [1..TH]$$

Constraints

1. $\boldsymbol{ReleaseSizeCannotExceedCapacity}$ (defined in computation #3)

InterfaceMetrics, also denoted **IM,** is a tuple $\langle LaborCashFlow, firstDay, lastDay\rangle$, where:

• $\boldsymbol{CashFlow}(d)$ is defined in computation #16.

• $\boldsymbol{firstDay}(r)$ is defined in computation #4.

• $\boldsymbol{lastDay}(r)$ is defined in computation #5.

4.10 Optimization Formulation

The formalizations in the previous sections are building blocks; we now use them to formulate the optimization of the NPV of the BSN configurations. Given the top-level formal optimization model

$ReleaseSchedule\langle Parms, DVs, Computation, Constraints, InterfaceMetrics\rangle$,

The optimal NPV for a time horizon of th days, is:

$$NPV = Maximize \quad ReleaseSchedule.IM.TimeWindowNPV(th)$$
$$s.t. \quad ReleaseSchedule.Constraints$$

Note that *ReleseSchedule.Constraints* is a roll up of all constraints from all components. Solving the MILP above produces:

- Optimal NPV that satisfies the *Demand* parameter
- A release schedule, which is the result of the Solver instantiating *IBF(r,f)* and *ITF(r,f)*.

- The BSN configuration at the end of each release, which is captured by the instantiated variables *On(s,r)*.

5. DECISION GUIDANCE SYSTEM AND METHODOLOGY

The *Opti-Soft+* framework includes a Decision Guidance Systems (DGSs) that is implemented in JSONiq and uses the Unity platform [29], [30], so we now describe Unity.

5.1 Unity DGMS

Decision Guidance Systems (DGSs) are an advanced class of Decision Support Systems (DSS) that are designed to provide "actionable recommendations, typically based on formal analytical models and techniques" [9]. DGSs usually employ one or more models and techniques from the following categories: descriptive, predictive and prescriptive. Because these categories cover a wide range of techniques like database, stochastic simulation, statistical learning, optimization and machine learning among others, historically DGSs have been custom built to solve a particular problem. The challenge in developing a DGS is that it has to be developed from scratch and requires significant domain-specific expertise with mathematical modeling and Operations Research.

Opti-Soft+ DGS relies on the Unity platform [29], [30], which addresses the built-from-scratch problem by providing a platform for building DGSs from reusable analytical models. Modular, reusable and composable models, called analytical models (AMs), are created and stored in a repository. The AMs are at a high level of abstraction and are translated by Unity into low level code that is specific to a particular mathematical tool.

Figure 7 shows Unity's reference architecture. Unity serves as a middleware between the client layer and the external tool layer.

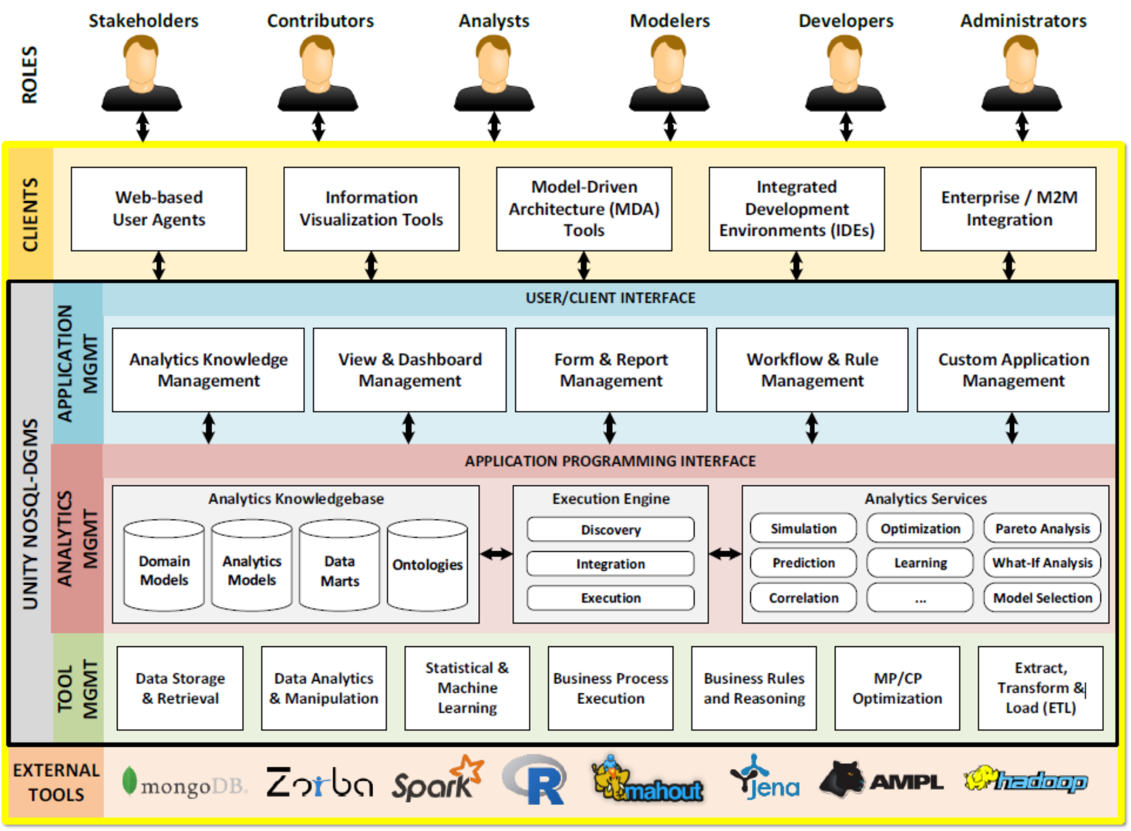


Figure 7 - Unity's Reference Architecture as presented in [29]

The following are some characteristics of Unity:

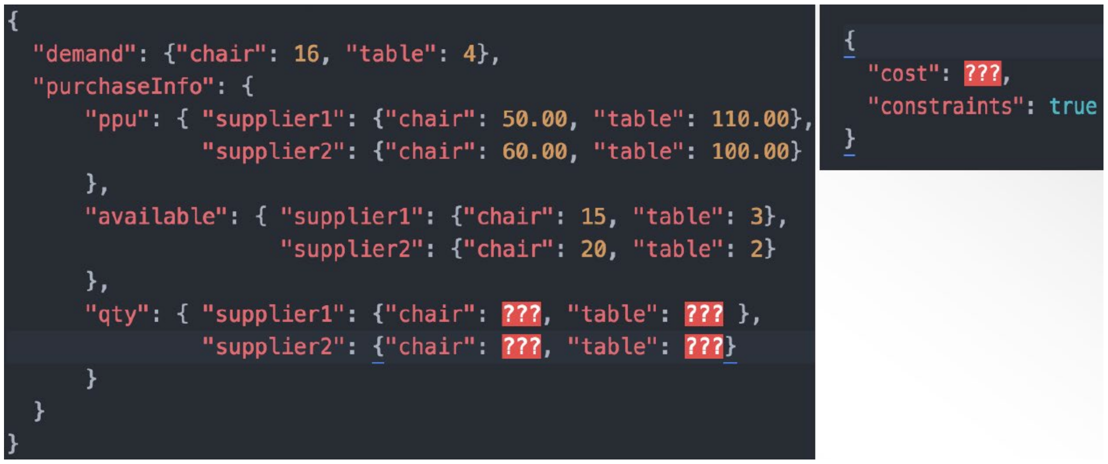
- Reusable Analytical Modules that are stored in the Analytical Management layer
- Unified, high-level language to develop the AMs that supports data manipulation, deterministic and stochastic computation, decision optimization based on Mathematical Programming/Constraint Programing and statistical/machine learning. The language provides a level of abstraction to hide the complexities of dealing with a diverse range of lower-level tools that are needed to implement a DGS.

- An algebra of operators that can be applied to AMs. The operators are in the Analytics Services layer
- The analytics management layer contains a compiler that uses the AMs to generate code for the tool management layer.
- The tool layer contains a wide range of tools that are invoked by the Analytical Services but are transparent to the end user, who writes code in the high-level language.

Unity's unified, high-level language is Decision Guidance Analytics Language (DGAL) [9], which is used for defining AMs and for executing analytical services against those models. DGAL is based on the JSON Query Language (JSONiq), which is a query and processing language for manipulation of JSON structures. JavaScript Object Notation (JSON) is a simple language that uses human-readable text to express complex data structures. JSON expressions are either key-value pairs, e.g. "firstName":"John", sequence of key-value pairs or arrays of key-value pairs. JSON is highly used as a simpler alternative to XML.

In Unity, a Performance Model, which is basically a DGS, has three parts: 1) fixed parameters *P*, 2) decision variables *V* and 3) an analytical model *AM*. *P* and *V* are coded as JSON expressions while *AM* is a coded in JSONiq. To perform maximization, the Unity function *argmax* is invoked while for minimization, the function *argmin* is used. Internally, Unity invokes a mathematical programming solver to compute the optimization. The parameters in *P* are regular key-value expressions, while the decision variables in *V* are annotated key-value pairs, where the key part contains the name of the decision variable, and the value part contains a question mark. Both are combined in a parameter file, which is then input to the AM program.

Figure 8, presented in [10], shows an example of a Unity input parameter file in JSON with annotated *V*s. The parameters *demand, ppu* and *available* have fixed values while the decision variables *chair* and *table* are annotated with question marks. The example is a simple Linear Program to determine the optimal purchase of tables and chairs to fulfil a demand of 4 tables and 16 chairs so that the cost is minimized. There are two suppliers; *supplier1* has the best cost for chairs while *supplier2* has the best cost for tables. There are no constraints, so the *constraints* variable on the right pane is set to *true*. Note that the objective function result is also annotated with question marks.



```
{
  "demand": {"chair": 16, "table": 4},
  "purchaseInfo": {
      "ppu": { "supplier1": {"chair": 50.00, "table": 110.00},
               "supplier2": {"chair": 60.00, "table": 100.00}
      },
      "available": { "supplier1": {"chair": 15, "table": 3},
                     "supplier2": {"chair": 20, "table": 2}
      },
      "qty": { "supplier1": {"chair": ???, "table": ??? },
               "supplier2": {"chair": ???, "table": ???}
      }
  }
}
```

```
{
  "cost": ???,
  "constraints": true
}
```

Figure 8 - Annotated Decision Variables in Unity as presented in [10]

Figure 9, also presented in [10], shows a schematic view of the optimization process in Unity. The parameters, annotated decision variables, objective function and analytical module are provided as input to Unity's *argmax/argmin* function. Unity then generates code in the low-level language for the optimizer, runs it, and the optimizer determines the optimal values for the decision variables and instantiates them.

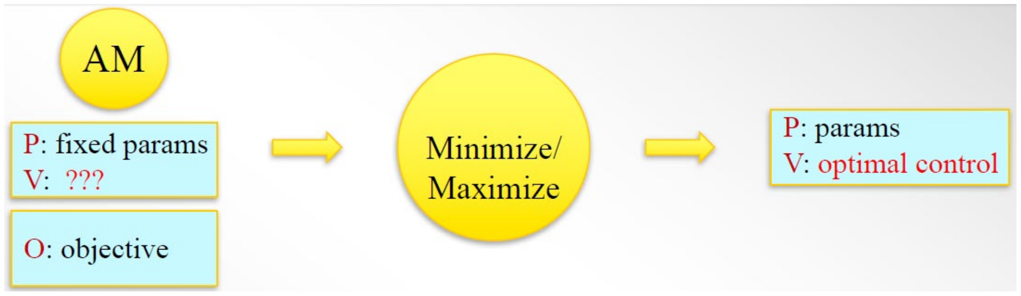
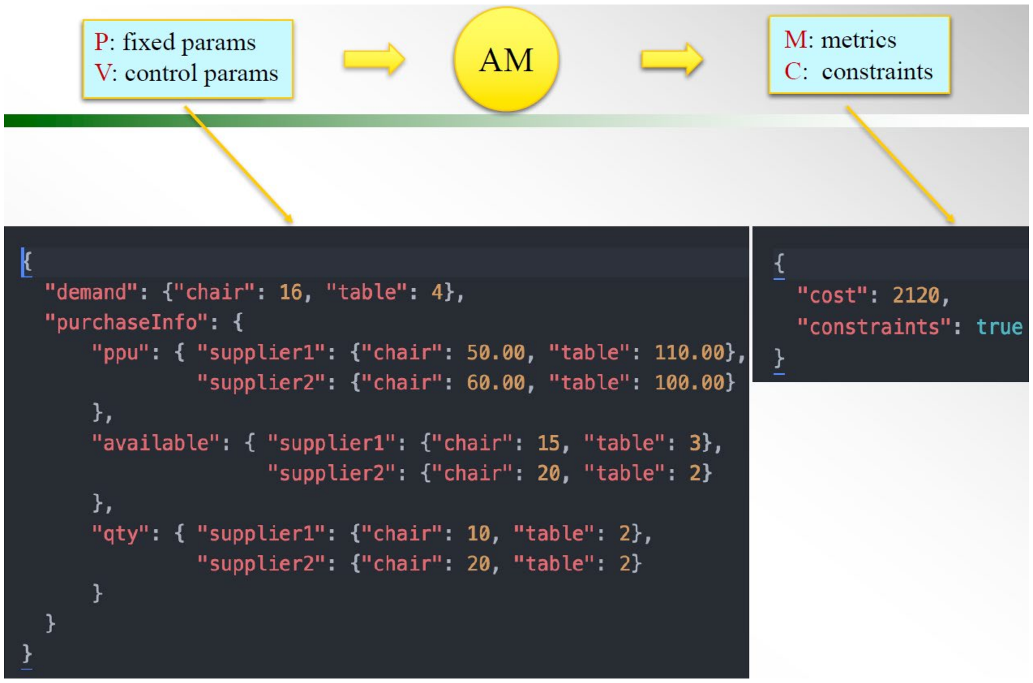


Figure 9 - Schematic View of Unity's Optimization Process as presented in [9]

Once the decision variables are instantiated, they are then provided as input to the analytical module (AM), which then computes the objective function. Figure 10 from [10] shows the result for the example, but it does not show the code to compute the objective function, which is in the AM file. The decision variables are instantiated for an optimal purchase of 10 chairs and 2 tables from *supplier1* and 20 chairs and 2 tables from *supplier2,* at a cost of 2120.



```
{
  "demand": {"chair": 16, "table": 4},
  "purchaseInfo": {
      "ppu": { "supplier1": {"chair": 50.00, "table": 110.00},
               "supplier2": {"chair": 60.00, "table": 100.00}
      },
      "available": { "supplier1": {"chair": 15, "table": 3},
                     "supplier2": {"chair": 20, "table": 2}
      },
      "qty": { "supplier1": {"chair": 10, "table": 2},
               "supplier2": {"chair": 20, "table": 2}
      }
  }
}
```

```
{
  "cost": 2120,
  "constraints": true
}
```

Figure 10 - Example of Optimization of a Cost Metric, as presented in in [10]

The Analytical Module is written in DGAL and a DGAL module is a JSONiq module with specialized functions that extend the semantics of JSONiq. DGAL's specialized functions have two properties: 1) the input and output of the function must be an indexable object, and 2) the output object must contain the key *constraints* with a value of type Boolean. The bottom right pane in Figure 10 shows the output object of a DGAL AM. The *constraints* key is an integrity constraint on the input module on the left of the figure and must result in *true*.

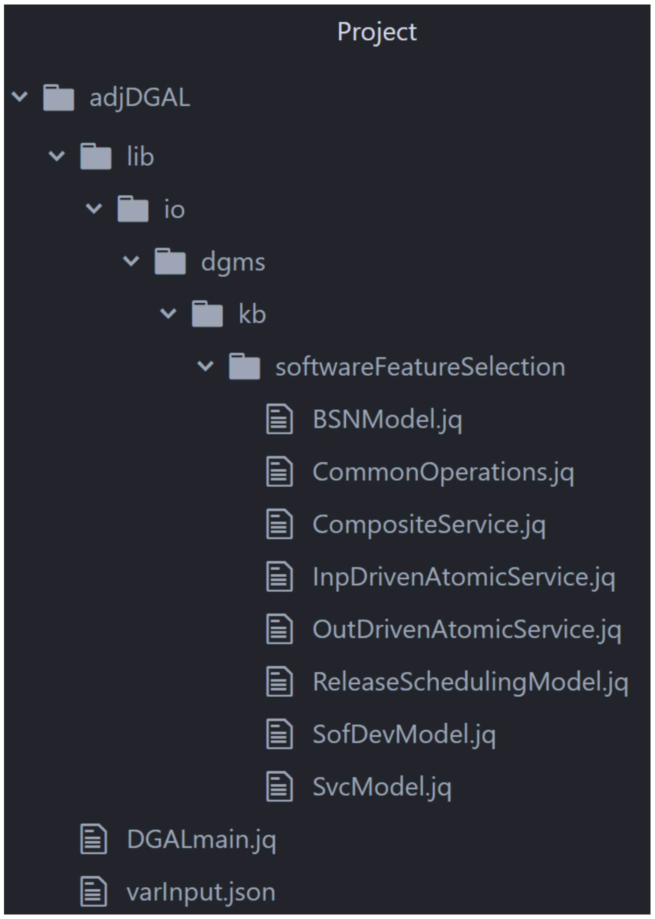
The JSONiq language, which is the basis for DGAL, processes and produces JSON data structures. Based on XQuery, it is set-oriented and declarative, that is, it does not describe how, it describes what to compute. It is also functional; every construct is an expression, a program is an expression and the result of a program is an expression. The language contains very powerful expressions, from simple JSON key/value pairs to programmatic expressions such as *let, if,* and the XQuery FLWOR (For, Let, Where, Order by and Return) construct, which corresponds to SQL's SELECT-FROM-WHERE statements, but is more general and more flexible.

Now that we described the underlying technology used by Unity, we describe the *Opti-Soft+* DGS, or Performance Model.

5.2 *Opti-Soft+* DGS

In Unity, a Performance Model, which is basically a DGS, has three parts: 1) fixed parameters *P*, 2) decision variables *V* and 3) an analytical model *AM*. *P* and *V* are coded as JSON expressions while *AM* is a module coded in JSONiq. To perform maximization, the Unity function *maximize* is invoked while for minimization, the function *minimize* is used. Internally, Unity invokes a mathematical programming solver to compute the optimization. A Performance Model (PM) formally describes feasibility constraints and metrics of interest, such as cost, as a function of fixed parameters and decision variables.

In Section 4, we showed that the *Opti-Soft+* formal model is made of eight components and broke down each component in the subparts: parameters, decision variables, computation, constraints and interface metrics. Our strategy for implementing the *Opti-Soft+* formal components in Unity is to map each component to a Analytical Module and then to develop a Performance Model that combines all the AMs. A single JSON module encodes all parameters *P* and decision variables *V*, while every formal model component (tuple) corresponds to a JSONiq/DGAL analytical module *AM* that encodes its computations, constraints and metrics. The resulting set of JSON and JSONiq modules comprises the *Opti-Soft+* DGS. Figure 11 below shows the JSONic/DGAL analytical modules, while Table 14 shows the mapping of analytical modules programming files to formal components. Files with extension .jq are JSONiq programs while files with extension .json are JSON files.



```
Project
adjDGAL
  lib
    io
      dgms
        kb
          softwareFeatureSelection
            BSNModel.jq
            CommonOperations.jq
            CompositeService.jq
            InpDrivenAtomicService.jq
            OutDrivenAtomicService.jq
            ReleaseSchedulingModel.jq
            SofDevModel.jq
            SvcModel.jq
  DGALmain.jq
  varInput.json
```

Figure 11 - DGS JSONic Modules

Table 14 - Mapping of Formal Components to JSONiq AM Modules

Formal Component	JSONiq AM Module
ReleaseScheduling	ReleaseSchedulingModel.jq
BusinessServiceNetwork	BSNModel.jq
Service	SvcModel.jq
ANDservice	CompositeService.jq
ORservice	CompositeService.jq
InputDrivenAtomicService	InputDrivenAtomicService.jq
OutputDrivenAtomicService	OutputDrivenAtomicService.jq
SoftwareDevelopment	SofDevModel.jq

Because the formalization of *ANDservice* and *ORservice* are very similar, we combine them in one module, *CompositeService*.jq. Note that the formal model allows a composite service to be comprised of composite services. To implement this, *CompositeService.jq,* uses recursion to call itself.

JSONiq module `DGALMain.jq`, shown above, is the DGS Performance Module, which encompasses the parameters *P*, decision variables *V* and the highest level analytical module *ReleaseSchedulingModel*. We reproduce it below, where:

- *P* and *V* are in `varInPut.json`
- *AM* = `ReleaseSchedulingModel.jq`

```
Jsoniq version "1.0";
import module namespace a = "http://dgms.io/modules/analytics";
import module namespace fetch = "http://zorba.io/modules/fetch";
import module namespace rs =
"http://kb.dgms.io.lib/softwareFeatureSelection/ReleaseSchedulingModel
.jq";

let $varInput :=
    jn:parse-json(fetch:content("varInput.json"))

let $optResult := a:maximize({
    model: rs:ReleaseSchedulingModel#1,
    input: $varInput,
    objective: function($output) {
        $output.metrics.timeWindowNPV[][520]},
    options: {
        solver: "cbc"
    }
})

return {
  varInput: $varInput,
  optResult: $optResult
}
```

Figure 12 - DGALMain.jq

The `DGALMain.jq` program above, which is the DGS Performance Module, runs under Docker and does the following:

1. Defines the project environment where the files are contained
2. Reads file `varInput.json`, parses it as a JSON expression and stores it in `$varInput`
3. Invokes Unity function `a:maximize` with the following parameters:
    - Analytical Module = `ReleaseSchedulingModel.jq`,
    - Input structure = `$varInput`,
    - Objective function = `timeWindowNPV[][520]`, which is based on the example in section 3.2.

- Optimizer solver = cbc

4. Returns as output, the original input structure as well as the result of the optimization.

We now show, in Figure 13, a snippet of the `varInput.json` file for the example in section 3.2. It contains both parameters and decision variables, and the snippet declares service "*AB*" of type "*InputDrivenAtomic*". Service "*AB*" requires business feature "*BF1*" and has a decision variable "*onDV*" for each of five investment periods.

```
    "AB": {
      "serviceType": "InputDrivenAtomic",
      "requiredBusinessFeature": ["BF1"],
      "onDV": [
        {"integer?": null},
        {"integer?": null},
        {"integer?": null},
        {"integer?": null},
        {"integer?": null}
      ]
      …
```

Figure 13 - `varInput.json` File, which Contains Parameters and Decision Variables

Below we show a snippet of the analytical module `SofDevModel.jq`, which is the implementation of the formal model *SoftwareDevelopment* from section 4.9. The third line computes the release capacity for every release; it corresponds to computation #1 (*RC*) in the formal model. The snippet also shows computation #16 for *CashFlow*. The last 4 lines show that it outputs the metric *CashFlow* as well as one single constraint called *releaseSizeCannotExceedCapacity*. The full source code is in Appendix 1.

```
declare function sd:SofDevModel($inpData) {
  (:-- Calculate release capacity   --:)
  let $relCapacity := [
    for $r in 1 to $noRel
      return
        $inpData.SofDev.teamSize[[$r]] *
        $inpData.SofDev.devProductivity[[$r]] *
        $inpData.RelSch.releaseDuration[[$r]]
  ]
 ...
 (:-- Calculate Cash Flow  --:)
 let $cashFlow := [
    for $d in 1 to $timeHorizon
    return $laborCashFlow[[$d]] + $resCashFlow[[$d]]
  ]
  return {
    constraints : $releaseSizeCannotExceedCapacity,
    metrics: {cashFlow: $cashFlow}
  }
```

Figure 14 - Snippet of SofDevModel.jq

The program `ReleaseSchedulingModel.jq`, being the top-level AM invoked by the PM `DGALMain.jq`, concatenates the constraints from all children modules with an AND operation. Because a solution to the MILP guarantees that `ReleaseSchedulingModel.constraints` is true, it indirectly guarantees that each constraint for each of the other seven AMs is also true.

Unity supports a large number of solvers via CasADi and Pyomo. *Opti-Soft+* DGS is translated into Pyomo into a MILP that is then executed by the solver. The solver is CBC, which is open-source and uses a branch and cut method. Pyomo is a Python-based open-source software package that supports a diverse set of optimization capabilities for formulating, solving, and analyzing optimization models. Its core capability is modeling structured optimization applications, and it can be used to define general symbolic problems through its high-level programming language. Unity compiles DGAL code into Pyomo and feeds it into the solver.

The use of Pyomo restricts the JSONiq code that can be used in the DGS in two ways: 1) only one decision variable can be used in a conditional statement and 2) the use of arrays of decision variable is not supported in some situations.

As in any MILP program, the constraints that the objective function is subject to have to be expressed in equalities or inequalities. Below is a sample of the constraints:

Constraint 1 Name: *ReleaseSizeCannotExceedCapacity*

Equation: $0 \leq RS(r) \leq RC(r) \quad \forall\, r \in [1..NR]$

Constraint type: simple inequality

Constraint: *relSize* ≤ *relCapacity*

Code:
```
every  $r in 1 to $noRel
       satisfies $relSize[[$r]] <= $relCapacity[[$r]]
```

Constraint 2 Name: *DeactivatedServicesIsSatisfied*

Equation:

Constraint type: If-then

Constraint: $-M * On(id,s) \leq InputThru$

$InputThru \leq M * On(id,s)$

Where M is an arbitrarily large number

5.3 METHODOLOGY

The goal of the methodology is to guide a decision-maker through the steps to setup and run the DGS to get a recommendation. The methodology contains the following major steps:

1. Generate candidate software features to be implemented
2. Capture the As-Is BSN configuration, and alternative BSN configurations that can be enabled by candidate software features
3. Gather and instantiate input parameters for the optimization model as described in Section 4, including the set of features, their dependencies, time horizon, number of releases, BSN services, etc…
4. Run the DGS to compute the baseline NPV for the As-Is BSN
5. Perform DGS optimization to come up with a recommended Release Plan (including chosen software features in each release) and the associated optimal BSN configuration (To-Be)
6. Calculate the savings, which is the NPV of the To-Be minus the NPV of the As-Is
7. During Release 1, use the recommendation in the previous step to:
   a) Operate the BSN according to the optimal BSN configuration.
   b) Implement the recommended software features for Release 1
8. For each release r = 2,…,n
   a) Update existing software features to include those implemented in the previous release

b) For the updated software features and refined demand/throughput requirements, run the DGS to recommend a new Release Plan and BSN configuration for the remaining releases (starting from Release r + 1)

c) Implement recommended software features and operate the BSN according to recommendation.

In step 1, the candidate software development features are determined, and each feature has its size estimated in effort points. Also the dependency graph is defined. The parameters that capture features are:

Parameter	Content
BF	Set of business features from the Product Backlog
TF	Set of technical features
DG	The feature dependency graph
FS	Maps each feature to a size in points

In step 2, a model of the Business Service Network is created. The first task is to create a diagram of the As-Is service network showing each subservice, its inputs and outputs. Then for each subservice, the following is determined: labor roles, the amount of time that each role spends to process one unit of input or to produce one unit of output, and the ratio between input and output. Each subservice as well as the entire As-Is service network are given identifiers. The type of each subservice is set to InputDrivenAtomicService or InputDrivenAtomicService and the type of the As-Is service network is set to ANDservice.

Next, the impact of each feature in the business network is analyzed with the goal of determining whether the feature, once delivered, would have a positive effect either by reducing the production time or by allowing that a role with a lower rate be employed. If a feature *f* has a positive impact in service *s* then a new service *s1* is created, and its labor role, rate, time and ratio parameters are determined. A new composite service of type ORservice is created where *s* and *s1* are its subservices. Also feature *f* is added to the set of required business features for service *s1*. This process continues until the impact of all features is fully modeled.

The complete service model has the following parameters for each composite service:

Parameter	**Content**
Id	Identifier of the service
I	Set of inputs
O	Set of outputs
ServiceType	ORservice or ANDservice
Subservices	Set of ids of the subservices

The complete service model has the following parameters for each atomic, that is, non-composite service:

Parameter	**Content**
Id	Identifier of the service
I	Set of inputs
O	Set of outputs
ServiceType	InputDrivenAtomicService or InputDrivenAtomicService

Parameter	Content
RBF	Set of business features required by the service
ServiceRoles	Set of roles involved in the business service
IOthruRatio	The amount of time in hours that each role spends to process one unit of input or output.
RoleTimePerIO	The amount of time in hours that each role spends to process one unit of input or output
ServiceCostPerDay	Non-labor fixed cost of the service for each day
CostPerInput	Maps each input to their non-labor fixed cost
CostPerOutput	Maps each output to their non-labor fixed cost

In step 3, the following global parameters are determined for the release schedule and the business service network:

Parameter	Content
TH	The investment time horizon
DiscountRate	The daily rate to discount the cash flows
NR	The number of releases
RD	The duration of each release in days
ResSet	Set of non-labor resources that have a fixed cost
ResCost	Maps each non-labor resource to a fixed cost
FeatureRes	Maps each feature to a set of resources
LR	Set of labor roles
Rate	Maps labor rates to labor roles
NLP	The number of labor payments over the entire investment time horizon
LaborPayDay	A vector of the days when a labor payment is made
BSNI	Set of input items that have to be processed by the BSN
BSNO	Set of input items that have to be processed by the BSN
Demand	The required processing throughput per day
RootID	The id of the top-level BSN

Also in step 3, the software development parameters described in section 4.9 are instantiated. These parameters describe the size of the team, cost, productivity, etc… Below is the full list of software development parameters:

Parameter	**Content**
TS	Team size in full time equivalents
DP	The expected developer productivity in points per day
DC	The developer cost in dollars per effort point
OC	The operations cost in dollars per effort point per day.
SS	The size, in effort points, of the As-Is system (prior to development).
NSP	The number or payments to the software team over the entire time horizon.
SWPayDay	A vector of the days when a software labor payment is made

Also in step 3, the values of the parameters above are codified as a single JSON object. In the following steps the DGS is ran to produce the As-Is NPV, the optimal NPV of the To-Be and the release schedule.

6. EXAMPLE

We now give an example of the formalization. The example is a continuation of the Application Adjudication from section 3.2.1.

In step 1 of the methodology, we identify the candidate features. The candidate features for the example are TF1, BF1, BF2, BF3 and BF4. Below are their descriptions and sizes.

Table 15 – Candidate Features

id	Type	Functionality	Size in points
TF1	Technical	Establish technical infrastructure: servers, database, environments, developer tools, application architecture	140
BF1	Business	eApplication. Capability to create and edit an electronic application. Edits enforce data types, consistency and completion rules.	140
BF2	Business	eAdjudication. Capability to annotate aspects of the application that pass or does not pass adjudication rules	280
BF3	Business	eReview. Capability to review adjudication decision and annotate issues that do not pass review.	280
BF4	Business	Self-service. Capability to allow an Applicant to submit an application on-line, update it, and monitor its workflow	280

The features dependency graph is shown below, where BF2 depends on BF1:

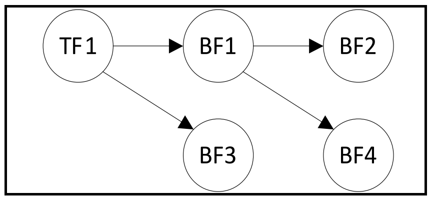


Figure 15 - Feature Dependency Graph

In step 2, we model the BSN. The As-Is BSN shown in Figure 2 is reproduced below. It is composed of three services: AA, BA and CA.

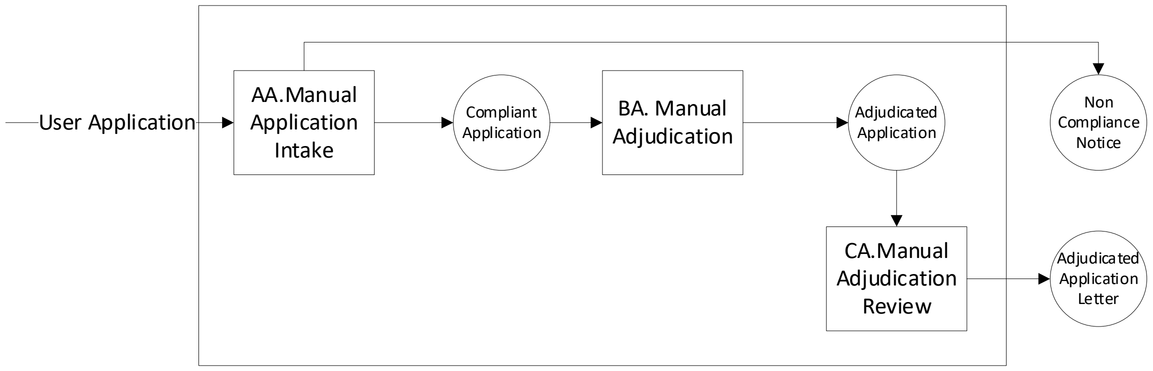


Figure 16 - As-Is BSN Configuration

We generalize the BSN with the model below, which covers the entire space of alternatives. In the generalized service, called Adj, the BSN is comprised of composite services A, B and C and all these services need to be present because the type of the top-level service, aka, the root service, is AND.

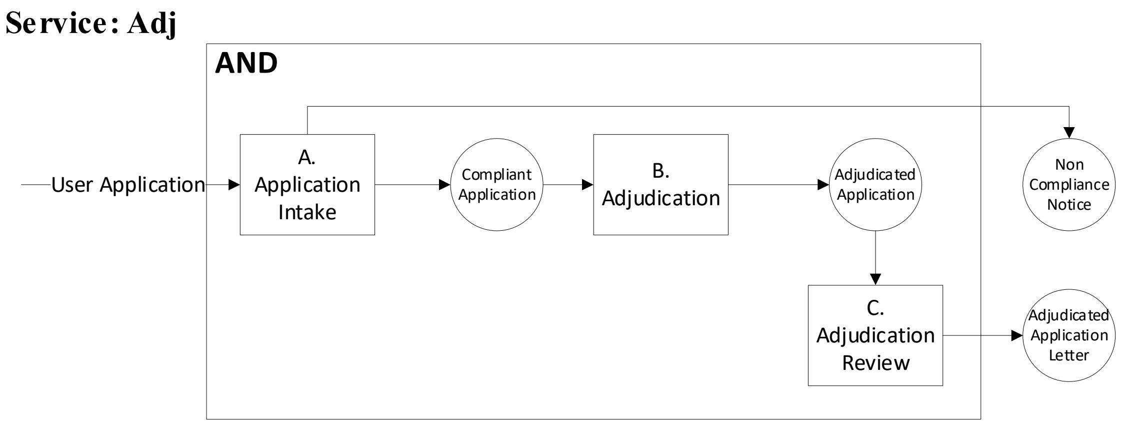


Figure 17 - Generalized BSN (Root Service)

Generalized, composite service A is comprised of subservices AA, AB and AC and its type is OR because only one subservice can be present, that is, A can have AA as a configuration, but not “AA, AB”.

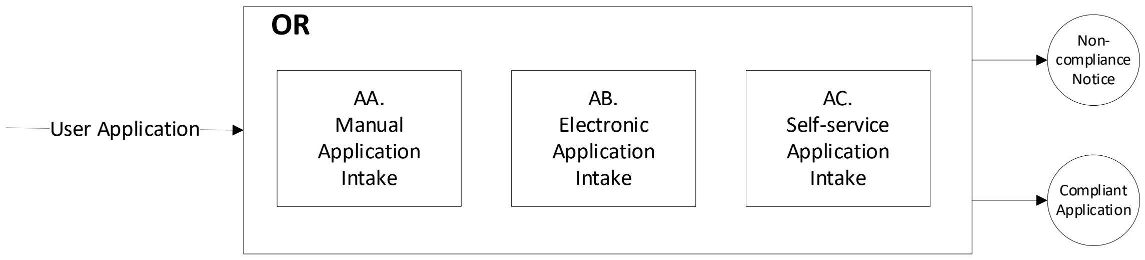

OR
User Application
AA. Manual Application Intake
AB. Electronic Application Intake
AC. Self-service Application Intake
Non-compliance Notice
Compliant Application

Figure 18 - Generalized Service A

By the same token, B is comprised of subservices BA and BB and its type is OR.

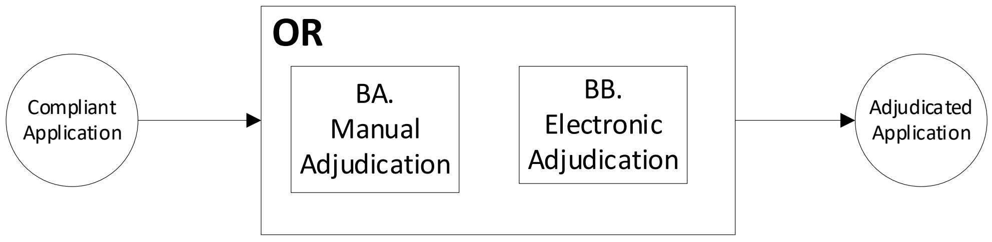


Figure 19 - Generalized Service B

C is comprised of subservices CA and CB and its type is also OR.

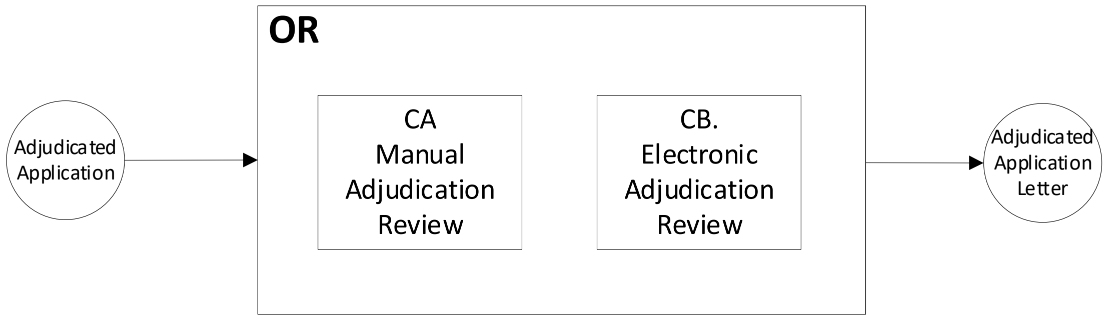

OR
Adjudicated Application
CA Manual Adjudication Review
CB. Electronic Adjudication Review
Adjudicated Application Letter

Figure 20 - Generalized Service C

In step 3, we identify and codify all the parameters for each of the components of the formal model in Figure 5, which we reproduce below:

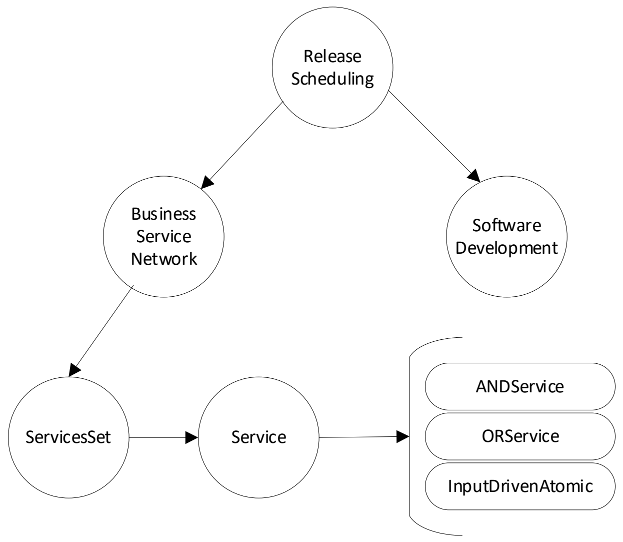


Figure 21 - Hierarchy of the Components of the Formal Model

6.1 Parameters

6.1.1 Parameters - Release Schedule

Below is the codification of the candidate features, the dependency graph and the other remaining parameters for the Release Scheduling Component, like investment period, discount rate, number of releases, release duration, resources used by features. Note that IBF (Implemented Business Features) and ITF (Implemented Business Features) are vectors of decision variables and are initialized with `"integer?"` to denote to Unity that they are decision variables of type integer. The `null` indicates that initially they don't have any value, but rather, the solver instantiates them. Each vector has 5 entries, one for each of the 4 releases and one for the period after the last release.

Table 16 - Release Scheduling Parameters

BF=	{BF1, BF2, BF3, BF4}		
TF=	{TF1}		
DG=	{(TF1,BF1), (BF1,BF2), (TF1,BF3), (BF1, BF4)}		
FS=	TF1	140	
	BF1	140	
	BF2	280	
	BF3	280	
	BF4	280	
TH=	520		
DiscountRate=	0.0192%		
NR=	4		
RD=	60		
ResSet=	{softwareLicense1}		
ResCost=	SoftwareLicense1	20,000	
FeatureRes	TF1	{}	
	BF1	{}	
	BF2	{}	
	BF3	{}	
	BF4	{softwareLicense1}	
IBF=	1	`integer?`	`null`
	2	`integer?`	`null`
	3	`integer?`	`null`
	4	`integer?`	`null`
	5	`integer?`	`null`
ITF=	1	`integer?`	`null`
	2	`integer?`	`null`
	3	`integer?`	`null`
	4	`integer?`	`null`
	5	`integer?`	`null`

6.1.2 Parameters - Business Service Network

The BSN parameters shown in the table below, determine the labor rates, payment schedule, the inputs and outputs of the BSN, the throughput demand required by the BSN, the set of services and the id of the root service. Note that the BSN's demand, i.e., the number of input flows that it must process per day is 100. This value has a profound effect on the instantiation of the decision variables.

Table 17 - Business Service Network Parameters

LR=	{Intake Officer, Adj Officer, Applicant, System}	
Rate=	Intake Officer	$160
	Adj Officer	$400
	Applicant	$0
	System	$0
NLP=	5	
LaborPayDay=	60,120,180,240,520	
BSNI=	{User Application}	
BSNO=	{}	
Demand=	User Application	100
ServicesSet=	{Adj, A, B, C, AA, AB, AC, BA, BB, CA, CB}	
rootID=	Adj	

Note that although the labor cost of the service network accrues each day, we only pay on the days of the *LaborPayDay* schedule above. The schedule in the example has only 5 payments to simplify the calculations, but the model accepts any number of payments.

6.1.3 Parameters – ANDService and ORService

Below are the parameters for the composite services of type AND and OR, showing their type, I/Os and subservices.

Table 18 – Composite Services Parameters

id	ServiceType	Input/Output		Subservices
Adj	ANDservice	User Application	Adjudicated Application Letter	{A, B, C}
		User Application	Non-compliance Notice	
A	ORservice	User Application	Compliant Application	{AA, AB, AC}
		User Application	Non-compliance Notice	
B	ORservice	Compliant Application	Adjudicated Application	{BA, BB}
C	ORservice	Adjudicated Application	Adjudicated Application Letter	{CA, CB}

6.1.4 Parameters – InputDrivenAtomicService

Below are the parameters for the atomic services, like AA. Table 19 shows the business features required by each service and the fixed cost per day for the service.

Table 19 - InputDrivenAtomicService Parameters RequiredFeatures and Cost per Day

id	RBF	ServiceCostPerDay
AA		200.00
AB	{BF1}	200.00
AC	{BF1, BF4}	200.00
BA		200.00
BB	{BF1, BF2}	200.00
CA		200.00
CB	{BF3}	200.00

Table 20 below shows the inputs, outputs, and the ratio of input to output. The ratio is important so that the Solver can calculate the correct number of flows given the demand on the BSN.

Table 20 - InputDrivenAtomicService Parameter *IOThruRatio*

id	ServiceType	Input/Output/IOThruRatio		
AA	InputDrivenAtomic	User Application	Compliant Application	70%
		User Application	Non-compliance Notice	30%
AB	InputDrivenAtomic	User Application	Compliant Application	70%
		User Application	Non-compliance Notice	30%
AC	InputDrivenAtomic	User Application	Compliant Application	70%
		User Application	Non-compliance Notice	30%
BA	InputDrivenAtomic	Compliant Application	Adjudicated Application	100%
BB	InputDrivenAtomic	Compliant Application	Adjudicated Application	100%
CA	InputDrivenAtomic	Adjudicated Application	Adjudicated Application Letter	100%
CB	InputDrivenAtomic	Adjudicated Application	Adjudicated Application Letter	100%

Table 21 below shows the time in hours to process each input and produce the corresponding output and identifies the role of the worker.

Table 21 - InputDrivenAtomicService Parameter *RoleTimePerIO*

id	Role	I	O	RoleTime PerIO
AA	Intake Officer	User Application		0.250
AA	Intake Officer		Compliant Application	0.125
AA	Intake Officer		Non-compliance Notice	0.219
AB	Intake Officer	User Application		0.145
AB	System		Compliant Application	0.000
AB	System		Non-compliance Notice	0.000
AC	Applicant	User Application		0.063
AC	System		Compliant Application	0.000
AC	System		Non-compliance Notice	0.000
BA	Adj Officer	Compliant Application		0.042
BA	Adj Officer		Adjudicated Application	0.208
BB	Adj Officer	Compliant Application		0.021
BB	Adj Officer		Adjudicated Application	0.143

id	Role	I	O	RoleTime PerIO
CA	Adj Officer	Adjudicated Application		0.021
CA	Adj Officer		Adjudicated Letter	0.167
CB	Adj Officer	Adjudicated Application		0.017
CB	Adj Officer		Adjudicated Letter	0.083

Table 22 below shows the fixed cost per input for service AA, which is 2.00, which is in addition to the labor cost required to process the inputs. The other services don't have this cost.

Table 22 - InputDrivenAtomicService Parameter *CostPerInput*

id	Input	CostPerInput
AA	User Application	2.00
AB	User Application	
AC	User Application	
BA	Compliant Application	
BB	Compliant Application	
CA	Adjudicated Application	
CB	Adjudicated Application	

Table 23 below shows the fixed cost per output for service AA. The other services don't have this cost.

Table 23 - InputDrivenAtomicService Parameter *CostPerOutput*

id	Output	CostPerOutput
AA	Compliant Application	3.00
AA	Non-compliance Notice	1.00
AB	Compliant Application	
AB	Non-compliance Notice	
AC	Compliant Application	
AC	Non-compliance Notice	
BA	Adjudicated Application	
BB	Adjudicated Application	
CA	Adjudicated Application Letter	
CB	Adjudicated Application Letter	

Table 24 below shows that each atomic service has a decision variable *On(s,r),* where *s* is the service id and *r* is the release number, that indicates whether the service is activated or not in each release. The type of O*n* is integer but additional constraints in the DGS forces it to be binary.

Table 24 - Decision Variable *On(r)* where s is the service id, and *r* is the release number

Service Id	On(1)	On(2)	On(3)	On(4)	On(5)
AA	Integer?	Integer?	Integer?	Integer?	Integer?
AB	Integer?	Integer?	Integer?	Integer?	Integer?
AC	Integer?	Integer?	Integer?	Integer?	Integer?
BA	Integer?	Integer?	Integer?	Integer?	Integer?
BB	Integer?	Integer?	Integer?	Integer?	Integer?
CA	Integer?	Integer?	Integer?	Integer?	Integer?
CB	Integer?	Integer?	Integer?	Integer?	Integer?

Table 25 below shows that each atomic service has a decision variable *InputThru(s,i,r),* where *s* is the service id, *i* is the input and *r* is the release number, that captures the throughput of the services, i.e., the number of inputs that flow through them. The *InputThru(i,r)* value is chosen by the MILP Solver and is constrained by *IOThruRatio* in Table 20 and *Demand* in Table 17.

Table 25 - Decision Variable *InputThru(s,i,r),* where s is the service id, *i* is the input and *r* is the release number

Svc Id	Input	InputThru Type	InputThru Rel 1	InputThru Rel 2	InputThru Rel 3	InputThru Rel 4	InputThru after Rel 4
AA	User Application	decimal?	null	null	null	null	null
AB	User Application	decimal?	null	null	null	null	null
AC	User Application	decimal?	null	null	null	null	null
BA	Compliant Application	decimal?	null	null	null	null	null
BB	Compliant Application	decimal?	null	null	null	null	null
CA	Adjudicated Application	decimal?	null	null	null	null	null
CB	Adjudicated Application	decimal?	null	null	null	null	null

The labor rates in Table 17, the ratio of input output in Table 20, the time per input in Table 21 and the throughput in Table 25 are used to compute the labor cost per day for each service.

The service cost per day in Table 19, the cost per input in Table 22, and the cost per output in Table 23 are used to calculate the fixed cost for each service.

6.1.5 Parameters – Software Development

Table 16 above shows parameters that impact the software releases such as the number of releases and their duration. Table 26 below shows the remaining parameters for the development of the software such as the team size, developer productivity in points per day, development cost/point/day, operations cost/point/day, size of the initial system in points and payment schedule.

Table 26 - Software Development Parameters

TS=	5
DP=	1
DC=	$1,040
OC=	$0.25
SS=	0
NSP=	5
SWPayDay=	60,120,180,240,520

6.2 Recommendation of the To-Be BSN

Step 4 of the methodology is covered in section 6.4. In step 5, we run the DGS with the parameters in the previous section in order to produce a recommendation, that is, the Solver instantiates the decision variables.

Decision Variables - Release Scheduling

Below are the instantiations of the decision variables IBF and ITF, which assign features to releases, and consequently make up the release plan. It shows, for example, that in release 1, both TF1 and BF1 are developed.

Table 27 - Release Scheduling Decision Variables

	Rel 1	Rel 2	Rel 3	Rel 4
TF1	1			
BF1	1			
BF2			1	
BF3		1		
BF4				1

<u>Decision Variables - Services</u>

The table below contains the instantiations for the Decision Variable *On*.

Table 28 - Service Decision Variable *On(s,r),* where s is the service id and *r* is the release number

Service	Rel 1	Rel 2	Rel 3	Rel 4	after Rel 4
Adj	1	1	1	1	1
A	1	1	1	1	1
B	1	1	1	1	1
C	1	1	1	1	1
AA	1	0	0	0	0
AB	0	1	1	1	0
AC	0	0	0	0	1
BA	1	1	1	0	0
BB	0	0	0	1	1
CA	1	1	0	0	0
CB	0	0	1	1	1

The above Table 28 determines the instantiation of each service in each release, for example, AA is instantiated in release 1 but not in any other release. Table 29 below translates those instantiations into BSN configurations. Row 1 shows that the initial BSN, prior to the software being developed, is AA, BA, CA and it matches the As-Is configuration in Figure 16. Per Table 27 , in release 1 features TF1 and BF1 are developed and according to Table 19, they allow service AB to be activated, consequently after release 1 the configuration is AB, BA, CA.

Table 29 - Example of the Configurations of the SN

Period	Configuration
Before release 1	AA, BA, CA
After Release 1	AB, BA, CA
After Release 2	AB, BA, CB
After Release 3	AB, BB, CB
After Release 4	AC, BB, CB

After release 4, the final To-Be BSN configuration, depicted in Figure 22 below, is AC, BB, CB.

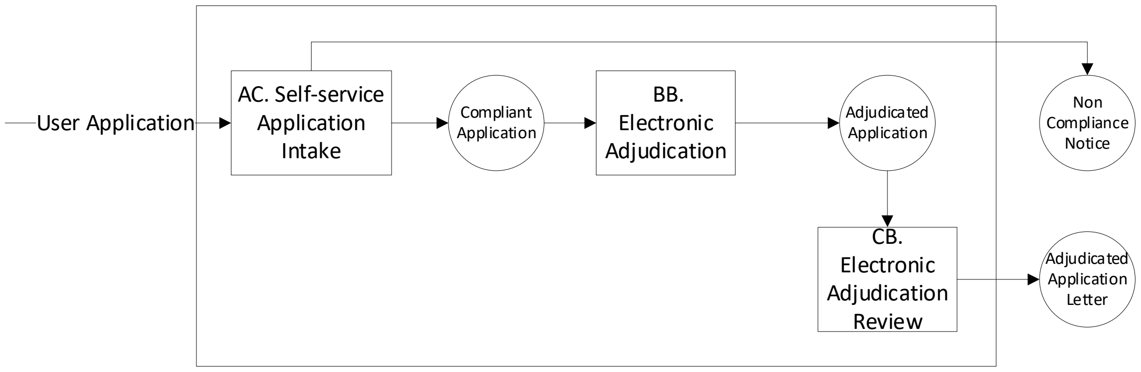

User Application
AC. Self-service Application Intake
Compliant Application
BB. Electronic Adjudication
Adjudicated Application
Non Compliance Notice
CB. Electronic Adjudication Review
Adjudicated Application Letter

Figure 22 - To-Be BSN

The table below contains the instantiations for the Decision Variable *InputThru*, which have a direct impact in the calculation of the labor cost of each service.

Table 30 - Service Decision Variable *InputThru(s, i,r)*, where s is the service id, *i* is the input and *r* is the release number

id	Input	R=1	R=2	R=3	R=4	R=4+
AA	User Application	100	0	0	0	0
AB	User Application	0	100	100	100	0
AC	User Application	0	0	0	0	100
BA	Compliant Application	70	70	70	0	0
BB	Compliant Application	0	0	0	70	70
CA	Adjudicated Application	70	70	0	0	0
CB	Adjudicated Application	0	0	70	70	70

6.3 Computation of the To-Be BSN

Once the decision values are instantiated, the next step for the DGS is to perform the computations according to the formal model in section 4. The computations are performed bottom up according to the hierarchy in Figure 21. First, the software computations are performed, then the atomic component, then the OR, then the AND, then the BSN and finally the Release Schedule.

6.3.1 Computation – Software Development

First the software computations are performed. The release size in points is calculated, as it drives the costs. The final calculation is the software *CashFlow* per day, which is the sum of *LaborCashFlow* and the cashflow spent by the resources, that is, the features. *CashFlow* is the interface metrics for the SoftwareDevelopment component, consequently it is rolled up the hierarchy to ReleaseScheduling.

Table 31 - Software Development Computations

	RC:	300	RS:	300	
	R1	**R2**	**R3**	**R4**	**After R4**
ImplementedFeatures=	TF1, BF1	BF3	BF2	BF4	
Service Network=	AA, BA, CA	AB, BA, CA	AB, BA, CB	AB, BB, CB	AC, BB, CB
relRes=				softwareLicense1	
First Day=	1	61	121	181	241
Last Day=	60	120	180	240	520
devCostPerDay=	$5,200	$5,200	$5,200	$5,200	$0
opsCostPerDay=	$0	$75	$150	$225	$300
SWCostForDay	$5,200	$5,275	$5,350	$5,425	$300
CostPeriod=	[1..60]	[61..120]	[121..180]	[181..240]	[241..520]
SWPayDay=	60	120	180	240	520
FeatureRes=				{softwareLicense1}	
ResCost=				$20,000	
SWLaborPayment=	$312,000	$316,500	$321,000	$325,500	$84,000
LaborCashFlow(last day Rel)=	-$312,000	-$316,500	-$321,000	-$325,500	-$84,000
ResCashFlow:(lastday Rel)=	$0	$0	-$20,000		$0
CashFlow(last day Rel)=	-$312,000	-$316,500	-$341,000	-$325,500	-$84,000
CashFlow(other days)=	0	0	0	0	0

6.3.2 Computation – Atomic Service

Second, the atomic services are computed. The first calculation is of the *OutputThru*. Then, *TimePerDay* is computed by multiplying *RoleTimePerIO* by *InputThru*, and adding the multiplication of *RoleTimePerIO* times *OutputThru*. The result is shown in Table 32.

Table 32 - InputDrivenAtomicService Computations - *TimePerDay*

		TimePerDay					
id	**Role**	**R=1**	**R=2**	**R=3**	**R=4**	**R=4+**	**Rate**
AA	Intake Officer	**R=1**	**R=2**	**R=3**	**R=4**	**R=4+**	**Rate**
AB	Intake Officer	40.32	0.00	0.00	0.00	0.00	160.00
AB	System	0.00	14.50	14.50	14.50	0.00	160.00
AB	System	0.00	0.00	0.00	0.00	0.00	0.00
AC	Applicant	0.00	0.00	0.00	0.00	0.00	0.00
AC	System	0.00	0.00	0.00	0.00	6.30	0.00
BA	Adj Officer	0.00	0.00	0.00	0.00	0.00	0.00
BB	Adj Officer	17.50	17.50	17.50	0.00	0.00	400.00
CA	Adj Officer	0.00	0.00	0.00	11.48	11.48	400.00
CB	Adj Officer	13.16	13.16	0.00	0.00	0.00	400.00

Once *TimePerDay* is known, *LaborCostPerDay* is calculated by multiplying *TimePerDay* by the labor rate. Next, *FlowCostPerDay* is calculated by multiplying *CostPerInput* by *InputThru* and adding to *CostPerOutput* times *OutputThru*. The interface metric *CostPerDay* is the sum of *LaborCostPerDay* plus *FlowCostPerDay* plus parameter *ServiceCostPerDay* from Table 19.

Table 33 - InputDrivenAtomicService Computation - *CostPerDay*

id	R=1	R=2	R=3	R=4	R=4+
AA	$7,091.20	$0.00	$0.00	$0.00	$0.00
AB	$0.00	$2,520.00	$2,520.00	$2,520.00	$0.00
AC	$0.00	$0.00	$0.00	$0.00	$200.00
BA	$7,200.00	$7,200.00	$7,200.00	$0.00	$0.00
BB	$0.00	$0.00	$0.00	$4,792.00	$4,792.00
CA	$5,464.00	$5,464.00	$0.00	$0.00	$0.00
CB	$0.00	$0.00	$3,000.00	$3,000.00	$3,000.00

6.3.3 Computation – OR Services

The *CostPerDay* of an OR service, shown below, is the sum of the *CostPerDay* of the subservices in Table 33.

Table 34 - ORservice Computation - *CostPerDay*

id	R=1	R=2	R=3	R=4	R=4+
A	$7,091.20	$2,520.00	$2,520.00	$2,520.00	$200.00
B	$7,200.00	$7,200.00	$7,200.00	$4,792.00	$4,792.00
C	$5,464.00	$5,464.00	$3,000.00	$3,000.00	$3,000.00

6.3.4 Computation – AND Service

The *CostPerDay* of an AND service, shown below, is the sum of the *CostPerDay* of the subservices in Table 34.

Table 35 - ANDservice Computation - *CostPerDay*

id	R=1	R=2	R=3	R=4	R=4+
Adj	$19,755.20	$15,184.00	$12,720.00	$10,312.00	$7,992.00

6.3.5 Computation - Business Service Network

The *BSNCostForDay* is the *CostPerDay* of the root service Adj, shown in Table 35. *BSNPayment* is the number of days in the period, times the *BSNCostPerDay*, and it is paid according to the payment schedule. In the example, payments are made on the last day of the period. The total cashflow of the BSN during the 60 days in release 1 is -$1,185,312.00 while during the same 60 days in release 4 is -$618,720.00. The large savings in release 4 is due to the utilization of the BSN configuration AB, BB, CB, which is much less costly than AA, BA, CA, because there is less time spent processing inputs due to some tasks being partially or fully automated by the implemented system.

Table 36 - BusinessServiceNetwork Computation - *CashFlow*

	Rel 1	Rel 2	Rel 3	Rel 4	After Rel 4
First Day=	1	61	121	181	241
Last Day=	60	120	180	240	520
BSNCostForDay	$19,755.20	$15,184.00	$12,720.00	$10,312.00	$7,992.00
CostPeriod=	[1..60]	[61..120]	[121..180]	[181..240]	[241..520]
LaborPayDay=	60	120	180	240	520
BSNPayment=	$1,185,312.00	$911,040.00	$763,200.00	$618,720.00	$2,237,760.00
CashFlow(last day Rel)=	-$1,185,312.00	-$911,040.00	-$763,200.00	-$618,720.00	-$2,237,760.00
CashFlow(other days)=	0	0	0	0	0

6.3.6 Computation - Release Schedule

In the ReleaseScheduling module, the cashflow of the BSN from Table 36 is combined with the cash flow of the software from Table 31. They are then discounted based on the payment schedule, resulting in *TimeWindowNPV*, which is also the objective function. Table 37 below shows that the NPV of the combined BSN and software development for the entire investment period of 520 days is -6,748,777.45.

Table 37 - ReleaseScheduling Computation - *CashFlow*

d=	60	120	180	240	520
BSN.IM.CashFlow=	-1,185,312.00	-911,040.00	-763,200.00	-618,720.00	-2,237,760.00
SWD.IM.CashFlow=	-312,000.00	-316,500.00	-341,000.00	-325,500.00	-84,000.00
Combined Cash Flow=	-1,497,312.00	-1,227,540.00	-1,104,200.00	-944,220.00	-2,321,760.00
Discounted CF=					
TimeWindowNPV=	-1,480,136.25	-2,679,675.43	-3,746,306.93	-4,647,941.93	-6,748,777.45

6.4 Computation of the As-Is NPV

In step 4 of the methodology, we compute the NPV of the As-Is. To do that we set the parameters as follows:

1. The *IBF* and *ITF* decision variables are set to 0 for all releases
2. The *On* decision variables are set to indicate that the As-Is atomic services (AA, AB, CA) are activated for all releases while all the other atomic services are not activated at all.

6.4.1 Software Development Computations

Because the *IBF* and *ITF* are set to 0, there is nothing to be developed consequently the cost of software is zero.

6.4.2 BSN Computations

The cost of the BSN is calculated, and the results are shown below.

Table 38 - As-Is BSN *CashFlow*

	R1	R2	R2	R4	After R4
First Day=	1	61	121	181	241
Last Day=	60	120	180	240	520
BSNCostForDay	$19,755.20	$19,755.20	$19,755.20	$19,755.20	$19,755.20
CostPeriod=	[1..60]	[61..120]	[121..180]	[181..240]	[241..520]
LaborPayDay=	60	120	180	240	520
BSNPayment=	$1,185,312.00	$1,185,312.00	$1,185,312.00	$1,185,312.00	$5,531,456.00
CashFlow (last day Rel)=	-$1,185,312.00	-$1,185,312.00	-$1,185,312.00	-$1,185,312.00	-$5,531,456.00
CashFlow (other days)=	0	0	0	0	0

6.4.3 Release Schedule Computations

The aggregated cost of the As-Is is the same as the cost of the BSN in Table 38 because the cost of software is zero. The total NPV of the As-Is is, shown in Table 39 below, is -9,611,947.49.

Table 39 - As-Is NPV

d=	60	120	180	240	520
BSN.IM.CashFlow=	-1,185,312.00	-1,185,312.00	-1,185,312.00	-1,185,312.00	-5,531,456.00
SWD.IM.CashFlow=	0.00	0.00	0.00	0.00	0.00
Combined Cash Flow=	-1,185,312.00	-1,185,312.00	-1,185,312.00	-1,185,312.00	-5,531,456.00
Discounted CF=	-1,171,715.22	-1,158,274.42	-1,144,987.79	-1,131,853.58	-5,005,116.48
TimeWindowNPV=	-1,171,715.22	-2,329,989.64	-3,474,977.43	-4,606,831.01	-9,611,947.49

6.5 Total Benefit

Now we can calculate the total benefit of the implementation of the software, by subtracting the NPV of the As-Is from the NPV of the To-Be as shown in the equation below:

$$Total\ Benefit = NPV(To - Be) - NPV(As - Is) =$$

$$-6{,}748{,}777.45 - (-9{,}611{,}947.49) = 2{,}863{,}170.04$$

We conclude that the release plan [{TF1, BF1}, BF3, BF2, BF4] saves 2,863,170.04 over the time horizon of 520 days. Note that this is the maximum savings, i.e., there is no other release plan and BSN configuration that produces a higher savings.

6.6 Constraint Computation and Validation for the To-Be

In the previous sections we showed the calculations of the total NPV for the To-Be and the As-Is. We now show the constraints for the To-Be for the same example. All constraints must evaluate to true in order for the recommendation described in section 0 to be feasible.

6.6.1 Constraint Computation and Validation – Software Development

Constraint: *ReleaseSizeCannotExceedCapacity*

The calculation of Release Size (RS) results in 280 points and that is also the value of the Release Capacity (RC), consequently RS≤RC holds for every release.

6.6.2 Constraint Computation and Validation – Atomic Service

Constraint: *FeatureDependencyIsSatisfied*

The constraint states that if the value of the *On* decision variable is 1 for service *s* and release *r*, then the required business feature (RBF) for service *s* must be implemented in a prior release, that is, must be a subset of *SoFarIBF(r-1),* which is the set of implemented business features up to *r-1*. Table 28 shows the value of *On*. Take service AB, for example, which is active in release 2, that is, *On(AB,2)=1*. According to the *RBF* column in Table 19, AB requires business feature BF1, i.e., *RBF(AB)={BF1}*. The release schedule in Table 27 shows that BF1 is implemented in release 1, consequently *SoFarIBF(2-1) = SoFarIBF(1) = {BF1}*. The constraint below:

$$On(AB,2) = 1 \rightarrow RBF(AB) \subseteq SoFarIBF(1)$$

holds because *RBF(AB)={BF1}* is a subset of *SoFarIBF(1)={BF1}*. Following the same logic, the constraint also holds for AB in releases 2 to 4 and holds for the other atomic services as well.

Constraint: *DeactivedServicesIsSatisfied*

The constraint, shown below, says that if a service is deactivated, then no input can flow through it.

$$On(id,r) = 0 \rightarrow InputThru(id,i,r) = 0$$

Table 28 shows the value of *On*. Take service BB, for example, which is deactivated in release 1, that is, *On(BB,1)=0*. Table 30 shows that *InputThru(BB,Compliant Application,1)=0*, which satisfies this constraint.

6.6.3 Constraint Computation and Validation – ORservice

Constraint: *OnlyOneServiceActivated*

The constraint, reproduced below, states that the value of *On(o,r)* for ORservice *o* is equal to the sum of *On(s,r)* for all its subservices *s*. If *On(o,r)=1*, then one subservice must have *On(s,r)=1* while all others must be zero. If *On(o,r)=0*, then all subservices must have *On(s,r)=0*

$$\sum_{i \in Subservices(id)} On(i,r) = On(id,r), \qquad \forall\, r \in [1..NR+1]$$

Take service B, for example, which is activated in release 1, that is, *On(B,1)=1*. Table 28 shows that *On(BA,1)=1* and *On(BB,1)=0* consequently *On(BA,1)+On(BB,1)=1=On(B,1)* and the constraint holds.

Constraint: *TotalSupplyMatchesTotalDemand*

This is a very important constraint because it balances the BSN, guaranteeing that the number of inputs matches the number of outputs at every level. This constraint, together with the BSN constraints in the next section, satisfy the required BSN throughput demand while at the same time guaranteeing that the number of input and output flows through the BSN are correct. Take ORservice B, for example, the constraint is reproduced below.

$$TotalSupply(B,j,r) = TotalDemand(B,j,r)$$

Where:

$$TotalSupply(B,j,r) = \begin{cases} InputThru(B,j,r) + InternalSupply(B,j,r) & if\ j \in I(B) \\ InternalSupply(B,j,r) & otherwise \end{cases}$$

$$TotalDemand(B,j,r) = \begin{cases} OutputThru(B,j,r) + InternalDemand(B,j,r) & if\ j \in O(B) \\ InternalDemand(B,j,r) & otherwise \end{cases}$$

$$InputThru(B,i,r) = InternalDemand(B,i,r) - InternalSupply(B,i,r)\ \forall i \in I(id)$$

$$OutputThru(B,o,r) = InternalSupply(B,o,r) - InternalDemand(B,o,r)\ \forall\ o \in O(id)$$

$$InternalDemand(B,j,r) = \begin{cases} \sum_{s \in Subservices(B)} InputThru(s,j,r) & if\ j \in I(s) \\ 0 & otherwise \end{cases}$$

$$InternalSupply(B,j,r) = \begin{cases} \sum_{s \in Subservices(B)} OutputThru(s,j,r) & if\ j \in O(s) \\ 0 & otherwise \end{cases}$$

Using *InputThru* instantiation from Table 30, here are the computations for input flow 'Compliant Application', or 'CA' in release 1.

InternalDemand(B,CA,1)=70+0=70

InternalSupply(B,CA,1)=0

InputThru(B,CA,1)=70-0=70

OutputThru(B,CA,1)=0

TotalSupply(B,CA,1)=70+0=70

TotalDemand(B,CA,1)=70

The computations show that *TotalSupply(B,CA,1)=70=TotalDemand(B,CA,1)* consequently the constraint holds.

6.6.4 Constraint Computation and Validation – ANDservice

Constraint: *AllSubservicesAreActivated*

The constraint, reproduced below, states that the value of *On(a,r)* for ANDservice *a* is equal to the sum of *On(s,r)* for all its subservices *s*, multiplied by the number of subservices. If *On(a,r)=1*, then all subservices must have *On(s,r)=1*, and if *On(a,r)=0*, then all subservices must have *On(s,r)=0*.

$$\sum_{i \in Subservices(id)} On(i,r) = nOn(id,r) \ where \ n = \text{cardinality of } Subservices(id)$$

Take service Adj, for example, which has 3 subservices, A, B and C, and is activated in release 1, that is, *On(Adj,1)=1*. Table 28 shows that *On(A,1)=1*, *On(B,1)=1* and *On(C,1)=1* consequently *On(A,1)+On(B,1)+On(C,1)=3=3(On(A,1))=3(1)=3* and the constraint holds.

Constraint: *TotalSupplyMatchesTotalDemand*

Same as in ORservice.

Constraint: *RootMustBeActivated*

The constraint, reproduced below states that the *On(rootID,r) for* ANDservice *rootID* has to be 1 for every release. This guarantees that the BSN top level service is always activated, which has a downstream effect to its subservices and their subservices, that is.

$$On(rootID, r) = 1$$

6.6.5 Constraint Computation and Validation – BusinessServiceNetwork

Constraint: *BSNDemandIsSatisfied*

For root service Adj in release 1 and Demand("User Application")=100 as shown in Table 17, the constraint is:

$$InputThru(Adj, User\ Application, 1) \geq 100$$

Using the *InputThru* instantiations in Table 30 and the equations from section 6.3.3Constraint Computation and Validation – ORservice, we compute:

InternalDemand(Adj,User Application,1)=100+0+0=100

InputThru(Adj,User Application,1)=100

As *InputThru=100* and 100≥100, the constraint holds.

Constraint: *BSNSupplyIsSatisfied*

In the example, the BSN does not have a demand for output flows, that is, BSNO is the empty set consequently the constraint does not apply.

6.6.6 Constraint Computation and Validation – ReleaseScheduling

Constraint: *FeatureSetsForReleasesArePairwiseDisjoint*

The constraint is reproduced below. It states that feature sets are pairwise disjoint across releases, that is, a feature can only be assigned to one release.

$$(IBF(i) \cup ITF(i)) \cap (IBF(j) \cup ITF(j)) = \emptyset \quad (\forall\, i, j, \in [1..NR], i \neq j)$$

To calculate the constraint, we use the values in Table 27, Release Schedule, which we reproduce below:

Table 40 - Release Schedule for the Example

	Rel 1	Rel 2	Rel 3	Rel 4
TF1	1			
BF1	1			
BF2			1	
BF3		1		
BF4				1

Below we calculate the constraint, starting with i=1 and j=2.

$\{TF1, BF1\} \cap \{BF3\} = \emptyset$ for *i=1, j=2*

$\{TF1, BF1\} \cap \{BF2\} = \emptyset$ for *i=1, j=3*

$\{TF1, BF1\} \cap \{BF4\} = \emptyset$ for *i=1, j=4*

$\{BF3\} \cap \{BF2\} = \emptyset$ for *i=2, j=3*

$\{BF3\} \cap \{BF4\} = \emptyset$ for *i=2, j=4*

$\{BF2\} \cap \{BF4\} = \emptyset$ for *i=3, j=4*

The values above show that the constraint holds for all cases.

Constraint: *FeatureDependenciesAreSatisfied*

The constraint is reproduced below. It states that if a feature f_1 precedes f_2 and f_2 is scheduled in release r, then f_1 must be scheduled in release r, or in a prior release.

$$(f_1 \prec f_2 \wedge f_2 \in IBF(r) \cup ITF(r)) \rightarrow (f_1 \in \bigcup_{i=1}^{r} IBF(r) \cup ITF(r))$$
$$(\forall r \in [1..NR])(\forall f_1, f_2 \in BF \cup TF),$$

where $f_1 \prec f_2$ are elements of the Dependency Graph from Table 16, which we reproduce here: DG={(TF1,BF1), (BF1,BF2), (TF1,BF3), (BF1, BF4)}

Below we compute the constraint, using the release schedule from Table 40. and the Dependency Graph:

Table 41 - Constraint: FeatureDependenciesAreSatisfied

r	$(f_1 \prec f_2 \wedge f_2 \in IBF(r) \cup ITF(r))$	$\rightarrow (f_1 \in \bigcup_{i=1}^{r} IBF(i) \cup ITF(i))$	**Result**
1	$TF1 \prec BF1 \wedge BF1 \in IBF(1)$	$TF1 \in IBF(1)$	TRUE
2	$TF1 \prec BF3 \wedge BF3 \in IBF(1)$	$TF1 \in IBF(1)$	TRUE
3	$BF1 \prec BF2 \wedge BF2 \in IBF(2)$	$BF1 \in IBF(1)$	TRUE
4	$BF1 \prec BF4 \wedge BF4 \in IBF(4)$	$BF1 \in IBF(1)$	TRUE

The computations show that the release schedule satisfies the constraint in all cases.

7. SENSITIVITY ANALYSIS

Because businesses are dynamic and changes happen all the time, stakeholders need to know the sensitivity of the recommendation to the main driver of cost, which is the demand on business process throughput. Knowing the sensitivity has the potential to increase the confidence in the recommendation and also help stakeholders make more informed decisions. To answer this, we developed a technique for sensitivity analysis as follows.

The objective function is the NPV of the cash outflow of the service network (SN) plus the cash outflow of developing the software features that allow the SN to transition to more efficient processes. *Opti-Soft+* has several parameters that influence the NPV, but the one with the most impact is the demand, which is the required throughput of the SN. In our example, the required demand is 100 applications per day.

The required demand, used as a parameter in the DGS, is an estimation and if there is a high degree of uncertainty in the estimation, a decision maker might not have a lot of confidence in the recommendation. That's why a sensitivity analysis based on the demand parameter is so valuable because it helps to understand risk.

In our sensitivity analysis technique, we use the NPC instead of the NPV because it is more intuitive. The goal is to determine the NPC delta, that is, the additional cost for an increase of one unit of demand. Given d_0, the original demand through the SN, we vary d, the new demand by 1. The delta of the demand is $\delta = d - d_0$. We then calculate UC, the cost per unit of demand d as follows:

$$UC(\delta) = \frac{NPC(d_0 + \delta)}{d_0 + \delta}$$

We can utilize the above technique to conduct two analyses for a range of δ: 1) fix the release plan and the BSN configuration, and 2) fix the release plan, allowing the BSN configuration to be optimized. The first analysis shows how the unit cost varies with each δ, while the second shows the unit cost variation and the stability of the BSN configuration.

7.1 <u>Sensitivity Analysis 1</u>

The steps to conduct analysis number 1 are as follows: 1) determine a range of δ, above and below d_0, to conduct the analysis, 2) run the DGS optimization with *demand=*d_0 to get a recommendation and the value of NPV_0, 3) instantiate the *ITF(r,f)*, *IBF(r,f)* and *On(s,r)* decision variables with the release planning schedule and SN configuration recommended by the DGS in the previous step, leaving *InputThru(s,i,r)* as a DV, 4) set the *demand* parameter to $d_0+\delta_1$, where δ_1, is the first value in the δ range, and run the DGS to get the value for NPC_1, 5) repeat steps 2-4 (i.e., now performing operational optimization when software features available are fixed) for all δ_i in the range, $i > 1$, 6) calculate the values of UC(δ_i), and 7) plot a chart with the values of δ_i and UC(δ_i).

We now apply our sensitivity analysis technique to the example in Section 4. In step 1, we determine that the estimated *demand* $d_0=100$ has an error or 10%, so we set the range of δ to -10 to +10. In step 2 we run the DGS with *demand=100* and produce the recommendation and *NPC*$_0$*=\$6,748,777.45*, described in Section 4. In step 3 we instantiate the release planning schedule and SN configuration DVs with the recommendation in

Section 4. In step 4, we take the first value in the δ range (-10) and set *demand=100-10=90* and run the DGS, getting $NPC_1=\$6,236,485.38$. In step 5, we repeat steps 2-4 for all the other values in the δ range and produce the NPC results in Table 42. In step 6, we calculate UC(δ_i), also shown in Table 42. In step 6 we plot the chart shown in Figure 23.

Table 42 - Results of the Sensitivity Analysis

d	δ	NPC(d_0+ δ)	UC(δ)
90	-10	$6,236,485.38	$69,294.28
91	-9	$6,287,714.60	$69,095.76
92	-8	$6,338,943.82	$68,901.56
93	-7	$6,390,173.05	$68,711.54
94	-6	$6,441,402.27	$68,525.56
95	-5	$6,492,631.49	$68,343.49
96	-4	$6,543,860.71	$68,165.22
97	-3	$6,595,089.93	$67,990.62
98	-2	$6,646,319.15	$67,819.58
99	-1	$6,697,548.37	$67,652.00
100	0	$6,748,777.45	$67,487.77
101	1	$6,800,006.67	$67,326.80
102	2	$6,851,235.89	$67,168.98
103	3	$6,902,465.11	$67,014.22
104	4	$6,953,694.33	$66,862.45
105	5	$7,004,923.56	$66,713.56
106	6	$7,056,152.78	$66,567.48
107	7	$7,107,382.00	$66,424.13
108	8	$7,158,611.22	$66,283.44
109	9	$7,209,840.43	$66,145.33
110	10	$7,261,069.65	$66,009.72

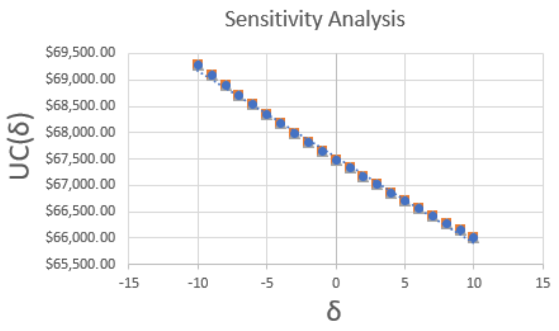


Figure 23 - Plot of δ and UC(δ)

The table and the chart show that as the demand *d* increases, the *UC*, which is NPC per unit of *d,* decreases. For a decision maker, this is a desirable behavior because the initial demand d_0 is just an estimation. If d_0 was underestimated, then the optimal NPC is even better than the value provided by the original recommendation. If d_0 was underestimated, it is easy to determine the reduction in NPC. This would help a decision maker to manage the estimation risk of the demand and consequently yield a higher degree of confidence in the DGS recommendation.

7.2 <u>Sensitivity Analysis 2</u>

To perform analysis number 2, we use the same steps as analysis number 1 with one change. In step 3, we do not instantiate *On(s,r)*, that is, we do not fix the BSN configuration, allowing it to be optimized.

We run all the steps, and for every *δ* in the range -10 to +10, the results are the same as in analysis number 1. In addition, the recommended BSN configuration is also the same. This means that for a delta in the range of -10 to +10, the recommendation is stable.

8. EVALUATION

We now evaluate *Opti-Soft+* and compare it with the two prominent methods IFM and F-EVOLVE* using a realistic case study.

8.1 <u>Case Study Description – Board of Professionals Web Portal</u>

There are many professions including accountants, nurses and teachers that have a board that certifies practitioners. Our case study is for a Board of Professionals that 1) provides certification exams and 2) registers those professionals who pass the exam, that is, certifies them.

Figure 24 shows the two processes: Exam Application (EA) and Board Registration (BR). Initially, there is no software system; consequently, applications for exams and registrations are done manually using paper forms.

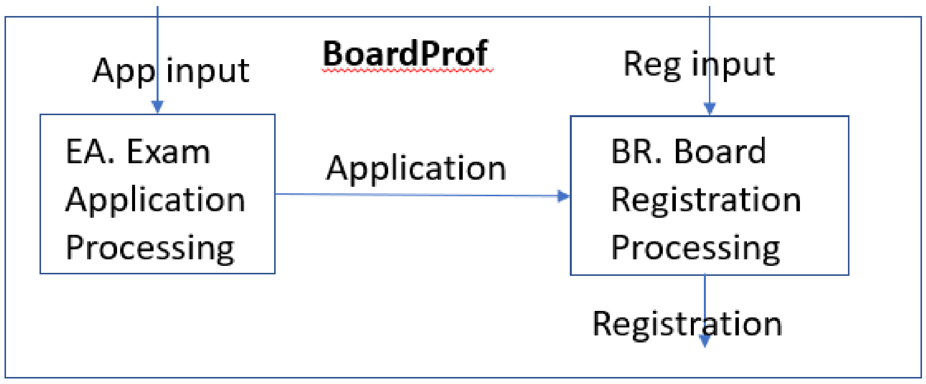

App input
BoardProf
Reg input
EA. Exam Application Processing
Application
BR. Board Registration Processing
Registration

Figure 24 – Board of Professionals: Processes

An information system is planned to be developed to automate the manual processes and ultimately provide a Web portal where both registrations and exam applications can be conducted online. Table 43 shows all the software features that are considered to be developed.

Table 43 - Board of Professionals: Software Features

Feature	Capability	Dependency
F1	Online Exam Application	
F2	Online Exam Application with Document Upload	F1
F3	Online Board Registration	

Table 44 shows that the initial, manual, paper processes are EA1 (Exam Application Paper Processing) and BR1 (Board Registration Paper Processing). Each Exam Application takes 50 minutes to process in EA1 while each Board Registration takes 30 minutes to process in BR1. Processes are enabled by certain implemented software features. Once feature F1 is implemented, EA1 transitions to EA2 (Exam Application Online Processing), which takes only 24 minutes as opposed to 50. Feature F2 enables EA2 to transition to EA3 (Exam Application Online with Doc Upload); feature F3 enables BR1 to transition to BR2 (Board Registration Online Processing); and the availability of both F1 and F3 enables BR2 to transition to BR3 (Application and Registration Online Processing). Table 44 shows all the processes in the business service network and the features that enable them. Because automation brings efficiencies, each subsequent process takes less time than the preceding one. Note that BR3 takes much less time than BR2; the reason is that once F1 and F3, their synergetic interaction allows the following efficiencies:

- Many fields become pre-populated, completely eliminating common errors that take time to resolve,
- All the paper operation is removed, which:
    - Saves labor time to create a physical folder, which has to be created if at least one process is on paper,
    - Saves labor time maintaining paper boxes in a warehouse, and
    - Saves labor time processing paper documents back and forth from warehouse.

Table 44 - Board of Professionals: Processes and Enabling Features

Id	Process	Enabling Feature	Time (min)
EA1	Exam Application Paper Processing		50
EA2	Exam Application Online Processing	F1	24
EA3	Exam Application Online with Doc Upload	F2	12
BR1	Board Registration Paper Processing		30
BR2	Board Registration Online Processing	F3	19
BR3	Application and Registration Online Processing	F1, F3	6

The initial Business Service Network (BSN) is EA1, BR1, and once F1 is developed, the BSN becomes EA2, BR1. These BSNs are shown in Figure 25.

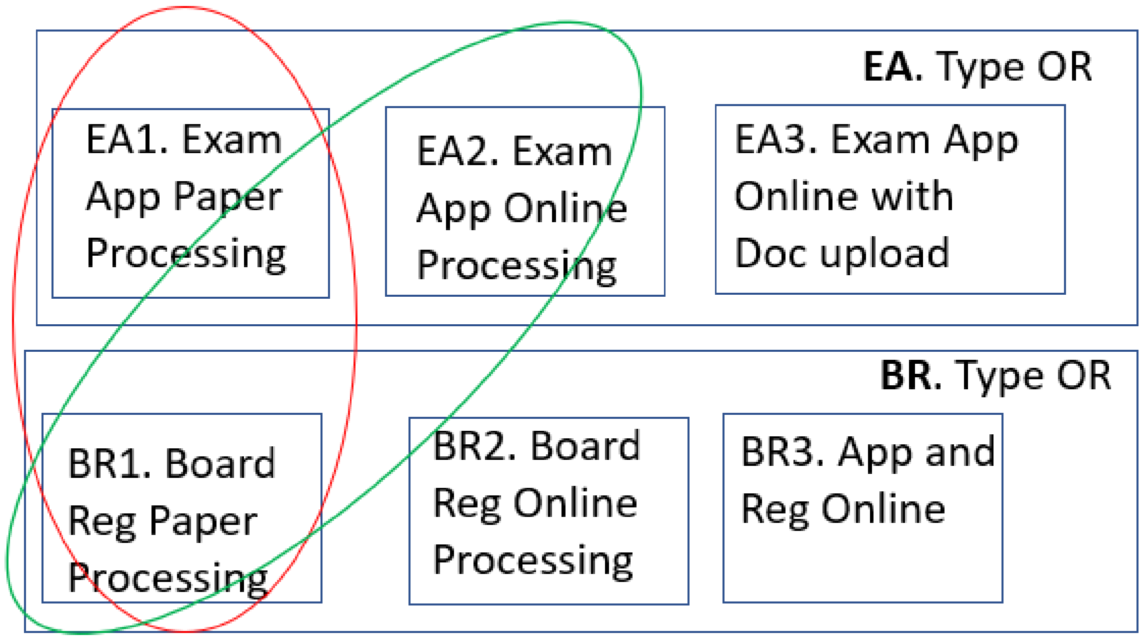

EA. Type OR
EA1. Exam App Paper Processing
EA2. Exam App Online Processing
EA3. Exam App Online with Doc upload
BR. Type OR
BR1. Board Reg Paper Processing
BR2. Board Reg Online Processing
BR3. App and Reg Online

Figure 25 – Board of Professionals: Initial BSN and First Transition

8.2 Case Study – *Opti-Soft*+ Recommendation

We now set *Opti-Soft*+ up to return a recommendation on an optimal release schedule of software features for the Board of Professionals case study. The parameters, in addition to the ones depicted in Section 8.1, are described in the following tables. In order to provide a fair comparison with the prominent methods, we set the discount rate to 0% to be compatible with IFM and F-EVOLVE* methods, which only discount at the end of each release, as opposed to daily discounting in *Opti-Soft*+. Another simplification is that all features are set to have the same size and the team capacity is such that only one feature can be developed per release.

Table 45 – Board of Professionals: Release Scheduling Parameters

BF=	{F1, F2, F3}
TF=	{}
DG=	{(F1, F2)}
FS=	F1 300; F2 300; F3 300
TH=	520
DiscountRate=	0%
NR=	3
RD=	60

Table 46 - Board of Professionals: BSN Parameters

LaborRate:	{"Admin": 160}
NLP=	4
LaborPayDay=	[60,120,180,520]
BSNI=	["App Input","Reg Input"]
BSNO=	{}
Demand(User Application)=	{"App Input": 100,"Reg Input": 100}
ServicesSet=	{BoardProf,EA, BR, EA1, EA2, EA3, BR1, BR2, BR3}
rootID=	BoardProf

Table 47 - Board of Professionals: Composite Services Parameters

id	**Input**	**Output**	**ServiceType**	**Subservices**
BoardProf	App Input Reg Input	Application Registration	ANDservice	{EA, BR}
EA	App Input	Application	ORservice	{EA1, EA2, EA3}
BR	Reg Input	Registration	ORservice	{BR1, BR2, BR3}

Table 48 - Board of Professionals: Atomic Services Parameters

id	RBF	I	RoleTime PerIO	Role
EA1	{}	App Input	0.833	Admin
EA2	{F1}	App Input	0.4	Admin
EA3	{F2}	App Input	0.2	Admin
BR1	{}	Reg Input	0.5	Admin
BR2	{F3}	Reg Input	0.3125	Admin
BR3	{F1, F3}	Reg Input	0.1	Admin

Table 49 - Board of Professionals: Software Development Parameters

TS=	5	
DP=	1	(points/day)
DC=	$1,040	($/point)
OC=	$0.25	($/point/day)
SS=	0	(points)
NSP=	4	
SWPayDay=	60,120,180,520	

Solving the optimization problem in *Opti-Soft+* with the parameters in the tables above, produces an NPV of -$5,281,680.00 and the following software release schedule recommendation for the To-Be state.

Table 50 - Board of Professionals: Release Plan and Corresponding BSN

Release	Feature	BSN
1	F1	EA1, BR1
2	F3	EA2, BR1
3	F2	EA2, BR3
3+		EA3, BR3

We also calculate the NPV for the As-Is, which has the BSN EA1, BR1 for the entire time horizon of 520 days, resulting in -$11,090,560.00. The difference between the NPV of the To-Be and the As-Is is -$5,808,880.00. This means that if the software system is developed according to the release plan in Table 50, the cost savings, or net return on investment is $5,808,880.00 over the investment period of 520 days.

The total cost of the software calculated by *Opti-Soft+* is $1,026,000. This investment, divided by the net return of $5,808,880 produces an ROI of 566%.

8.3 Case Study – Revenue

The next sections will apply the prominent methods IFM and F-EVOLVE* to the Board of Professionals case study. Both methods require that the revenue be externally calculated for each feature in each release. *Opti-Soft+* associates a cost per BSN process, not per feature, but the feature revenue can be calculated.

The revenue of a process is the cash flow generated by the process. In the case of the Board of Professionals case study, the business processes generate a cash outflow. A feature e_1, once implemented, allows a process a_1 to transition to a more efficient process a_2, with a lower Net Present Cost, that is, $NPC(a_2)<NPC(a_1)$. The cost reduction of implementing e_1 is $NPC(a_1)-NPC(a_2)$, while the revenue of e_1 is the cost reduction minus the software development cost. This is the approach we take so that we can compare *Opti-Soft+*, which is based on the cost of a process, with IFM and F-EVOLVE*, which are based on the revenue of features.

For the case study, the *Opti-Soft+* DGS calculated the cost per day for each process, which is shown in the second column of Table 51. The last column is the daily cost reduction, for example, the cost reduction of F1, which activates process EA2, is the cost per day of EA1 ($13,328) minus the cost per day of EA2 ($6,400), resulting in $6,928. Note that BR3 requires F1 and F3 but the corresponding cost reduction of $3,400 cannot be assigned to either F1 or F3. This is a limitation of IFM and F-EVOLVE*, where each revenue is associated with one and only one feature. In IFM and F-EVOLVE* there is no way to map the $3,400 revenue produced when both F1 and F3 are implemented; the revenue of F1 is $6,928 and the revenue of F3 is $3,000.

Table 51 - Board of Professionals: Daily Cost Reduction per Feature

	Cost per Day	Required Feature	Daily Cost reduction
EA1	$13,328.00		$0.00
EA2	$6,400.00	F1	$6,928.00
EA3	$3,200.00	F2	$3,200.00
BR1	$8,000.00		$0.00
BR2	$5,000.00	F3	$3,000.00
BR3	$1,600.00	F1, F3	$3,400.00

Now we consider the software development and operations cost. According to the DGS output, the cost of developing a 300 points feature is $5,200/day while the cost to operate a feature after it is developed is $75/day.

With the information above, we can now calculate the revenue for each feature in each release. Let's start with F1. If F1 is developed in release 1, then the initial version of the EA process is EA1, so during the release 1 period, which is the first 60 days of the time horizon, the revenue per day is the cost reduction of EA1, which is zero, minus the software development cost, which is $5,200. The total revenue for the release, which lasts 60 days is then -$312,000, showing in column 2 of Table 52.

In release 2, the daily revenue for F1 is $6,928 in cost reduction for EA2, minus $75, which is the operations cost, resulting in $6,853 per day and $411,180 for the entire period. The calculations for all 3 releases and the period after release 3 are shown in Table 52. The total revenue for F1, if it is developed in release 1, is the sum of all periods, resulting in $2,840,380.

Table 52 - Board of Professionals: F1 Revenue if Developed in Release 1

	Rel 1	**Rel 2**	**Rel 3**	**Rel 3+**
Active process:	EA1	EA2	EA2	EA2
Cost reduction /day:	$0.00	$6,928.00	$6,928.00	$6,928.00
Soft Cost/day	$5,200.00	$75.00	$75.00	$75.00
Revenue/day:	-$5,200.00	$6,853.00	$6,853.00	$6,853.00
Days in period:	60	60	60	340
Revenue for period:	-$312,000.00	$411,180.00	$411,180.00	$2,330,020.00
			TOTAL:	$2,840,380.00

Applying the same method, the revenues for the other features are calculated and shown in Table 53.

Table 53 - Board of Professionals: Revenue per Feature per Release

Rel	F1	F2	F3
1	$2,840,380	$1,125,500	$1,033,500
2	$2,429,200	$938,000	$858,000
3	$2,018,020	$750,500	$682,500

8.4 Case Study – IFM Recommendation

As described in Section 2.2.1 and in [15], the Incremental Funding Method (IFM) starts by calculating the revenue of each feature in each period. It then sequence-adjusts them, that is, calculates the total revenue for the entire time horizon based on the release when the feature is developed. This Sequence-Adjusted NPV (SANPV) is what we calculated in Section 8.3 and is shown in Table 53.

Once the SANPV is determined, the IFM then chooses the sequence of features. It employs a heuristic algorithm that uses a simple look-ahead approach. The algorithm views sequencing options as strands, which are sequences of features linked by dependencies. In step 1, it starts with strands that reflect the dependency graph with the first feature of each strand being a feature that has no dependency. In our case study, there are three releases, three features and one dependency where F1 is a predecessor of F2. The possible strands for release 1 are then 1, 12 and 3. Table 54 shows the SANPV for each strand as well as the Weight-average SANPV, which is calculated according to the formula.

$$WSANPV = SANPV \times (1 - WF \times (P - 1))$$

where WF is the weighting factor, generally 0.15, and P is the number of periods needed to develop the strand.

Table 54 - Board of Professionals: IFM Selection of Feature in Release 1

Release 1				
Strand	SANPV	WSANPV	Max WSANPV	Feature Selected
1	$2,840,380.00	$2,840,380.00		
12	$3,778,380.00	$3,211,623.00	$3,211,623.00	F1
3	$2,018,020.00	$2,018,020.00		

The strand selected is the one with the highest WSANPV. In release 1, the highest WSANPV corresponds to strand 12, so the feature selected to be developed is F1. In release 2, the strands are 2 and 3 and their SANPV and WSANPV are calculated and shown in Table 55.

Table 55 - Board of Professionals: IFM Selection of Feature in Release 2

Release 2				
Strand	SANPV	WSANPV	Max WSANPV	Feature Selected
2	$938,000.00	$938,000.00	$938,000.00	F2
3	$750,500.00	$750,500.00		

In release 2, the highest WSANPV corresponds to strand 2, so the feature selected to be developed is F2. In release 3, the only strand available is 3 and its SANPV and WSANPV are calculated and shown in Table 56.

Table 56 - Board of Professionals: IFM Selection of Feature in Release 3

Release 3				
Strand	SANPV	WSANPV	Max WSANPV	Feature Selected
3	$682,500.00	$682,500.00	$682,500.00	F3

Based on the steps above, the release plan recommended by IFM for the case study is shown in Table 57. Also shown are the total SANPV, aka cost reduction, which is $4,460,880.

Table 57 - Board of Professionals: IFM Release Plan

Release	Feature	SANPV
1	F1	$2,840,380.00
2	F2	$938,000.00
3	F3	$682,500.00
	Total NPV:	**$4,460,880.00**

The ROI of IFM is 435% ($4,460,880/1,026,000), as opposed to 566% in *Opti-Soft+*.

8.5 Case Study – F-EVOLVE* Recommendation

As described in Section 2.2.2, F-EVOLVE* formulates an Integer Linear Programming (ILP) problem with the goal of offering the most profitable sequence of features. Given a set of features {$F_{1,\dots}$ F_n}, a set of stakeholders {$S_{1,}$ ... S_q}, relative importance λ_p for each S_p, and K releases, *NPV(i,k,p)* as the net present value of a cost estimate for feature i in release k by S_p, the objective function is

$$\sum_{i=1}^{n}\sum_{k=1}^{K}\sum_{p=1}^{q}\lambda_p\, NPV(i,k,p)\; x(i,k)$$

Where: *x(i,k)* is 1 if F_i is selected in release k or 0 otherwise

Assuming only one stakeholder with a relative importance λ_p=1, the objective function becomes:

$$\sum_{i=1}^{n}\sum_{k=1}^{K} NPV(i,k)\, x(i,k)$$

The constraints are as follows:

1. $\sum_{k=1}^{K} x(i,k) \leq 1$
2. $\sum_{i=1}^{n} resource(i,r) x(i,k) \leq Cap(k,r) \ \forall k = 1..K, r = 1..R$
3. $\sum_{k=1}^{K} (K+1-k)\big(x(i_1,k) - x(i_2,k)\big) \geq 0$ if feature i_1 precedes feature i_2

Constraint #1 guarantees that every feature is implemented only once. Constraint #3 guarantees feature dependencies, which in our case is F1 precedes F2. Constraint #2 guarantees that the sum of all development resources *r* for all features *i* developed in release *k,* cannot exceed the capacity of resource *r* in release k. We consider only one resource, the development team, which in the case study, has a capacity of 300 Agile points per release. Features consume points based on their size and only one feature can be developed in each release, so the constraint #2 formula can be simplified as:

$$size(i,k)x(i,k) \leq Cap(k)$$

Where: *size(i,k)* is the size in points of feature *i* if implemented in release *k* and *Cap(k)* is the capacity in points of release *k*.

The first step in F-EVOLVE* is to calculate the NPV for each feature in each release, and that was already done in Section 8.3, and it is displayed in Table 53 and reproduced in Table 58.

Table 58 - Board of Professionals: F-EVOLVE* Revenue per Feature per Release

k	NPV(F1,k)	NPV(F2,k)	NPV(F2,k)
1	\$2,840,380.00	\$1,125,500.00	\$1,033,500.00
2	\$2,429,200.00	\$938,000.00	\$858,000.00
3	\$2,018,020.00	\$750,500.00	\$682,500.00

The second step is to formulate the constraints, shown in Table 59.

Table 59 - Board of Professionals: F-EVOLVE* constraints

$x(1,1)+x(1,2)+x(1,3)$	$\leq$	1	F1 can only be implemented once
$x(2,1)+x(2,2)+x(2,3)$	$\leq$	1	F2 can only be implemented once
$x(3,1)+x(3,2)+x(3,3)$	$\leq$	1	F3 can only be implemented once
$\sum_{i=1}^{3} size(i,1)x(i,1)$	$\leq$	300	Release 1 points can't exceed capacity
$\sum_{i=1}^{3} size(i,2)x(i,2)$	$\leq$	300	Release 2 points can't exceed capacity
$\sum_{i=1}^{3} size(i,3)x(i,3)$	$\leq$	300	Release 3 points can't exceed capacity
$\sum_{k=1}^{3} (K+1-k)\big(x(1,k)-x(2,k)\big)$	$\geq$	0	F1 precedes F2
$x(1,1),x(1,2),x(1,3),$ $x(2,1),x(2,2),x(2,3),$ $x(3.1),x(3,2),x(3,3)$		int	All decision variables are integer

Solving the integer linear program above produces the decision variables instantiations in Table 60. The recommended release plan is identical to the one produced by IFM and shown in Table 57.

Table 60 - Board of Professionals: F-EVOLVE* Instantiated Decision Variables

k	X(F1,k)	X(F2,k)	X(F2,k)
1	1	0	0
2	0	1	0
3	0	0	1

The value of the objective function, according to the solver, is $4,460,880.00, which is the sum of the values in the diagonal of Table 58. This value is also identical to the one produced by IFM.

8.6 Case Study – Comparison

Table 61 summarizes the recommendations from IFM, F-EVOLVE* and *Opti-Soft+* for the case study. IFM and F-EVOLVE* produce exactly the same release plan and cost savings, while *Opti-Soft+* produces a different release plan, a 30% higher cost savings and a 131% higher ROI.

Note that even though IFM and F-EVOLVE* do not have a method for estimation of the revenue, we used *Opti-Soft+* method in order to be able to compare the three approaches, that is, IFM and F-EVOLVE* benefited from *Opti-Soft+* built-in cost model.

Table 61 - Board of Professionals: Recommendations of the Three Methods

	Release Plan	Cost Savings (Net Return on Investment)	ROI
IFM	F1, F2, F3	$4,460,880	435%
F-EVOLVE*	F1, F2, F3	$4,460,880	435%
Opti-Soft+	F1, F3, F2	$5,808,880	566%

To understand the reason that *Opti-Soft+* outperforms IFM and F-EVOLVE*, look at Table 62, which is a reproduction of Table 51. The key difference happens in release 2; IFM and F-EVOLVE* choose F2, while *Opti-Soft+* chooses F3. Note that F2 activates process EA3, which has a cost reduction of $3,200 while F1 and F3 activate BR3, which has a higher cost reduction of $6,400 ($3,000+$3,400). *Opti-Soft+* chooses the highest cost reduction produced by F1, F3 but IFM and F-EVOLVE* are not able to ever activate BR3, because they do not have a mechanism for mapping a revenue to a group of features. In general, *Opti-Soft+* will outperform whenever there is a synergy among a set of features like F1 and F3 above, and the cost reduction of the set is higher than the cost reduction of each feature in the set.

Table 62 - Board of Professionals: Cost Reduction per Feature (reproduction)

	Cost per Day	Required Feature	Cost reduction
EA1	$13,328.00		$0.00
EA2	$6,400.00	F1	$6,928.00
EA3	$3,200.00	F2	$3,200.00
BR1	$8,000.00		$0.00
BR2	$5,000.00	F3	$3,000.00
BR3	$1,600.00	F1, F3	$3,400.00

To resolve this simple example in IFM, stakeholders had to provide 12=(features*(releases+1)) estimations and 5=(releases*2*dependencies) calculations. In F-EVOLVE*, stakeholders had to provide 12=(features*(releases+1)*stakeholders) estimations, and the Integer Linear Programming had to be formulated from scratch with

7=(features*releases) constraints. For a medium-size project with 40 features, 20 releases, 15 dependencies and 2 stakeholders, IFM would require 1,680 stakeholder-provided estimations and 50 calculations while F-EVOLVE would require 1,680 stakeholder-provided estimations and the ILP would have 75 constraints that involve 61 calculations. While the effort to use IFM and F-EVOLVE* has a cubic growth ($f*r*s$), *Opti-Soft+* effort grows linearly according to the formula $f+4a+7$, where f is the number of features, r is the number of releases, s is the number of stakeholders and 4 is the number of atomic processes. For the medium size project, *Opti-Soft+* would require the estimation of 207 cost factors, as opposed to F-EVOLVE which would require 1,741 estimations to determine the revenues and constraints. Note that a cost factor estimation in *Opti-Soft+* requires low effort because it is a single measurable number like labor rate, while each revenue estimation in IFM or E-EVOLVE* require considerable effort to calculate because they are based on multiple cost factors. This basically means that effort wise, it might not be realistic to use IFM and F-EVOLVE* in large projects.

In addition, IFM requires the formulation of a new ILP for every problem while *Opti-Soft+* only requires the setup of parameters describing the problem, because the DGS adapts to the particular parameter configuration.

This leads to the conclusion that IFM and F-EVOLVE* might not be practical for a high number of releases and features.

We did not apply the case study to van den Akker because it is very similar to F-EVOLVE* and would produce the same recommendation. van den Akker does have an extension of the basic model that allows a revenue to be mapped to a pair of features but that complicates the model by forcing the decision variable to have 2 dimensions instead of one and by adding two summation terms to the objective function, one that increases the revenue and another that decreases the revenue. Also, this extension does not change the fact that the revenue must be externally estimated and provided, whereas *Opti-Soft*+ supports many-to-many mapping of features to revenues in a standard way.

9. COMPARISON OF *OPTI-SOFT+* WITH RELATED APPROACHES

We now continue comparing *Opti-Soft+* with the prominent approaches, which we started in Section 1.4.2. We start with characteristics or capabilities.

9.1 Comparison: Characteristics and Capabilities

Table 63 below compares the characteristics of the approaches; it shows that *Opti-Soft+* not only aggregates the best characteristics of each approach but is the only approach that has a built-in valuation model and allow a many-to-many mapping of features to estimations in a native way.

Table 63 - Characteristics of Prominent Approaches Compared to *Opti-Soft+*

Characteristic	IFM	F-EVOLVE*	van den Akker et al.	*Opti-Soft+*
Business benefit metric	Total NPV	Total NPV	Total Revenue	Total NPV
Built in valuation model	No, value estimates are given	No, value estimates are given	No, value estimates are given	Yes
Value metric	Projected Cash Flow	Projected Revenue	Projected Revenue	Calculated Cash Flow
Granularity of value estimation	Feature	Feature	Feature	Business Process
Mapping of feature to an estimation value	One-to-one	One-to-one	One-to-one Many-to-many is handled in a complex way	Many-to-many
Value estimation incorporates software development	Yes	Yes	No	Yes
Time horizon is built into estimations	Yes	Yes	Yes	Yes
Discounting method	Discount at the end of each release period	Discount at the end of each release period	None	Discount when a payment is made

Characteristic	IFM	F-EVOLVE*	van den Akker et al.	*Opti-Soft+*
Release schedule recommendation	For the entire software project	For the entire software project	Only for the next release	For the entire software project
Recommendation method	Heuristics	ILP	ILP	MILP
Supports feature dependencies	Yes	Yes	Yes	Yes
Release size is constrained by the size of feature	No more than 1 feature per release	Yes	Yes	Yes

We now look into the limitations of the current methods and how Opti-Soft closes the limitation gap.

9.2 Comparison: Closing the Limitation Gap

Table 64 below shows that *Opti-Soft+* addresses the limitations of the prominent approaches, which closes the research gap.

Table 64 - Limitations of Prominent Approaches Compared to *Opti-Soft+*

Limitation	IFM	F-EVOLVE*	van den Akker et al.	*Opti-Soft+*
Characteristics that lead to imprecision	1) lack of valuation model, 2) granularity, 3) one-to-one mapping 4) discounting method 5) software maintenance cost not included	1) lack of valuation model, 2) granularity, 3) one-to-one mapping 4) discounting method 5) software maintenance cost not included	1) lack of valuation model, 2) granularity, 3) one-to-one mapping 4) discounting method 5) software cost not included	None
Effort to produce the recommendation (f=features, r=releases, s=stakeholder, d=dependencies)	(f*r*s) stakeholder-provided estimations + (r+2d) calculations	(f*r*s) stakeholder-provided estimations + (f+r+d) constraints to setup the ILP + running ILP	(f*r) stakeholder-provided estimations + (f+r+d) constraints to setup the ILP + running ILP	9 cost parameters + running DGS

Limitation	**IFM**	**F-EVOLVE***	**van den Akker et al.**	***Opti-Soft+***
Valuation effort when factors change	Very high. Manual recalculations needed when any factor change such as labor rates, discount rate, investment period, etc...	Very high. Manual recalculations needed when any factor change such as labor rates, discount rate, investment period, etc...	Very high. Manual recalculations needed when any factor change such as labor rates, discount rate, investment period, etc...	Low, just change the affected parameter and rerun the DGS
Effort to conduct 'what if' or sensitivity analysis	High, require manual recalculations	High, require manual recalculations	High, require manual recalculations	Low, any change is exposed as a data point
Reason discounting causes imprecision	Done at the end of the release	Done at the end of the release	Not done	None

9.3 Comparison: Claims

For an important and very common class of information systems that improve a Business Service Network, we make the following claims about *Opti-Soft+:*

- Claim 1: it is as good as the current methods for all cases and outperforms them in cases of synergy among features. The reason is that it solves for optimality, like the best methods, and allows many-to-many mapping of feature to benefit, while the others are limited to one-to-one mapping.
- Claim 2: it outperforms in accuracy, due to its comprehensive cost/benefit model, while others rely on subjective, external estimations.
- Claim 3: it requires considerably less effort and is applicable to large projects, because it is reusable and parameterized, while others require a MP formulation for each use case and cost estimations are done manually.

10. CONCLUSION

The motivation for this dissertation is that many projects fail because they don't deliver value back to the business that is commensurate to the investment. One strategy to improve the failure rate is to treat software development as an investment by adopting a value-oriented approach. Of the many types of approaches that arouse, the financial-based approach is the most successful. Several financial-based approaches were proposed but all of them, including the most popular, the Incremental Funding Method (IFM), have lots of limitations, the main one being inaccuracy, because they lack a built-in cost/benefit model of the investment environment, they just assume that external stakeholders provide the cost/benefit estimates. They are also very inflexible because they require all estimates to be recalculated manually whenever a single cost factor, like a labor rate, is changed.

In this dissertation, we addressed the above problem by focusing on a specific class of information systems that reduce the operations cost of a Business Service Network (BSN). For this class of systems, we developed a value-oriented, financial-based framework that (1) delivers business value commensurate to the investment, and (2) is accurate and has less limitations than the prominent methods.

The framework is composed of a formal model, a DGS and a methodology for recommending a release schedule that optimizes the overall business value, i.e., minimizes the aggregate cost of the BSN operations and the development of the software. The DGS, given parameters and decision variables as input, and the analytical model, which is automatically translated into a MILP formulation, uses a MILP solver to produce the recommendation. The release schedule recommendation (1) is optimal, i.e., there is no other release schedule that has a lower total cost; (2) covers all aspects of the investment domain including the BSN, the software development process, and the savings achieved due to automation; (3) is accurate, because it is based on the analytical model, which contains all the necessary factors and equations to precisely compute valuation points, decision variables, constraints, and the objective function; (4) associates benefits to a set of interacting software features instead of only one as in the prominent approaches; (5) lacks manual estimations; and (6) is flexible to change valuation factors and conduct 'what if' or sensitivity analysis.

The framework is novel and unique in many ways, such as (1) it significantly improves the precision of the valuation points, because it (a) encompasses all aspects of the investment domain (business, software, release plan) and (b) includes all necessary cost factors and equations to precisely compute valuation points, decision variables, constraints, and the objective function; (2) it leverages the insight that there is a direct correlation between the implementation of a set of software features and the improvement in a business process, and that the degree of the improvement can be accurately calculated; (3) it completely eliminates the effort to manually estimate or re-estimate valuation points by exposing valuation factors as parameters that can be easily changed prior to using the DGS to re-compute the recommendation; (4) it significantly reduces the effort to make changes or conduct 'what if' or sensitivity analysis, because different scenarios and sensitivities can be easily analyzed by changing one or more parameters, rerunning the DGS and comparing the results; (5) it is written in a high level, general purpose language, not requiring expertise in Mathematical Programming; and (6) it is modular, extensible and reusable, not requiring the formulation of a new MILP model for each use case.

For an important and very common class of information systems that improve a Business Service Network, we claim that *Opti-Soft+*: (1) is as good as the current methods for all cases and outperforms them in cases of synergy among features; (2) outperforms in accuracy; and (3) requires considerably less effort and is applicable to large projects.

The contributions of the dissertation are: 1) the formal analytical model, 2) the methodology and the DGS, and 3) a technique for sensitivity analysis, and 4) an evaluation.

We conclude that the dissertation fulfils the thesis, because the *Opti-Soft+* framework (1) recommends a release schedule of software features; (2) minimizes the combined cost of software development and improved business operations; and (3) outperforms the existing approaches in terms of added business value and required effort.

11. LIMITATIONS AND FUTURE WORK

In terms of limitations, the major one is the scope of software projects that *Opti-Soft+* supports, which are projects that implement information systems that improve a business service network.

This limitation is not as strong as it seems because it can be applied to any domain that can be modelled as a business service network. The most common BSN is an internal company's process that is labor intensive. Other examples of a BSN is a manufacturing process where robots would be used to reduce the cost and time.

A second limitation is that it clumps all developers into one group and models them as a single team with a single set of cost factors, e.g., developers' productivity and labor cost. Many projects do involve multiple development teams, so *Opti-Soft+* forces the user to provide a single set of numbers, probably the average, of all teams. If there is a large cost disparity among teams, this could be problematic.

A third limitation is that it covers only developers, even though in many projects, other professionals like UX designers, testers and business analysts are part of the team.

Future work could remove the second and third limitations. The software development formal model could be extended to support multiple teams with multiple types of professionals. This level of granularity would have the following impact:

1. In the software formal model, the following parameters would have to be extended to support multiple teams: team size, developer productivity, developer cost, operations cost and system size

2. The feature size parameter would have to be extended because different teams might estimate the size of the feature differently.
3. The recommendation would have to be extended to assign features to a specific team in a particular release

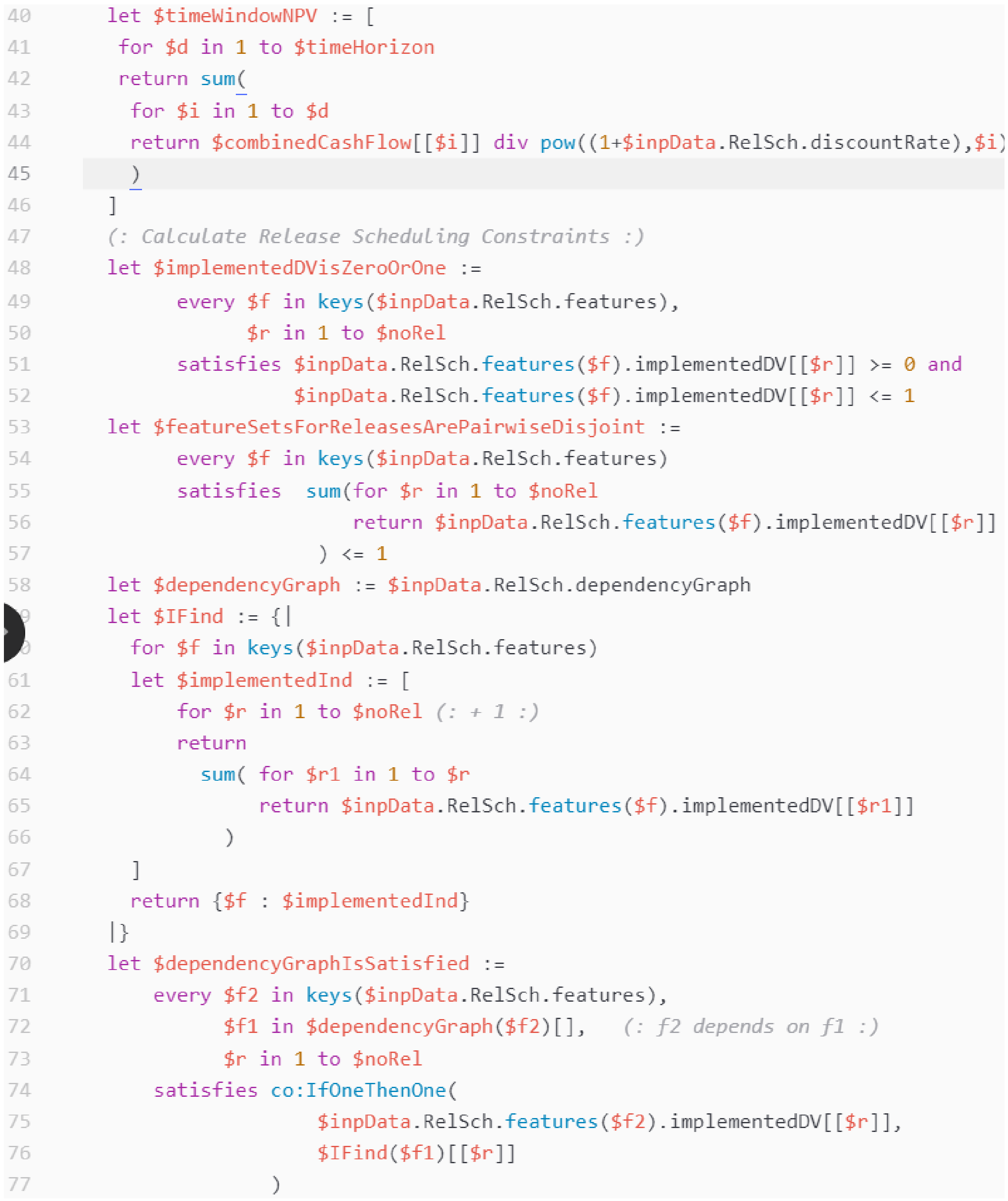
Future work could relax the first limitation. One improvement would be for the BSN to generate revenue instead of consuming labor. This improvement would allow *Opti-Soft+* to support a wide-range of applications, for example, a free application, web-based or mobile, that has several potential use cases where each use case would bring an expected revenue through ads. Each use case would be mapped to a BSN atomic service that would be activated by the development of a set of features.

APPENDIX 1 – DGS SOURCE CODE

ReleaseScheduling.jq

```
jsoniq version "1.0";
module namespace rs =
  "http://kb.dgms.io.lib/softwareFeatureSelection/ReleaseSchedulingModel.jq";
import module namespace sd =
  "http://kb.dgms.io.lib/softwareFeatureSelection/SofDevModel.jq";
import module namespace bsn =
  "http://kb.dgms.io.lib/softwareFeatureSelection/BSNModel.jq";
import module namespace co =
  "http://kb.dgms.io.lib/softwareFeatureSelection/CommonOperations.jq";

(:----- ReleaseSchedulingModel function --------------------:)
declare function rs:ReleaseSchedulingModel($inpData) {
  let $noRel       := $inpData.RelSch.noReleases
  let $timeHorizon := $inpData.RelSch.timeHorizon
  (: Calculate Release Scheduling Metrics part 1:)
  let $soFarIBF  := {|
    for $f in $inpData.RelSch.businessFeatures[]
    let $ImplementedInd := [
        for $r in 1 to $noRel + 1
        return
            if ($r = 1)
            then  0
            else sum( for $r1 in 1 to $r - 1
                      return $inpData.RelSch.features($f).implementedDV[[$r1]]
            )
    ]
    return {$f : $ImplementedInd}
  |}
  (: Invoke Lower Models to Calculate Metrics and Constraints :)
  let $sofDevModel := sd:SofDevModel($inpData)
  let $BSNModel    := bsn:BSNModel($inpData,
                                   $soFarIBF,
                                   $sofDevModel.interface.firstDay,
                                   $sofDevModel.interface.lastDay)
  (: Calculate Release Scheduling Metrics part 2:)
  let $combinedCashFlow := [
    for $d in 1 to $timeHorizon
    return $BSNModel.metrics.cashFlow[[$d]]+$sofDevModel.metrics.cashFlow[[$d]]
  ]
```



```
let $timeWindowNPV := [
 for $d in 1 to $timeHorizon
 return sum(
  for $i in 1 to $d
  return $combinedCashFlow[[$i]] div pow((1+$inpData.RelSch.discountRate),$i)
  )
]
(: Calculate Release Scheduling Constraints :)
let $implementedDVisZeroOrOne :=
      every $f in keys($inpData.RelSch.features),
            $r in 1 to $noRel
      satisfies $inpData.RelSch.features($f).implementedDV[[$r]] >= 0 and
                $inpData.RelSch.features($f).implementedDV[[$r]] <= 1
let $featureSetsForReleasesArePairwiseDisjoint :=
      every $f in keys($inpData.RelSch.features)
      satisfies  sum(for $r in 1 to $noRel
                     return $inpData.RelSch.features($f).implementedDV[[$r]]
                  ) <= 1
let $dependencyGraph := $inpData.RelSch.dependencyGraph
let $IFind := {|
  for $f in keys($inpData.RelSch.features)
  let $implementedInd := [
      for $r in 1 to $noRel (: + 1 :)
      return
        sum( for $r1 in 1 to $r
             return $inpData.RelSch.features($f).implementedDV[[$r1]]
          )
  ]
  return {$f : $implementedInd}
|}
let $dependencyGraphIsSatisfied :=
    every $f2 in keys($inpData.RelSch.features),
          $f1 in $dependencyGraph($f2)[],   (: f2 depends on f1 :)
          $r in 1 to $noRel
    satisfies co:IfOneThenOne(
                  $inpData.RelSch.features($f2).implementedDV[[$r]],
                  $IFind($f1)[[$r]]
              )
    let $constraints:=  $implementedDVisZeroOrOne                     and
                        $featureSetsForReleasesArePairwiseDisjoint and
                        $dependencyGraphIsSatisfied                   and
                        $BSNModel.constraints                         and
                        $sofDevModel.constraints
    (: Return Release Metrics Interface:)
    return {
        constraints : $constraints,
        metrics: {
          timeWindowNPV : $timeWindowNPV
        },
        BSNModel:                       $BSNModel,
        sofDevModel:                    $sofDevModel
    }
  }; (: end of ReleaseSchedulingModel function:)
```

BSNModel.jq

```
jsoniq version "1.0";
module namespace bsn =
  "http://kb.dgms.io.lib/softwareFeatureSelection/BSNModel.jq";
import module namespace sm =
  "http://kb.dgms.io.lib/softwareFeatureSelection/SvcModel.jq";
import module namespace co=
  "http://kb.dgms.io.lib/softwareFeatureSelection/CommonOperations.jq";

(:----- BSNModel Function --------------------:)
declare function bsn:BSNModel($inpData,$SoFarIBF,$firstDay,$lastDay) {
  let $noRel       := $inpData.RelSch.noReleases
  let $timeHorizon := $inpData.RelSch.timeHorizon
  let $root        := $inpData.BSN.rootId
  (: Invoke Service Model  to Calculate Metrics and Constraints :)
  let $svcModel  := sm:SvcModel($inpData,$SoFarIBF)
  let $costPerDayKV  := {|
    for $r in 1 to size($svcModel.costPerDay)
      return {$r: $svcModel.costPerDay[[$r]]}
  |}
```

```
    (:-- Calculate BSN Constraints --:)
    let $BSNDemandIsSatisfied :=
      every $i in $inpData.BSN.BSNInputs[],
            $r in 1 to $noRel+1
      satisfies (
        $svcModel.serviceMetrics($root).metrics.inputThru[[$r]]($i) >=
          $inpData.BSN.demand($i) and
        $svcModel.serviceMetrics($root).metrics.inputThru[[$r]]($i) <= 1000
        )
    let $BSNSupplyIsSatisfied :=
      every $o in $inpData.BSN.BSNOutputs[],
            $r in 1 to $noRel+1
      satisfies (
        $svcModel.serviceMetrics($root).metrics.outputThru[[$r]]($o) >=
          $inpData.BSN.demand($o)
      )
    let $constraints := $BSNDemandIsSatisfied and
                        $BSNSupplyIsSatisfied and
                        $svcModel.constraints
    (:-- Calculate Cash Flow Metric  --:)
    let $BSNCostForDay := [
      for $d in 1 to $timeHorizon
        let $rel := sum(
          for $r in 1 to $noRel + 1
            let $satisfyBoundary1 := if ($firstDay[[$r]] <= $d) then 1 else 0
            let $satisfyBoundary2 := if ($d <= $lastDay[[$r]]) then 1 else 0
            return
              if ($satisfyBoundary1 + $satisfyBoundary2 >= 2)
                then $r
                else 0
        )
        return $costPerDayKV($rel)
    ]
    let $BSNPayment := [
      for $p in 1 to $inpData.BSN.noLaborPayments
      return
        if ($p=1)
          then sum(
            for $d in 1 to $inpData.BSN.laborPayDay[[$p]]
            return $BSNCostForDay[[$d]]
          )
          else sum(
            for $d in $inpData.BSN.laborPayDay[[$p - 1]]+1
                   to $inpData.BSN.laborPayDay[[$p]]
            return $BSNCostForDay[[$d]]
          )
    ]
    let $cashFlow := [
      for $d in 1 to $timeHorizon
      return
        if (some $p in 1 to $inpData.BSN.noLaborPayments
           satisfies $d = $inpData.BSN.laborPayDay[[$p]])
        then
          for $p in 1 to $inpData.BSN.noLaborPayments
                where $d = $inpData.BSN.laborPayDay[[$p]]
           return -$BSNPayment[[$p]]
        else 0
    ]
    (:-- Return BSN Interface --:)
    return {
      constraints : $constraints,
      metrics: {
        cashFlow : $cashFlow
      },
      svcModel : $svcModel
    }
  }; (: end of BSNModel function:)
```

SvcModel.jq

```
jsoniq version "1.0";
module namespace sm =
  "http://kb.dgms.io.lib/softwareFeatureSelection/SvcModel.jq";
import module namespace cs =
  "http://kb.dgms.io.lib/softwareFeatureSelection/CompositeService.jq";
import module namespace idas =
  "http://kb.dgms.io.lib/softwareFeatureSelection/InpDrivenAtomicService.jq";
(:----- Service Model Function --------------------:)
declare function sm:SvcModel($inpData,$soFarIBF) {
  let $BSNroot := $inpData.BSN.rootId
  let $serviceMetrics := sm:ComputeService($inpData,$soFarIBF,$BSNroot)
  let $costPerDay := $serviceMetrics($BSNroot).metrics.costPerDay
  let $constraints:= $serviceMetrics($BSNroot).constraints
  return { BSNroot: $BSNroot,
           constraints: $constraints,
           costPerDay: $costPerDay,
           serviceMetrics: $serviceMetrics
  }
}; (: end SvcModel function :)

(:----- Service Metrics Computation: ComputeService ------------:)
declare function sm:ComputeService($inpData,$soFarIBF,$svc) {
  let $noRel       := $inpData.RelSch.noReleases
  let $svcType     := $inpData.services($svc).serviceType
  let $subServices := $inpData.services($svc).subService[]
  let $serviceMetrics :=
    if ($svcType = "InputDrivenAtomic" ) then
      (: -- Atomic service -- :)
      {$svc: idas:InpDrivenAtomicService($inpData,$soFarIBF,$svc)}
    else
      (: -- Composite service (type 'AND' or 'OR') -- :)
      (: -- Recursively calculate metrics of subservices -- :)
      let $subServiceMetrics := {|
        for $s in $subServices
        return sm:ComputeService($inpData,$soFarIBF,$s)
      |}

```

```
        (:-- Calculate constraints and metrics of Composite Service  --:)
        let $compositeService :=
            cs:CompositeService($inpData,$svc,$subServiceMetrics)
        let $compositeMetrics := {
          $svc: {
              type: $svcType,
              constraints: $compositeService.constraints,
              metrics: $compositeService.metrics,
              debug: $compositeService.debug
          }
        }
        (:-- return Composite Service interface, which includes subservices -- :)
        return {| ( $compositeMetrics , $subServiceMetrics ) |}
    (:-- return interface of root service -- :)
    return $serviceMetrics
}; (: end ComputeService function :)
```

CompositeService.jq

```
jsoniq version "1.0";
module namespace cs =
    "http://kb.dgms.io.lib/softwareFeatureSelection/CompositeService.jq";
import module namespace co =
    "http://kb.dgms.io.lib/softwareFeatureSelection/CommonOperations.jq";
(:----- Composite Service Function --------------------:)
declare function cs:CompositeService($inpData,$svc,$subServiceMetrics) {
  let $svcType            := $inpData.services($svc).serviceType,
      $noRel              := $inpData.RelSch.noReleases,
      $subServices        := $inpData.services($svc).subService[],
      $noSubsvc           := size($inpData.services($svc).subService),
      $qtySubsvcRequired := if ($svcType = "AND") then $noSubsvc else 1
  (: -- Compute Metrics -- :)
  (:-- Compute sets of input and output flows --:)
  let $subServicesInput :=
    for $ss in $subServices
    return $inpData.services($ss).input[]
  let $subServicesOutput :=
    for $ss in $subServices
    return $inpData.services($ss).output[]
    let $setAllIO := distinct-values({
                      $inpData.services($svc).input[],
                      $inpData.services($svc).output[],
                      $subServicesInput,
                      $subServicesOutput})
    let $setExtIO := distinct-values({
                      $inpData.services($svc).input[],
                      $inpData.services($svc).output[]})
    let $setIntOnlyIO := { for $j in $setAllIO
                            where every $k in $setExtIO satisfies $k ne $j
                            return  $j
    }
    (: -- Compute InternalSupply --:)
     let $internalSupply := [
      for $r in 1 to $noRel + 1
      return {|
        for $j in $setAllIO
        let $summation :=
          sum (
            for $s in $subServices
            where some $k in $inpData.services($s).output[] satisfies $k eq $j
            return $subServiceMetrics($s).metrics.outputThru[[$r]]($j)
          )
        return {$j: $summation}
      |}
    ]
    (: -- Compute InternalDemand --:)
     let $internalDemand := [
      for $r in 1 to $noRel + 1
      return {|
        for $j in $setAllIO
        let $summation :=
          sum (
            for $s in $subServices
            where some $k in $inpData.services($s).input[] satisfies $k eq $j
            return $subServiceMetrics($s).metrics.inputThru[[$r]]($j)
          )
        return {$j: $summation}
      |}
    ]
    (: -- Compute inputThru  --:)
    let $inputThru := [
      for $r in 1 to $noRel + 1
      return {|
        for $i in $inpData.services($svc).input[]
        let $flow := $internalDemand[[$r]]($i) - $internalSupply[[$r]]($i)
        return {$i: $flow}
      |}
    ]
    (: -- Compute outputThru  --:)
    let $outputThru := [
      for $r in 1 to $noRel + 1
      return {|
        for $o in $inpData.services($svc).output[]
        let $flow := $internalSupply[[$r]]($o) - $internalDemand[[$r]]($o)
        return {$o: $flow}
      |}
    ]
    (: ---- Compute totalSupply & totalDemand --- :)
    let $totalDemand := [
     for $r in 1 to $noRel + 1
     return {|
       for $j in $setAllIO
       let $result :=
           if (some $k in $inpData.services($svc).output[] satisfies $k eq $j)
            then $internalDemand[[$r]]($j) + $outputThru[[$r]]($j)
            else $internalDemand[[$r]]($j)
       return {$j: $result}
     |}
    ]
    let $totalSupply := [
     for $r in 1 to $noRel + 1
     return {|
       for $j in $setAllIO
       let $result :=
           if (some $k in $inpData.services($svc).input[] satisfies $k eq $j)
            then $internalSupply[[$r]]($j) + $inputThru[[$r]]($j)
            else $internalSupply[[$r]]($j)
       return {$j: $result}
     |}
    ]
    (: ---- Compute Constraints --- :)
    (: ---- Calculate activatedService constraint ---- :)
    let $activatedSubservice := [
      for $r in 1 to $noRel + 1
        let $qtySubsvc := sum(
          for $ss in $inpData.services($svc).subService[]
          return $inpData.services($ss).onDV[[$r]]
        )
      return
        ($qtySubsvc = ($qtySubsvcRequired * $inpData.services($svc).onDV[[$r]]))
    ]
    let $activatedSubserviceConstraint :=
      every $r in 1 to size($activatedSubservice)
      satisfies $activatedSubservice[[$r]]
    (: Calculate Total Balancing Constraints :)
    let $totalSupplyMatchesTotalDemandConstraint :=
      every $j in $setAllIO,
            $r in 1 to $noRel + 1
      satisfies $totalSupply[[$r]]($j) >= $totalDemand[[$r]]($j) and
                $totalSupply[[$r]]($j) <= $totalDemand[[$r]]($j)
    (: onDV can be 0 or 1 :)
    let $onDVisZeroOrOne :=
      every $r in 1 to $noRel + 1
      satisfies $inpData.services($svc).onDV[[$r]] >= 0 and
                $inpData.services($svc).onDV[[$r]] <=1
    let $rootOnDVisOne :=
      if ($svc ne $inpData.BSN.rootId)
        then true
        else
          every $r in 1 to $noRel + 1
          satisfies $inpData.services($svc).onDV[[$r]] = 1
    let $subServiceConstraints :=
      every $s in $subServices
      satisfies $subServiceMetrics($s).constraints
    let $constraints := $activatedSubserviceConstraint           and
                        $totalSupplyMatchesTotalDemandConstraint and
                        $onDVisZeroOrOne                         and
                        $rootOnDVisOne                           and
                        $subServiceConstraints
    (: -- Aggregate costPerDay of subservices -- :)
    let $costPerDay := [
      for $r in 1 to $noRel + 1
      return sum(
        for $s in $subServices
        return $subServiceMetrics($s).metrics.costPerDay[[$r]]
      )
    ]
    (: return interface :)
    return {
      type: $svcType,
      constraints : $constraints,
      metrics: {
        costPerDay:     $costPerDay,
        inputThru:     $inputThru,
        outputThru:    $outputThru
      }
    }
  }; (: end ofCompositeService Function :)
```

InpDrivenAtomicService.jq

```
jsoniq version "1.0";
module namespace idas =
  "http://kb.dgms.io.lib/softwareFeatureSelection/InpDrivenAtomicService.jq";
import module namespace co=
  "http://kb.dgms.io.lib/softwareFeatureSelection/CommonOperations.jq";
(:----- Input Driven At$pue Model Function ---------------------:)
declare function idas:InpDrivenAtomicService($inpData,$soFarIBF,$svc) {
  let $svcType := $inpData.services($svc).serviceType,
      $serviceCostPerDay := $inpData.services($svc).serviceCostPerDay,
      $costPerInput := $inpData.services($svc).costPerInput,
      $costPerOutput := $inpData.services($svc).costPerOutput,
      $noRel := $inpData.RelSch.noReleases,
      $timeHorizon := $inpData.RelSch.timeHorizon
      let $inputThru := $inpData.services($svc).inputThruDV
  (: Calculate constraints :)
  let $featureDependencyIsSatisfied :=
    every $r in 1 to $noRel + 1,
          $f in $inpData.services($svc).requiredBusinessFeature[]
    satisfies co:IfOneThenOne(           (: (1-$rhs) <= (1-$lhs) :)
             $inpData.services($svc).onDV[[$r]],
             $soFarIBF.$f[[$r]])
    let $deactivatedServicesIsSatisfied1 :=
        every $r in 1 to $noRel + 1,
        $i in keys($inpData.services($svc).inputThruDV[[$r]])
        satisfies co:IfZeroThenEqual($inpData.services($svc).onDV[[$r]],
                      $inpData.services($svc).inputThruDV[[$r]]($i), 0)
    let $outputThru := [
      for $r in 1 to $noRel + 1
      return {|
        for $o in $inpData.services($svc).output[]
        let $summation :=
          sum(
            for $i in $inpData.services($svc).input[]
            return $inpData.services($svc).IOThruRatio($i)($o) *
                   $inpData.services($svc).inputThruDV[[$r]]($i)
          )
        return {$o: $summation}
      |}
    ]
    let $timePerDay := {|
      for $l in $inpData.services($svc).serviceRole[]
      let $timePerRel := [
        for $r in 1 to $noRel + 1
        let $sumInputTime := sum(
          for $i in $inpData.services($svc).input[]
          return $inpData.services($svc).roleTimePerIO($l)($i) *
                 $inpData.services($svc).inputThruDV[[$r]]($i)
          )
        let $sumOutputTime := sum(
          for $o in $inpData.services($svc).output[]
          return $inpData.services($svc).roleTimePerIO($l)($o) *
                 $outputThru[[$r]]($o)

          )
        return $sumInputTime + $sumOutputTime
      ]
      return {$l:$timePerRel}
    |}
    let $laborCostPerDay := [
      for $r in 1 to $noRel + 1
      return  sum(
        for $l in $inpData.services($svc).serviceRole[]
        return $inpData.BSN.laborRoleRate($l) * $timePerDay($l)[[$r]]
      )
    ]
    let $flowCostPerDay := [
      for $r in 1 to $noRel + 1
      let $sumInputCost := sum(
        for $i in keys($costPerInput)
        return $costPerInput($i) * $inpData.services($svc).inputThruDV[[$r]]($i)
        )
      let $sumOutputCost := sum(
        for $o in keys($costPerOutput)
        return $costPerOutput($o) * $outputThru[[$r]]($o)

        )
      return $sumInputCost + $sumOutputCost
    ]

    let $costPerDay := [
      for $r in 1 to $noRel + 1
      return $laborCostPerDay[[$r]] +
             $flowCostPerDay[[$r]] +
             $serviceCostPerDay * $inpData.services($svc).onDV[[$r]]
    ]
    let $inputThruDVsum := [
      for $r in 1 to $noRel + 1
      return sum(
            for $j in keys($inpData.services($svc).inputThruDV[[$r]])
            return $inpData.services($svc).inputThruDV[[$r]]($j)
            )
    ]
    let $inputThruDVboundConstraint :=
      every $r in 1 to $noRel + 1
      satisfies $inputThruDVsum[[$r]] >= 0
    let $onDVisZeroOrOne :=
      every $r in 1 to $noRel + 1
      satisfies $inpData.services($svc).onDV[[$r]] >= 0 and
                $inpData.services($svc).onDV[[$r]] <= 1
    let $constraints := $featureDependencyIsSatisfied    and
                        $deactivatedServicesIsSatisfied1 and
                        $inputThruDVboundConstraint      and
                        $onDVisZeroOrOne
    (: Return Interface :)
    return {
      type: $svcType,
      constraints : $constraints,
      metrics: {
        costPerDay:        $costPerDay,
        inputThru:        $inputThru,
        outputThru:       $outputThru
      }
    }
  }; (: end InpDrivenAtomicService function :)
```

SofDevModel.jq

```
jsoniq version "1.0";
module namespace sd =
    "http://kb.dgms.io.lib/softwareFeatureSelection/SofDevModel.jq";
import module namespace co=
    "http://kb.dgms.io.lib/softwareFeatureSelection/CommonOperations.jq";
(:----- Software Development Model Function --------------------:)
declare function sd:SofDevModel($inpData) {
  let $noRel       := $inpData.RelSch.noReleases
  let $timeHorizon := $inpData.RelSch.timeHorizon
  let $resSet      := $inpData.RelSch.resSet
  let $resCost     := $inpData.RelSch.resCost
  (:-- Calculate actual release size -- :)
  let $relSize := [
      for $r in 1 to $noRel
      return sum(
          for $j in keys($inpData.RelSch.features)
          return ( $inpData.RelSch.features($j).size *
                   $inpData.RelSch.features($j).implementedDV[[$r]])
      )
  ]
```

```
    (:-- Calculate release capacity   --:)
    let $relCapacity := [
      for $r in 1 to $noRel
        return
             $inpData.SofDev.teamSize[[$r]] *
             $inpData.SofDev.devProductivity[[$r]] *
             $inpData.RelSch.releaseDuration[[$r]]
    ]
    (:-- Calculate CapacityIsZeroWhenSizeIsZero :)
    let $capacityIsZeroWhenSizeIsZero :=
        every $r in 1 to $noRel
        satisfies co:IfZeroThenEqual($relSize[[$r]],$relCapacity[[$r]], 0)
    (:-- Calculate Capacity Constraint --:)
    let $releaseSizeCannotExceedCapacity :=
        every  $r in 1 to $noRel
          satisfies $relSize[[$r]] <= $relCapacity[[$r]]
    (:-- Calculate Release Dates --:)
    let $firstDay := [
      for $r in 1 to $noRel + 1
        return
          if ($r=1)
            then 1
            else
              sum(
                for $j in 1 to $r - 1
                  return $inpData.RelSch.releaseDuration[[$j]]
                  ) + 1
    ]
    let $lastDay := [
      for $r in 1 to $noRel + 1
        return
          if ($r=$noRel + 1)
            then $timeHorizon
            else $firstDay[[$r + 1]] - 1
    ]
    (:-- Calculate CashFlow  --:)
    let $devCostPerDay := {|
      for $r in 1 to $noRel + 1
        return {$r :
          if ($r=$noRel + 1)
            then 0
        else
          $relCapacity[[$r]] div $inpData.RelSch.releaseDuration[[$r]]
          * $inpData.SofDev.devCost
    }
|}
let $opsCostPerDay := {|
  for $r in 1 to $noRel + 1
    return {$r:
      if ($r=1)
        then $inpData.SofDev.systemSize * $inpData.SofDev.opsCost
        else
          (sum(for $i in 1 to $r - 1 return $relCapacity[[$i]])
            +$inpData.SofDev.systemSize) * $inpData.SofDev.opsCost
    }
|}
let $SWCostForDay := {|
   for $r in 1 to $noRel + 1
   let $SWcostPerDay := $devCostPerDay($r) + $opsCostPerDay($r)
   for $d in $firstDay[[$r]] to $lastDay[[$r]]
   return {$d: $SWcostPerDay}
|}
let $SWPayment := [
  for $p in 1 to $inpData.SofDev.noSoftwarePayments
    return
      if ($p=1)
        then sum(
          for $d in 1 to $inpData.SofDev.softwarePayDay[[$p]]
            return $SWCostForDay($d)
        )
        else sum(
          for $d in $inpData.SofDev.softwarePayDay[[$p - 1]]+1
            to $inpData.SofDev.softwarePayDay[[$p]]
            return $SWCostForDay($d)
        )
]
let $laborCashFlow := [
  for $d in 1 to $timeHorizon
    return
     if (some $p in 1 to $inpData.SofDev.noSoftwarePayments
          satisfies $d=$inpData.SofDev.softwarePayDay[[$p]])
      then
```

```
              for $p in 1 to $inpData.SofDev.noSoftwarePayments
                  where $d eq $inpData.SofDev.softwarePayDay[[$p]]
                  return -$SWPayment[[$p]]
            else 0
      ]
      let $relRes := [
        for $r in 1 to $noRel
        let $implementedFeature := [
            for $f in keys($inpData.RelSch.features)
            return
                if ($inpData.RelSch.features($f).implementedDV[[$r]] = 1
                and $inpData.RelSch.features($f).featureRes ne "")
                then $inpData.RelSch.features($f).featureRes
                else ()
        ]
        return $implementedFeature
      ]
      let $cumRelRes := [
        for $r in 0 to $noRel
          let $unionRelRes := [
            if ($r=0) then () else (
              for $r1 in 1 to $r
              let $array := $relRes[[$r1]]
              return $array[])
          ]
          return $unionRelRes
      ]
      let $newRelRes := [
        for $r in 1 to $noRel
          let $diff := [
          for $i in 1 to size($relRes[[$r]])
            let $sum := sum(
              for $j in 1 to size($cumRelRes[[$r - 1]])
              return if ($relRes[[$r]][[$i]] = $cumRelRes[[$r - 1]][[$j]])
                        then 1
                        else 0
            )
            return {if (size($cumRelRes[[$r - 1]]) = 0 or $sum = 0)
                      then $relRes[[$r]][[$i]]
                      else ()}
          ]
          return $diff
      ]
      let $resCostPerRel := [
        for $r in 1 to $noRel
          let $arrayEntry := $newRelRes[[$r]]
          return sum(for $e in $arrayEntry[] return $resCost($e))
      ]

      let $resCashFlow := [
        for $d in 1 to $timeHorizon
          return
            if (some $r in 1 to $noRel satisfies $d = $firstDay[[$r]])
            then
              for $r in 1 to $noRel where $d = $firstDay[[$r]]
              return -$resCostPerRel[[$r]]
            else 0
      ]
      let $cashFlow := [
        for $d in 1 to $timeHorizon
        return $laborCashFlow[[$d]] + $resCashFlow[[$d]]
      ]
      let $constraints := $releaseSizeCannotExceedCapacity and
                          $capacityIsZeroWhenSizeIsZero

      return {
        constraints : $constraints,
        metrics: {
          cashFlow: $cashFlow
        },
        interface: {
          firstDay: $firstDay,
          lastDay : $lastDay
        }
      }
    }; (: end of SofDevModel function :)
```

CommonOperations.jq

```
jsoniq version "1.0";

module namespace co =
  "http://kb.dgms.io.lib/softwareFeatureSelection/CommonOperations.jq";

(:-----  Operation: IfOneThenOne(lhs,rhs)                -----:)
(:-----   lhs --> rhs (implication)                      -----:)
(:----- If left hand side is 1 then right hand side is 1 -----:)
(:----- 0 <= $lhs, $rhs <=1 and $lrs, $rhs are integers  -----:)
declare function co:IfOneThenOne($lhs, $rhs) {
  let $result := (1-$rhs) <= (1-$lhs)
  return $result
};

(:-----  Operation: IfOneThenEqual(boolean,lhs,rhs)                    -----:)
(:-----     boolean=1 --> lhs = rhs     (implication)                  -----:)
(:----- If boolean is 1 then left hand side is equal to right hand side -----:)
(:----- $lrs, $rhs are integers                                         -----:)
declare function co:IfOneThenEqual($b, $lhs, $rhs) {
  let $M := 1000000000  (: M is a very large positive number :)
  let $result :=
    -$M * (1-$b) <= ($lhs - $rhs) and ($lhs - $rhs) <= $M * (1-$b)
  return $result
};

(:-----  Operation: IfZeroThenEqual(boolean,lhs,rhs)                    -----:
(:-----     boolean=0 --> lhs = rhs     (implication)                   -----:)
(:----- If boolean is 0 then left hand side is equal to right hand side -----:)
(:----- $lrs, $rhs are integers                                         -----:)
declare function co:IfZeroThenEqual($b, $lhs, $rhs) {
  let $M := 1000000000  (: M is a very large positive number :)
  let $result :=
    (-$M * $b) <= ($lhs - $rhs) and ($lhs - $rhs) <= ($M * $b)
  return $result
};

(:-----  Operation: ComputeItemThruMetrics($inpData,$svc) -----:)
(: ----  for every input or output for service $svc,      -----:)
(: ----  return the value of inputThruDv or outputThruDV  -----:)
(: ----  if the DV is defined, or zero otherwise          -----:)
declare function co:ComputeItemThruMetrics($inpData,$svc,$outputThru) {
    let $noRel := $inpData.RelSch.noReleases
    (:-- Calculate set of all inputs and outputs including subservices --:)
    let $subServicesInput :=
      for $ss in $inpData.services($svc).subService[]
      return $inpData.services($ss).input[]
    let $subServicesOutput :=
      for $ss in $inpData.services($svc).subService[]
      return $inpData.services($ss).output[]
    let $setAllIO := distinct-values({
                       $inpData.services($svc).input[],
                       $inpData.services($svc).output[],
                       $subServicesInput,
                       $subServicesOutput})
    (: -- Calculate inputItemThru for all elements in setAllIO   --:)
    let $inputItemThru := {|
      for $j in $setAllIO
      let $inputItemThruArray := [
        for $r in 1 to $noRel + 1
        let $normalizedInputItemThru :=
          if (exists($inpData.services($svc).inputThruDV[[$r]]($j)))
          then $inpData.services($svc).inputThruDV[[$r]]($j)
          else 0
        return $normalizedInputItemThru
      ]
      return {$j: $inputItemThruArray}
    |}
    (: -- Calculate outputItemThru for all elements in setAllIO   --:)
    let $outputItemThru := {|
      for $j in $setAllIO
      let $outputItemThruArray := [
        for $r in 1 to $noRel + 1
        let $normalizedOutputItemThru :=
          if (exists($outputThru($j)[[$r]]))
          then $outputThru($j)[[$r]]
          else 0
        return $normalizedOutputItemThru
      ]
      return {$j: $outputItemThruArray}
    |}
```

```
  return {
    inputItemThru:     $inputItemThru,
    outputItemThru:    $outputItemThru,
    setAllIO:          $setAllIO
  }
};

(:-----  Operation: ComputeItemThruSubMetrics($inpData,$svc)           -----:)
(: ----  for every input or output for service $svc,                   -----:)
(: ----  return the sum of inputThruDv or outputThruDV for subservices -----:)
(: ----  if the DV is defined or zero otherwise                        -----:)
declare function co:ComputeItemThruSubMetrics($subServiceMetrics,
                                              $svc,
                                              $subServices,
                                              $setAllIO,
                                              $noRel) {
  (: -- For all  in setAllIOc, calculate sum inputThru for all subservices --:)
    let $inputItemThruSub := {|
      for $j in $setAllIO
      let $inputItemThruSubArray := [
        for $r in 1 to $noRel + 1
        return sum(
          for $ss in $subServices
          return $subServiceMetrics($ss).metrics.inputItemThru($j)[[$r]]
        )
      ]
      return {$j: $inputItemThruSubArray}
    |}
  (: -- For all in setAllIOc, calculate sum outputThru for all subservices --:)
  let $outputItemThruSub := {|
    for $j in $setAllIO
    let $outputItemThruSubArray := [
      for $r in 1 to $noRel + 1
      return sum(
        for $ss in $subServices
        return $subServiceMetrics($ss).metrics.outputItemThru($j)[[$r]]
      )
    ]
    return {$j: $outputItemThruSubArray}
  |}
  return {
    inputItemThruSub:  $inputItemThruSub,
    outputItemThruSub: $outputItemThruSub
  }
};
```

APPENDIX 2 - DEFINITIONS

Capacity is the quantity of points that a particular Agile development team is estimated to be able to accomplish in a particular timebox.

Decision Guidance System is an advance class of Decision Support Systems that are designed to provide actionable recommendations, typically based on formal analytical models and techniques.

DGAL, Decision Guidance Analytics Language, is a language used by Unity for the definition and manipulation of analytical modules.

Effort point, usually called point in Agile projects, is a metric that estimates the relative size of software functionality. The metric takes into consideration the size, effort and complexity of the software.

Feature is a slice of business functionality that is meaningful to the customer or user [35] and consequently delivers value to the business when released. In the context of this dissertation, a feature is small enough to fit in one single release.

Feature-driven project is a project where each release deploys a set of features according to a prioritization that takes into consideration the business value of each feature.

FLWOR is a XQuery and JSONiq programming construct that correspond to SQL's SELECT-FROM-WHERE statements, but are more general and more flexible.

Full time equivalent is the equivalent of one full time worker.

JSON is a simple language for expressing complex, semi-structured data.

JSONiq is a language for the manipulation of JSON structures

Performance Model formally describes feasibility constraints and metrics of interest, such as cost, as a function of fixed and control parameters.

Product Backlog is a prioritized inventory of yet-to-be-worked-on product backlog items [35].

Product Owner is the central point of product leadership, responsible for defining what to do and in what order [35].

Release is a container of features that are deployed as a single unit, that is, once deployed, all the features in a release are available to the customer. A release has a certain size capacity, determined by its duration, the size and productivity of the developers' team. Following Agile principles, Releases, like Sprints, have a fixed duration, called a timebox.

Release Plan specifies the set of features that are expected to be developed in each release. The output of release planning.

Release Schedule is the same as Release Plan

Scrum is a term borrowed from the sport of rugby. It is a lightweight, Agile, iterative and incremental approach to developing products and managing work [35].

Service Network, as defined in [11], is a network of service-oriented components that are linked together to produce products. A service is a synonym for a process. Services can be composite, that is, have subservices or atomic, that is, indivisible.

Sprint is a short-duration, timeboxed iteration, typically between one and four weeks, "during which the Scrum team is focused on producing a potentially shippable product increment" [35].

Stakeholder is a “person, group, or organization that affects or can be affected by an organization’s actions” [35].

Timebox “is a fixed-length period of time during which an activity is performed” [16]. In Scrum, both sprints and releases are timeboxed.

Unity is a platform for building DGSs from reusable analytics models.

REFERENCES

[1] T. AlBourae, G. Ruhe, and M. Moussavi, “Lightweight Replanning of Software Product Releases”, in *2006 International Workshop on Software Product Management (IWSPM’06 - RE’06 Workshop)*, 2006, pp. 27–34.

[2] D. Ameller, C. Farré, X. Franch, A. Cassarino, D. Valerio, and V. Elvassore. "Replan: A Release Planning Tool", *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER),* 516–520, 2017.

[3] F. B. Aydemir, F. Dalpiaz, S. Brinkkemper, P. Giorgini, and J. Mylopoulos. "The Next Release Problem Revisited: A New Avenue for Goal Models", *IEEE 26th International Requirements Engineering Conference (RE),* 5–16, 2019.

[4] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley, “The next release problem”, *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, 2001.

[5] F. Boccanera and A. Brodsky, “Decision Guidance on Software Feature Selection to Maximize the Benefit to Organizational Processes”, in *22nd International Conference on Enterprise Information Systems (ICEIS)*, 2020, pp. 381-395.

[6] F. Boccanera and A. Brodsky. “*Opti-Soft*: Decision Guidance on Software Release Scheduling to Minimize the Cost of Business Processes”, in *Enterprise Information Systems: 22nd International Conference, ICEIS 2020*, Revised Selected Papers (2021). Springer, pp. 184-216.

[7] F. Boccanera and A. Brodsky. "*Opti-Soft+*: A Recommender and Sensitivity Analysis for Optimal Software Feature Selection and Release Planning", in *24th International Conference on Enterprise Information Systems (ICEIS)*, 2022

[8] B. Boehm. "Value-based software engineering", *GSOFT Software Engineering Notes*, Vol. 28, No. 2, pp. 1-12, March 2003

[9] A. Brodsky and J. Luo, "Decision Guidance Analytics Language (DGAL)-Toward Reusable Knowledge Base Centric Modeling", in *17th International Conference on Enterprise Information Systems (ICEIS),* Barcelona, Spain, 2015, pp. 67–78.

[10] A. Brodsky, "Decision Guidance Systems and Applications To Manufacturing, Power Grid, Supply Chain and IoT", in *20th International Conference on Enterprise Information Systems (ICEIS)*, Funchal, Portugal, 2018, p. 92.

[11] A. Brodsky, M. Krishnamoorthy, M. O. Nachawati, W. Z. Bernstein, and D. A. Menascé, "Manufacturing and contract service networks: Composition, optimization and tradeoff analysis based on a reusable repository of performance models", in *IEEE International Conference on Big Data (Big Data),* 2017, pp. 1716–1725.

[12] J. Cleland-Huang, and M. Denne. "Financially informed requirements prioritization", in *27th International Conference on Software Engineering*, 2005. ICSE 2005.

[13] M. Cohn, "Agile Estimating and Planning", *Prentice Hall*, 2005.

[14] M. Denne and J. Cleland-Huang, "Software by Numbers: Low-Risk, High-Return Development", *Prentice Hall,* 2003.

[15] M. Denne and J. Cleland-Huang, "The incremental funding method: data-driven software development", *IEEE Software,* vol. 21, no. 3, pp. 39–47, May 2004.

[16] S. Devaraj and R. Kohli, “The IT Payoff: Measuring the Business Value of Information Technology Investments”, *FT Press*, 2002.

[17] A. Elsaid, R. Salem, and H. Abdelkader "Proposed framework for planning software releases using fuzzy rule-based system" in *IET Software*, 13(6), 543–554, 2019

[18] G. J. Fell, “Decoding the IT value problem: an executive guide for achieving optimal ROI on critical IT investments”. *Wiley CIO Series*, Hoboken, New Jersey, 2013.

[19] M. Fowler and et al., “The Agile Manifesto”, *Software Development Magazine*, August 2001.

[20] X. Franch, and G. Ruhe. "Software Release Planning", *IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C),* 894–895, 2016.

[21] G. Fridgen, J. Klier, M. Beer, and T. Wolf, “Improving Business Value Assurance in Large-Scale IT Projects—A Quantitative Method Based on Founded Requirements Assessment”, *ACM Transactions on Management Information Systems*, vol. 5, no. 3, pp. 12:1–12:17, Aug. 2014.

[22] J. E. Hannay, H. C. Benestad, and K. Strand, “Benefit points—the best part of the story”, *IEEE Software*, Vol 34, Issue 3, 73-85, May 2017.

[23] Z. Irani and P. Love, “Evaluating information systems: public and private sector”, *Elsevier/Butterworth-Heinemann*, 2008.

[24] J. M. Keen, “Making Technology Investments Profitable: ROI Road Map from Business Case to Value Realization”, 2 edition. Hoboken, N.J: *Wiley*, 2011.

[25] C. Lin, G. Pervan, D. McDermid. "IS/IT Investment Evaluation and Benefits Realization Issues in Australia", *Journal of Research & Practice in Information Technology*, August 2005

[26] K. Marner, S. Wagner, and G. Ruhe. "Stakeholder identification for a structured release planning approach in the automotive domain", *Requirements Engineering*, 27(2), 211–230, 2022.

[27] S. Maurice, G. Ruhe, O. Saliu, and A. Ngo-The, "Decision Support for Value-Based Software Release Planning", in *Value-Based Software Engineering, Springer*, Berlin, Heidelberg, 2006, pp. 247–261.

[28] D. Mougouei. "Factoring requirement dependencies in software requirement selection using graphs and integer programming", *31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 884–887, 2016.

[29] M. O. Nachawati, A. Brodsky, and J. Luo, "Unity: A NoSQL-based Platform for Building Decision Guidance Systems from Reusable Analytics Models", *George Mason University, Technical Report* GMU-CS-TR-2016-4, 2016.

[30] M. Nachawati, A. Brodsky, and J. Luo. "Unity Decision Guidance Management System: Analytics Engine and Reusable Model Repository", in *19th International Conference on Enterprise Information Systems (ICEIS)*, 312–323, 2017.

[31] M. Nayebi, and G. Ruhe. "Asymmetric Release Planning: Compromising Satisfaction against Dissatisfaction", *IEEE Transactions on Software Engineering*, 45(9), 839–857, 2019.

[32] O. Oni and E. Letier. "Analyzing Uncertainty in Release Planning: A Method and Experiment for Fixed-Date Release Cycles", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2021.

[33] J. Pucciarelli and D. Wiklund, "Improving IT Project Outcomes by Systematically Managing and Hedging Risk," *IDC Report*, 2009.

[34] N. Riegel, and J. Doerr. "An Analysis of Priority-Based Decision Heuristics for Optimizing Elicitation Efficiency", in *Requirements Engineering: Foundation for Software Quality* (pp. 268–284), Springer International Publishing, 2014.

[35] K. S. Rubin, "Essential Scrum: a practical guide to the most popular agile process", Upper Saddle River, NJ: *Addison-Wesley*, 2012.

[36] G. Ruhe and A. Ngo The, "Hybrid Intelligence in Software Release Planning," *International Journal of Hybrid Intelligent Systems*, vol. 1, no. 1–2, pp. 99–110, Sep. 2004.

[37] T. Şahin, T. Huth, J. Axmann, and T. Vietor. "A Methodology for Value-oriented Strategic Release Planning to Provide Continuous Product Upgrading", *2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 1032–1036, 2020.

[38] N. Salleh, F. Mendes, E. Mendes. "A Systematic Mapping Study of Value-based Software Engineering", *45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA),* 2019

[39] R.S. Sangwan, A. Negahban, R.L. Nord, and I. Ozkaya. "Optimization of Software Release Planning Considering Architectural Dependencies, Cost, and Value", *IEEE Transactions on Software Engineering,* 48(4), 1369–1384, 2022.

[40] X. Song and N. Letch, “Research on IT/IS evaluation: a 25 year review”, *Electronic Journal of Information Systems Evaluation*, vol. 15, no. 3, pp. 276–287, 2012.

[41] The Standish Group, “CHAOS 2020.” 2020.

[42] The Standish Group, “CHAOS Manifesto 2011.” 2011.

[43] M. van den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal, "Determination of the Next Release of a Software Product: An Approach using Integer Linear Programming." in *CAiSE Short Paper Proceedings*, 2005.

[44] M. van den Akker, S. Brinkkemper, G. Diepen and J. Versendaal. "Software product release planning through optimization and what-if analysis" in *Information and Software Technology* (pp. 101-111), 2008

[45] W. Van Grembergen, “*Information Technology Evaluation Methods and Management”,* Hershey, Pa, *IGI Global,* 2001.

[46] S. Walter, T. Spitta. "Approaches to the ex-ante evaluation of investments into information systems" in *Wirtschaftsinformatik* 46, 171–180, 2004.

[47] K. Wnuk, T. Gorschek, D. Callele, E.-A. Karlsson, E. Åhlin, and B. Regnell. "Supporting Scope Tracking and Visualization for Very Large-Scale Requirements Engineering-Utilizing FSC+, Decision Patterns, and Atomic Decision Visualizations", *IEEE Transactions on Software Engineering,* 42(1), 47–74, 2016.

PUBLISHED PAPERS

F. Boccanera and A. Brodsky, "OptiHealth: A Recommender Framework for Pareto Optimal Health Insurance Plans," in *19th International Conference on Enterprise Information Systems (ICEIS)*, Porto, Portugal, 2017, pp. 599–609.

F. Boccanera and A. Brodsky, "Decision Guidance on Software Feature Selection to Maximize the Benefit to Organizational Processes", in *22nd International Conference on Enterprise Information Systems (ICEIS),* 2020, pp. 381-395.

F. Boccanera and A. Brodsky, "*Opti-Soft:* Decision Guidance on Software Release Scheduling to Minimize the Cost of Business Processes", in *Enterprise Information Systems: 22nd International Conference, ICEIS 2020, Revised Selected Papers (2021). Springer*, pp. 184-216.

F. Boccanera and A. Brodsky, *Opti-Soft+:* A Recommender and Sensitivity Analysis for Optimal Software Feature Selection and Release Planning. *in 24th International Conference on Enterprise Information Systems (ICEIS),* 2022.

BIOGRAPHY

Fernando Boccanera received his Bachelor of Computer Science from Campinas State University, Brazil in 1978 and his Master of Software Engineering from George Mason University in 1998. He has worked his entire career in organizations that develop software or market tools for software development, where he held positions as programmer, systems administrator, technical support, data base administrator, IT project manager and IT manager. Since 2015 Mr. Boccanera has worked for the United States Patent and Trademark Office, an agency of 12 thousand employees that issues patents and trademarks, which collectively drive innovations and have a major impact in the US economy. USPTO, an agency of the Department of Commerce, is regarded as the premier Intellectual Property organization in the world and is recognized within the Federal Government as a model and innovative IT office.