Department of Computer Science
George Mason University Technical Reports

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/ 703-993-1530

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

Fernando Boccanera
fboccane@gmu.edu

Alexander Brodsky
brodsky@gmu.edu

Technical Report GMU-CS-TR-2022-1

ABSTRACT

Many approaches have been developed to increase the return on a software investment, but each one has drawbacks. Proposed in this paper is the *Opti-Soft*+ framework that addresses this problem by producing a software release schedule that maximizes the business value of investments in information systems that automate business processes. The optimal release schedule is the result of solving a mixed integer linear programming problem. *Opti-Soft*+ includes a formal optimization model, a Decision Guidance System that implements the model and a methodology. *Opti-Soft*+ is an extension of the *Opti-Soft* framework proposed earlier with (1) a refined cost model, (2) a technique for sensitivity analysis of the normalized cost per unit of production, and (3) an atomic business process model that is driven by output throughputs in addition to input throughputs.

1 INTRODUCTION

Software development projects that are successful and return to the business a value that justify their investments are not common. According to (The Standish Group, 2018) only 36% of projects are successful. To improve the rate of success, organizations have been using Agile methods. As reported in (Serrador & Pinto, 2015,) Agile does have a statistically significant impact on three dimensions of project success, but adopting Agile is not a guarantee of a return on the investment.

Because the Agile software development lifecycle is based on short iterations, at each iteration the team needs to decide what functionality to include. This process, called Release Planning, provides an opportunity to improve the business value of the software because different functionalities result in different value profiles.

Several release planning approaches have been developed to maximize the business value of software delivery. The highly influential Incremental Funding Methodology (IFM) by (M. Denne & Cleland-Huang, 2004) uses heuristics to select a release schedule that maximizes the business value of software investments. F-EVOLVE*'s approach (Maurice et al., 2006) is to involve stakeholders iteratively to achieve releases that result in the highest degree of satisfaction. A third approach by (Van den Akker et al., 2005) applies integer linear programming to maximize the revenue.

The IFM, F-EVOLVE* and Van den Akker approaches use cash flow as a proxy for business value. They all require the estimation of cash flows for each software feature and that's very challenging due to the difficulty of drawing a direct correlation between a particular business benefit, like a reduction in cost, and a specific piece of software. (Devaraj & Kohli, 2002) have acknowledged this difficulty of isolating the effect of IT on firm performance.

The main pitfall of the existing approaches is that each and every dollar of cash flow needs to be allocated to one and only one feature which is not a realistic assumption, because often, realizing a business benefit requires the implementation of multiple software features. Another pitfall is that each cash flow estimate combines the business benefit with the software development cost, which means that all the estimations have to be done externally, which is typically difficult and often inaccurate.

In (Boccanera & Brodsky, 2020) and (Boccanera & Brodsky, 2021), we proposed a new approach, called *Opti-Soft,* to address the pitfalls of existing methods for a class of software projects that automate a Business Process Network (BPN). *Opti-Soft* is a decision guidance framework for release planning that maximizes the business value as measured by the Net Present Value (NPV), based on a model of the underlying business process and savings achieved due to the combined effect of new software features on improved business process efficiency.

However, the *Opti-Soft* approach has limitations, as we discovered looking at a number of real software project examples. First, the cost model was based on labor costs only, whereas realistic cost models may be considerably more involved. Second, for stakeholders to have a high confidence on the recommendations on software feature selection and release planning, they often need to know the sensitivity of the recommendations to assumptions on demand on business process throughput, e.g., the number of daily patent applications to be processed by the Patent Office. This question is important to release planners because, potentially, a small change in expected business process throughput may lead to unexpected changes in recommended features and release schedules. However, addressing sensitivity was not part of *Opti-Soft.* Third, while the business processes can be hierarchically composed, *Opti-Soft* only supported a limited atomic (leaf) process in the hierarchy in which the cost is driven by input throughput (e.g., number of patent applications that need to be processed per day). Whereas, atomic processes driven by output throughput were not supported.

Lifting these limitations is exactly the focus of this paper. More specifically, the contributions of this paper are as follows. First, we extend the cost model, of both BPN and software development, beyond labor cost to include a range of variable and fixed costs (i.e., of resources required). The extended cost structure includes: 1) non-labor costs incurred by each input flow through the BPN, 2) fixed costs associated with processes in the BPN, and 3) fixed costs associated with software development. Extending *Opti-Soft* to support these additional costs is not trivial because sometimes a fixed cost can be incurred by more than one software feature, that is, the relationship is not one-to-one.

Second, we develop a technique for sensitivity analysis of the normalized cost per unit of production, for a recommended release plan and associated improved BPN, as a function of BPN throughput. The analysis involves fixing some of the decision variables while allowing others to be instantiated by the optimizer. The idea is to determine the delta change in the objective function for a one-unit change of the required BPN throughput.

Third, we develop an atomic service model that is driven by output throughputs in addition to the model driven input throughputs. We add an indivisible atomic service, whose throughput is driven by the number of outputs that it needs to produce. Each output is associated with a number of input flows (e.g, applications) and the cost is based on the outputs produced, for example, a report.

Opti-Soft+ is the result of these extensions. It takes advantage of the fact that the implementation of software features leads to more efficient business processes 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 key idea is that, because the improved business efficiency is a direct consequence of the availability of software features, this relationship can be formally modelled using mixed integer linear programming (MILP) constraint formulation, which allows the use of MILP solvers to find optimal release plan.

The uniqueness of *Opti-Soft* framework is its accurate estimate of business value improvement by formally modeling BPN and associated costs over the investment time horizon, as a function of chosen software features and release plan. Also, *Opti-Soft*+ removes the limitation of existing approaches that force every investment dollar to be assigned to one and only one software feature. The *Opti-Soft*+ model allows one software feature to impact multiple processes and allows a process to be impacted by multiple features, on a many-to-many relationship. In *Opti-Soft*+, the estimation of the business value of software features is not external to the approach, but instead, is at the heart of the cost model. The *Opti-Soft*+ framework is composed of a methodology, a formal optimization model and a Decision Guidance System (DGS) which implements the formal model and produces an optimal recommendation.

This paper is organized as follows: Section 2 provides an overview of the *Opti-Soft*+ model, including the cost approach, the BSN modelling and release planning. Section 3 briefly describes the optimization model; Section 4 describes the DGS and the methodology; Section 5 provides an example of the extensions; Section 6 conducts a sensitivity analysis and Section 7 provides concluding remarks. Appendix 1 shows the entirety of the formal model.

2 *OPTI-SOFT*+ MODEL OVERVIEW

In order to maximize business value associated with new software features, we need to estimate the cost of software development the benefit of the implementation of the software. For software that implements information systems to automate a business process, the benefit of the software is the cost savings in the business process due to automation.

A business process, formally modelled in Section 3, consumes input flows, e.g., patent applications, and produces output flows, e.g., patent grants. The cost associated with the business process is a function of the process fixed and control parameters, including labor rates and time spent. This means that the benefit (savings) of a software feature that automates a business process can be determined by subtracting the cost of the automated process from the cost of the process before automation.

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

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, resulting in a Net Present Cost $NPC(BPN_1)$, 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 $NPC(BPN_r)$.

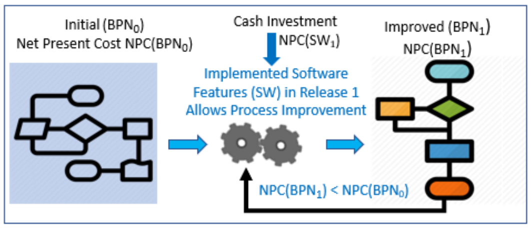

Initial (BPN_0)
Net Present Cost NPC(BPN_0)
Cash Investment
NPC(SW_1)
Improved (BPN_1)
NPC(BPN_1)
Implemented Software Features (SW) in Release 1 Allows Process Improvement
NPC(BPN_1) < NPC(BPN_0)

Fig. 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 enabling software development features.

BSN Modelling

To intuitively understand BPN modelling, consider the example depicted in Figure 2. It shows a parent process P composed of subprocesses A, B and C, all of which must be executed. Note that the output from A serves as input to B and the output from B serves as input to A. Subprocess A has three alternatives, AA, AB and AC, whereas only one of them must be executed. Similarly, B has alternatives BA and BB, and C has alternatives CA and CB. By choosing among the alternatives for each subprocess, a new configuration of P is established.

Note that a valid configuration for P requires one and only one of each of its three subprocesses A, B and C, which establishes an AND relationship between process P and its subprocesses A, B and C. The relationship between A and its alternatives (subprocesses) AA, AB and AC is an OR because either AA or AB or AC can be present in P. B and C also have OR relationships with their subprocesses.

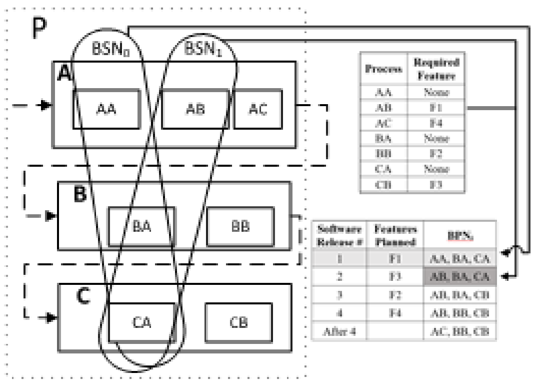
We model the BPN as a Service Network (SN) (Brodsky et al., 2017) which is a "network of service-oriented components that are linked together to produce products". The linkage among service components is through inputs and outputs. In Figure 2, P is a composite (parent) service because it is composed of subservices A, B and C. A, B and C are also composites while all the other subservices are atomic, that is, indivisible.

BSN Transition and Release Planning

The transition from a subprocess alternative to another requires the implementation of specific software functionality called features. For example, subprocess alternative AB requires feature F1.

We assume that features are implemented in iterations called releases. At the beginning of each release, the team decides which features to include in the scope. This is called release planning. Note that the implementation of features results in automation of certain aspects of the original business process, allowing it to transition to a more efficient process alternative that results in labor and other savings.

In the example of Figure 2, we assume that AA, BA and CA are manual processes, and the initial BPN configuration (BPN_0) is AA, BA, CA with NPC(BPN_0). The top table on the right shows the required features for each process while the table on the bottom shows the BPN configuration after each release. Note that A's alternative subprocess AB is more cost effective than AA and it requires feature F1. Because F1 is implemented in release 1, after release 1 is completed, BPN_0 transitions to BPN_1 which is configured with AB, BA, CA, with the cost of NPC(BPN_1). Note that F1 'activates' AB and this activation property, which is unique to *Opti-Soft+*, is used extensively in the formalization of the Mixed Integer Linear Programming problem, described in section 3. At the end of each release, the availability of implemented software features allows the activation of alternative processes that are more cost effective, reducing the overall NPC of the SN. Subprocess AA transitions to AB then to AC, BA transitions to BB and CA transitions to CB. The final, optimal SN configuration is then AC, BB, CB. Note that Fig 2 shows not only the BSN transitions, but also the release plan, that is, the software features implemented in each release.



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

Software Release #	Features Planned	BPN
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

Fig. 2 - Example of BPN Transition as a result of feature delivery.

Cost Model

The NPC over the investment time horizon is the combined NPC of all the BPNs plus the NPC of software development in all releases. Costs are accrued daily and are paid on a set schedule. The NPC is composed of five types of cost:

1. Variable labor costs of the SN
2. Variable non-labor costs of the SN
3. Fixed non-labor costs of the SN
4. Variable labor costs of software features
5. Fixed non-labor costs of software features

Note that in the *Opti-Soft+* model, we use the Net Present Value (NPV), which is simply NPC with the negative sign.

Variable Labor Costs of the SN

Each process of the SN 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. The cost of a process, or *LaborCostPerDay*, is the labor rate times the duration to handle all inputs and outputs in a day.

Variable Non-labor Costs of the SN

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

Parameters *CostPerInput* and *CostPerOutput* capture the non-labor costs for each input and output. These parameters are used to compute *FlowCostPerDay*.

Fixed Non-labor Costs of the SN

Fixed non-labor costs are not driven by inputs or outputs, instead they are driven by the services. An example of a fixed cost associated with a particular service is rent. Parameter *ServiceCostPerDay* captures the daily cost for each atomic service and is used to calculate the *ServiceCostPerDay*.

Variable Labor Costs of Software Development

Opti-Soft+ follows an Agile practice called feature-driven, where release planning is done at the feature level, that is, features are removed from the product backlog and assigned to releases. The size of features is estimated in points, which is a unit based on the perceived effort to develop the feature. The release size, that is, the sum of the points for all features in the release, cannot exceed the capacity of the team, which is the average productivity of a developer times the size of the team. Development labor cost, in turn, is computed by multiplying the team's capacity by the developer cost per effort point. The formal model captures the software cost in *SWCostPerDay* and then uses a pay schedule to calculate the *LaborCashFlow*.

Fixed Non-labor Costs of Software Development

Fixed costs associated with features are experienced during software development, where features are produced. They are incurred by resources such as a hardware server, a software tool, etc…

Every feature requires 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.

The cost of resources is captured in *ResCashFlow*, whose computation uses the following parameters:

- ***ResSet*** is the set of all non-labor resources
- ***FeatureRes*** maps features to resources
- ***ResCost*** maps a resource to its cost

Computation of the SN Cash Flow

The *CostPerDay* of each atomic process is the sum of *LaborCostPerDay*, *FlowCostPerDay* and *ServiceCostPerDay*. *CostPerDay* is used to calculate the schedule of payments, or *CashFlow(d)* for each *d* in the time horizon. The *CashFlow(d)* for each subprocess of a parent process is aggregated and then rolled up to determine the *CashFlow(d)* of the entire SN.

Computation of the Software Cash Flow

The *CashFlow(d)* of the development of software is the sum of *LaborCashFlow* and *ResCashFlow.*

Computation of NPV

The *CashFlow(d)* for the SN and for the Software Development are combined and discounted to produce the *TimeWindowNPV*.

3 *OPTI-SOFT*+ OPTIMIZATION MODEL

Opti-Soft+ produces an optimal release schedule and SN configuration by solving a maximization problem given a set of parameters like the services in the SN, feature sizes, number of releases, time horizon, labor rates, size of the development team, etc... It maximizes the NPV of the total cost of the service network plus the software development cost over the investment horizon, subject to constraints such as the space of process alternatives. *Opti-Soft*+'s formal model with its parameters, computations, constraints and maximization formulation is presented in its entirety in Appendix 1 and here we explain some aspects of it.

The Decision Variables in the formal model are:

1. *On(s,r)* – a Boolean indicating whether service *s* is activated in the SN configuration during the development of release *r*.
2. *IBF(r,f)* - a boolean indicating whether business feature *f* is implemented in release *r.*
3. *ITF(r,f)* - a boolean indicating whether technical feature *f* is implemented in release *r.*
4. *InputThru(s,i,r)* – a real, indication the number of inputs of type *i* that go through atomic service *s* during the development of release *r.*

The formal model is broken down in seven hierarchical components, each with its own set of Parameters, Decision Variables (DVs), Computation, Constraints and Interface Metrics (results exposed to the other components). The components 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 parameters describe the features, their dependencies, sizes, time horizon, number of releases and release duration. The DVs 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 has parameters id, type, set of inputs and set of outputs. The DV is *On(s,r)*, a boolean indicating whether the service is activated.
4. **ANDservice** describes 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 set of subservices.
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 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 DV is the required throughput, *InputThru(s,i,r)* and the service 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 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 DV is the required throughput, InputThru(s,i,r) and the service 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 parameters are the size of the team, productivity, the cost per unit, etc… It uses feature point as a unit of cost.

The formulation of the optimization is of a Mixed-Integer Linear Programming (MILP) problem, because 1) three of the DVs (*On(s,r), IBF(r,f), ITF(r,f)*) are Boolean, 2) one DV (*InputThru(s,i,r))* is real, and 3) the objective function is linear because it is the result of the addition of various cost parameters which themselves are linear. Section A9 of Appendix 1 describes the MIPL formulation, which is summarized below:

$$\begin{aligned}&\textbf{\textit{Given Parameters}}\\ &\textbf{\textit{Max}}\quad \textbf{\textit{ReleaseScheduling.TimeWindowNPV}}\\ &\qquad \textbf{\textit{s.t.}}\quad \textbf{\textit{ReleaseScheduling.Constraints}}\end{aligned}$$

The constraints are the space of SN alternative configurations, the required software features, the interdependencies among features, the capacity of the development team, etc… Each of the six formal components shown in Appendix 1 implements constraints that are then aggregated under *ReleaseScheduling.Constraints.*

Note that the *CashFlow* and *TimeWindowNPV* produced by the formal model are negative numbers consequently maximizing the NPV results in minimizing the cost.

Opti-Soft+ includes a Decision Guidance System (DGS) which implements the formal model in Appendix 1 and includes a MILP Solver. The DGS produces:

1. Optimal NPV of the business benefit
2. A release schedule, which is the result of the Solver instantiating DVs *IBF(r,f)* and *ITF(r,f)*.
3. The optimal service network configuration at the end of each release, which is the result of the Solver instantiating DV *On(s,r)*.

4 *Opti-Soft+* Methodology and Decision guidance System

The optimization model described in the previous section is implemented within a Decision Guidance System (DGS), which uses the *Parameters* in the input file to maximize the NPV, subject to the *Constraints*. During the maximization, the DGS performs the *Computation* and chooses the optimal *DecisionVariables*. The *Opti-Soft+* DGS is implemented using Unity (Nachawati, M. O., Brodsky, A., & Luo, J., 2016), (Nachawati, M. O., Brodsky, A., & Luo, J., 2017), a platform for building DGSs from reusable Analytical Models (AMs). Unity exposes an algebra of operators and provides an unified, high-level language called Decision Guidance Analytics Language (DGAL) (Brodsky, Alexander, & Luo, J., 2015).

The Opti-Soft+ framework is composed of the optimization model, the DGS and a methodology. We covered the first two so now we cover the latter. The *Opti-Soft+* methodology, which extends the methodology in (Boccanera & Brodsky, 2021), contains the following steps:

1. Generate candidate software features to be implemented
2. Capture the As-Is BPN configuration, and alternative BPN configurations that can be enabled by candidate software features
3. Gather and instantiate input parameters for the optimization model as described in Appendix 1, including the set of features, their dependencies, time horizon, number of releases, BPN services
4. Compute the baseline NPV for the As-Is BPN
5. Perform *Opti-Soft+* DGS optimization to come up with a recommended Release Plan (including chosen software features in each release) and the associated optimal BPN configuration (To-Be)
6. Calculate the savings, which is the NPV of the To-Be minus the NPV of the As-Is
7. Per recommendation in the previous step, during Release 1:
   a. Operate the BPN according to the optimal BPN configuration.
   b. Implement recommended software features
8. For each release r = 2,…,n
   a. Update existing software features to include those implemented in the previous release
   b. For updated software features and refined demand/throughput requirements, run operational optimization to find the best BPN configuration. Operate the BPN according to it.
   c. Repeat Opti-Soft+ DGS optimization for updated existing features and demand to update recommended Release Plan for the remaining releases (starting from Release r + 1)
   d. Implement recommended software features

5 *OPTI-SOFT+* PRODUCES EXAMPLE OF EXTENSIONS

Sections 5 of (Boccanera & Brodsky, 2021) describe an example of a service network composed of 3 parent processes (A, B and C). The optimal release plan and SN configuration, is reproduced in Table 1. The example has 4 releases, each lasting 60 days and a time horizon of 520 days, and a BSN that requires processing 100 user applications per day, that is, for demand=100. In the example, the optimized objective function, or NPV, produced by the DGS is -$6,411,432.73.

Table 1-Optimal release schedule and SN configuration.

Software Release #	**Features implemented**	**Optimal SN Configuratio n**
1	TF1, BF1	AA, BA, CA
2	BF3	AB, BA, CA
3	BF2	AB, BA, CB
4	BF4	AB, BB, CB
After 4		AC, BB, CB

We now take the example from (Boccanera & Brodsky, 2021) and add the extensions described in section 1. Table 2 shows the extended parameters.

Table 2-Extended Parameters

Forma-lization	Parameter	Value
RSch	ResSet	"softwareLicense1"
	ResCost	"softwareLicense1":20,000
	FeatureRes	TF1:"" BF1:"" BF2:"" BF3:"" BF4: "softwareLicense1"
Input Driven Atomic Service	ServiceCost PerDay	AA:200 AB:200 AC:200 BA:200 BB:200 CA:200 CB:200
	CostPerInput	AA.UserApplication:2 AB.UserApplication:0 AC.UserApplication:0 BA.CompliantApplic:0 BB.CompliantApplic:0 CA.AdjudicatedApplic:0 CB.AdjudicatedApplic:0
	CostPerOutput	AA.CompliantApplic:3 AA.NonComplianceNtc:1 AB.CompliantApplic:0 AB.NonComplianceNtc:0 AC.CompliantApplic:0 AC.NonComplianceNtc:0 BA.AdjudicatedApplic:0 BB.AdjudicatedApplic:0 CA.AdjudApplicLetter:0 CB.AdjudApplicLetter:0

There is a software license that costs $20,000 when feature BF4 is implemented. There are fixed costs per day of $200 for each of the atomic processes (AA, AB, AC, BA, BB, CA, CB). Atomic process AA incurs $2 in cost per "User Application" input, $3 per 'CompliantApplication" output and $1 per "NonCompliantNotice" output.

Using the parameters in Table 2, plus the parameters in Section 5 of (Boccanera & Brodsky, 2021), the DGS maximizes the objective function and produces an optimal NPV of -$6,748,777.45. The increase from -$6,411,432.73 is expected and is a direct result of the extended costs listed in Table 2.

In order to determine the savings of DGS' recommendation, we need to compare the NPV of the extended example (-$6,748,777.45), called the To-Be, with the NPV of the As-Is, which is the BPN prior to the development of the software.

To calculate the As-Is recommendation, we change the example parameters as follows: 1) set to zero the parameters used in the Software Development Formal Model and 2) Set the BPN configuration to AA, BA, CA for the entire duration of the time horizon of the investment. Running the DGS with these modified parameters, the resulting NPV for the As-Is is -$9,611,947.49.

The savings is the difference between the To-Be (-$6,748,777.45) and the As-Is (-$9,611,947.49), or $2,863,170.04. Note that this is the maximum savings, i.e., there is no other release plan and BSN configuration that produces a higher savings.

6 OPTISOFT+ SENSITIVITY ANALYSIS

One aspect that a decision-maker would be interested in, is how sensitive the total NPC is to certain changes in parameters. 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 will show how the unit cost varies with for each δ, while the second will show the unit cost variation and the stability of the BSN configuration.

Sensitivity Analysis 1

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 3. In step 6, we calculate UC(δ_i), also shown in Table 3. In step 6 we plot the chart shown in Fig 3.

Table 3 – 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

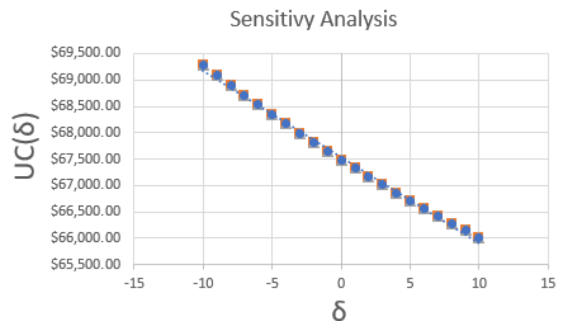


Fig. 3 – Plot of δ and UC(δ)

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.

Sensitivity Analysis 2

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.

7 Conclusion and Future Work

In this paper we introduced *Opti-Soft+*, an extended framework to produce a software release schedule that maximizes the business value of investments in the development of software applications that automate business processes. *Opti-Soft+*employs a realistic cost approach, and models the MILP optimization problem formally, which is implemented by a Decision Guidance System. We also conducted a sensitivity analysis that helps a decision maker to understands the range of parameters that the solution would hold.

The contributions of this paper are: 1) extending the cost model, of both BPN and software development, beyond labor cost to include a range of variable and fixed costs (i.e., of resources required), 2) developing a technique for sensitivity analysis of the normalized cost per unit of production, for a recommended release plan and associated improved BPN, as a function of BPN throughput, and 3) developing an atomic service model that is driven by output throughputs in addition to the model driven input throughputs..

The benefits of the above contributions are: 1) making the cost model more realistic and allowing a cost to be incurred my multiple features, 2) providing a decision maker with analytical results showing how sensitive the recommendation is to certain changes in parameters, and 3) allowing a natural way to model process that are output driven or that are driven by both input and output, which increases the practicality of the framework.

Potential future work involve comparing *Opti-Soft+* with other frameworks such as the popular Incremental Funding Methodology (Cleland-Huang & Denne,2005) and conducting a case study.

APPENDIX 1: FORMAL MODEL WITH EXTENSIONS

A1. 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 4.2
- ***SWD.Parameters*** is defined in section 4.7

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 A.2.
- $\boldsymbol{SWD.DecisionVariables}$ is defined in section A.8.

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:
$$\begin{aligned} &CombinedCashFlow(d) \\ &\quad = BSN.IM.Cashflow(d) \\ &\quad + SWD.IM.CashFlow(d) \end{aligned}$$

where:

- ***BSN.IM.CashFlow*** is defined in section *BSN.InterfaceMetrics* of section A.2
- ***SWD.IM.CashFlow*** is defined in section *Software.InterfaceMetrics* of section A.8.

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: $(\forall\ i, j, \in [1..NR], i \neq j), IF(i) \cap IF(j) = \emptyset$
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 A.2.

4. ***SWD.Constraints*** is defined in section A.8.

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 A.2
- ***SWD.InterfaceMetrics*** is defined in section A.8

A2. 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
- $\boldsymbol{Rate}: LR \rightarrow \mathbb{R}^+$ 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
  - $\boldsymbol{LaborPayDay}: [1..NLP] \rightarrow\ [1..TH]$ 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 hour.
- ***ServicesSet*** is the set of all services in the Service Network, defined in section A.3.
- ***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 A.3.

DecisionVariables is the set $\{s.DecisionVariables \mid s \in ServicesSet\}$. See section A.3.

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 A.3.

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 A.3.

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 A.3.
- $SWD.IM.firstDay$ and $SWD.IM.lastDay$ are defined in section A.8.

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

Constraints

1. $\boldsymbol{BSNDemandIsSatisfied}$(see Computation #2)

2. $\boldsymbol{BSNSupplyIsSatisfied}$ (see Computation #3)

3. $\boldsymbol{Service.IM(rootID, r).Constraints}$ (See section A.3)

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

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

A3. Service Formalization

ServicesSet formalization is a set of **Service**, where:
Service is a tuple ⟨*Parameters, DecisionVariables, Computation, Constraints, InterfaceMetrics*⟩, defined separately for each $ServiceType \in \{ANDservice, ORservice, InputDrivenAtomicService\}$

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 A.4.
- ***ORservice*** type is defined in section A.5.
- ***InputDrivenAtomicService*** type is defined in section A.6.
- ***OutputDrivenAtomicService*** type is defined in section A.7.

A4. ANDservice Formalization

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:

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

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:

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

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:

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

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:

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

Constraints are as follows:

1. ***AllSubservicesAreActivated*** (see computation #1)
2. ***TotalSupplyMatchesTotalDemand*** **(**see computation # 9)

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.

A5. 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 $\langle id, ServiceType(id), I(id), O(id), Subservices(id)\rangle$ 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 $\langle On(id), InputThru(id), OutputThru(id)\rangle$ 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:

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

Computation

1. *OnlyOneServiceIsActivated* constraint:

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

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

Constraints are as follows:

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

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.

A6. InputDrivenAtomicService Formalization

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:

- $\boldsymbol{id}$ 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: $(\forall\ i \in I(id))$, $(\forall\ o \in O(id))$, $IOthruRatio(id)(i, o)$ 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:

- $\boldsymbol{On}(id): [1..NR+1] \to \{0,1\}$ 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:

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

- $\boldsymbol{InputThru}(id): I(id) \times [1..NR+1] \to \mathbb{R}^+$ 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:
$$On(id, r) = 1 \to RBF(id) \subseteq RSch.IM.SoFarIBF(r)$$
$$\forall\ r \in [1..NR+1]$$
2. *DeactivatedServicesIsSatisfied* constraint:
$$\forall\ i \in I(id), \forall r \in [1..NR+1],$$
$$On(id, r) = 0 \to InputThru(id, i, r) = 0$$
3. Let $OuputThru(id): O(id) \times [1..NR+1] \to \mathbb{R}^+$ 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 $TimePerDay(id): [1..NR + 1] \times ServiceRoles(id) \to \mathbb{R}^+$ 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:

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

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:

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

7. Let $CostPerDay(id): [1..NR + 1] \to \mathbb{R}$ 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:

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

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.

A.7 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:

- $\boldsymbol{id}$ 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.

- $\boldsymbol{OIthruRatio}(id)\colon O(id) \times I(id) \rightarrow \mathbb{R}^{+}$ is a function described as follows: $(\forall\, o \in O(id))$, $(\forall\, i \in I(id))$, $OIOthruRatio(id)(i,o)$ also denoted as $OIthruRatio(id,i,o)$, gives for output o and input i, the ratio of input throughput based the output throughput.
- $\boldsymbol{RoleTimePerOI}(id)\colon ServiceRoles(id) \times (O(id) \cup I(id)) \rightarrow \mathbb{R}^{+}$ 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.
- $\boldsymbol{ServiceCostPerDay}\colon ServicesSet \rightarrow \mathbb{R}^{+}$ is a function described as follows: $(\forall\, s \in ServicesSet)$, $FixedCostPerDay(s)$ gives the non-labor fixed cost of service s for each day.
- $\boldsymbol{CostPerInput}(id)\colon I(id) \rightarrow \mathbb{R}^{+}$ 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*.
- $\boldsymbol{CostPerOutput}(id)\colon O(id) \rightarrow \mathbb{R}^{+}$ 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:

- $\boldsymbol{On}(id)\colon [1..NR+1] \rightarrow \{0,1\}$ 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:

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

- $\boldsymbol{OutputThru}(id)\colon O(id) \times [1..NR+1] \rightarrow \mathbb{R}^{+}$ 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:
$$On(id,r) = 1 \rightarrow RBF(id) \subseteq RSch.IM.SoFarIBF(r)$$
$$\forall\, r \in [1..NR+1]$$
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 $InputThru(id)\colon I(id) \times [1..NR+1] \rightarrow \mathbb{R}^{+}$ 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
$$\forall\, i \in I(id), \forall\, r \in [1..NR+1],$$
$$InputThru(id,i,r) = \sum_{o \in O(id)} (OIthruRatio(id,o,i) \times OutputThru(id,o,r))$$
4. Let $TimePerDay(id)\colon [1..NR+1] \times ServiceRoles(id) \rightarrow \mathbb{R}^{+}$ 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:
$$TimePerDay(id,l,r) = \sum_{j \in I(id)} (RoleTimePerOI(id,l,j) \times InputThru(id,j,r)) + \sum_{j \in O(id)} (RoleTimePerOI(id,l,j) \times OutputThru(id,j,r))$$
5. Let $LaborCostPerDay(id)\colon [1..NR+1] \rightarrow \mathbb{R}$ 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:

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

7. Let $CostPerDay(id): [1..NR+1] \to \mathbb{R}$ 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:

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

Constraints are as follows:

1. $\boldsymbol{FeatureDependencyIsSatisfied}$ (see computation #1)
2. $\boldsymbol{DeactivatedServicesIsSatisfied}$ (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.

A8. 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:

- $\boldsymbol{TS} : [1..NR] \to \mathbb{R}^+$ is a function that gives the team size, in full time equivalents, for each release.
- $\boldsymbol{DP} : [1..NR] \to \mathbb{R}^+$ 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).
- $\boldsymbol{SWPaySched}$, 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.
  - $SWPayDay: [1..NSP] \to [1..TH]$ 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 $RC : [1..NR] \to \mathbb{R}^+$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:

$$RC(r) = TS(r) \times DP(r) \times RSch.Parm.RD(r)$$

2. Let $RS : [1..NR] \to \mathbb{R}^+$ 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:

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

3. $ReleaseSizeCannotExceedCapacity$ constraint:

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

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:

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

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:

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

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

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:

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

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.,

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

9. Let $SWPayment: [1..NSP] \rightarrow \mathbb{R}$ 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:

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

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:

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

11. Let $RelRes: [1..NR] \rightarrow 2^{ResSet}$ 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:

$$RelRes(r) = \bigcup_{f \in IBF(r) \cup ITF(r)} RSch.Parm.FeatureRes(f)$$

$\forall\, r \in [1..NR]$

12. Let $CumRelRes: [1..NR] \rightarrow 2^{RSch.Parm.ResSet}$ 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:

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

13. Let $NewRelRes$: $[1..NR] \to 2^{ResSet}$ 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:

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

14. Let $ResCostPerRel : [1..NR] \to \mathbb{R}^{+}$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 $ResCashFlow : [1..TH] \to \mathbb{R}^{+}$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 $CashFlow : [1..TH] \to \mathbb{R}^{+}$, 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:

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

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 #10.
- $\boldsymbol{firstDay}(r)$ is defined in computation #4.
- $\boldsymbol{lastDay}(r)$ is defined in computation #5.

A9. Optimization Formalization

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

$$RSch \left\langle \begin{matrix} Parameters, DecisionVariables, \\ Computation, Constraints, IM \end{matrix} \right\rangle,$$

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

$$NPV_{BPN} = Max \quad RSch.IM.TimeWindowNPV(th) \quad s.t. \quad RSch.Constraints$$

Each of the six formal components implements constraints that are then aggregated under *RSch.Constraints.*

The solution produces:

4. Optimal NPV of the business benefit
5. A release schedule, which is the result of the Solver instantiating *IBF(r,f)* and *ITF(r,f).*
6. The service network configuration at the end of each release, which is captured by the instantiated variables *On(s,r).*

REFERENCES

Boccanera, F., Brodsky, A. (2020). Decision Guidance on Software Feature Selection to Maximize the Benefit to Organizational Processes. 22nd *International Conference on Enterprise Information Systems (ICEIS), pp. 381-395*.

Boccanera, F., Brodsky, A. (2021). *Opti-Soft*: Decision Guidance on Software Release Scheduling to Minimize the Cost of Business Processes. *Enterprise Information Systems: 22nd International Conference, ICEIS 2020, Virtual Event, May 5–7, 2020, Revised Selected Papers* (2021). Springer.

Brodsky, A., Krishnamoorthy, M., Nachawati, M. O., Bernstein, W. Z., & Menascé, D. A. (2017). Manufacturing and contract service networks: Composition, optimization and tradeoff analysis based on a reusable repository of performance models. *2017 IEEE International Conference on Big Data (Big Data)*, 1716–1725.

Brodsky, Alexander, & Luo, J. (2015). Decision Guidance Analytics Language (DGAL)-Toward Reusable Knowledge Base Centric Modeling. *17th International Conference on Enterprise Information Systems (ICEIS)*, 67–78.

Cleland-Huang, J., & Denne, M. (2005). Financially informed requirements prioritization. *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 710–

Denne, M., & Cleland-Huang, J. (2004). The incremental funding method: Data-driven software development. *IEEE Software*, *21*(3), 39–47.

Denne, Mark, & Cleland-Huang, J. (2003). Software by Numbers: Low-Risk, High-Return Development. *Prentice Hall*.

Devaraj, S., & Kohli, R. (2002). The IT Payoff: Measuring the Business Value of Information Technology Investments. *FT Press.*

Elsaid, A. H., Salem, R. K., & Abdelkader, H. M. (2019). Proposed framework for planning software releases using fuzzy rule-based system. *IET Software*, *13*(6), 543–554.

Hannay, J. E., Benestad, H. C., & Strand, K. (2017). Benefit Points: The Best Part of the Story. *IEEE Software*, *34*(3), 73–85.

Maurice, S., Ruhe, G., Saliu, O., & Ngo-The, A. (2006). Decision Support for Value-Based Software Release Planning. In *Value-Based Software Engineering* (pp. 247–261). Springer, Berlin, Heidelberg.

Nachawati, M. O., Brodsky, A., & Luo, J. (2016). Unity: A NoSQL-based Platform for Building Decision Guidance Systems from Reusable Analytics Models. *Technical Report GMU-CS-TR-2016-4*. George Mason University.

Nachawati, M. O., Brodsky, A., & Luo, J. (2017). Unity Decision Guidance Management System: Analytics Engine and Reusable Model Repository. 19th *International Conference on Enterprise Information Systems (ICEIS)*, pp 312–323.

Pucciarelli, J., & Wiklund, D. (2009). Improving IT Project Outcomes by Systematically Managing and Hedging Risk. *IDC Report*.

Riegel, N., & Doerr, J. (2014). An Analysis of Priority-Based Decision Heuristics for Optimizing Elicitation Efficiency. In *Requirements Engineering: Foundation for Software Quality* (pp. 268–284). Springer International Publishing.

Serrador, P., & Pinto, J. (2015). Does Agile work? - A quantitative analysis of agile project success—ScienceDirect. *International Journal of Project Management*, *33*(5), 1040–1051.

The Standish Group. (2018). *CHAOS Report 2018*.

Van den Akker, M., Brinkkemper, S., Diepen, G., & Versendaal, J. (2005). Determination of the Next Release of a Software Product: An Approach using Integer Linear Programming. *CAiSE Short Paper Proceedings*.