Submitted to
manuscript (Please, provide the mansucript number!)

A Hierarchical Framework for Organizing a Software Development Process

Foaad Iravani*, Sriram Dasu†, Reza Ahmadi*
*Anderson School of Management, University of California, Los Angeles, Los Angeles, California 90095
firavani@anderson.ucla.edu, rahmadi@anderson.ucla.edu
†Marshall School of Business, University of Southern California, Los Angeles, California 90089, dasu@marshall.usc.edu

Every year, companies that produce consumer tax preparation software struggle with a massive amount of work imposed by thousands of state and federal changes to tax laws and forms. With their release not even beginning before August, these changes still must be processed and incorporated into the application by mid-December. Three companies dominate this competitive market with its short selling season so that release delays create significant losses. Though systematic resource allocation and process management are crucial, the volume and complexity of the changes, the brief timeframe to implement them, and feedback loops built into the system for error resolution make it extremely difficult to analyze the process. One of the leading tax software providers tasked us with developing systematic approaches for managing the process flow and staffing each stage so that the company met the deadline at the lowest cost. Based on the characteristics of the process, we develop deterministic models that partition tax forms into groups and determine the staffing levels for each group. Partitioning the development process into groups has the benefit of simplifying workflow management and making it easier to find staffing levels. To provide the company with a range of resource configurations, we use two modeling approaches to obtain lower and upper bounds on the number of resources at each stage. Numerical experiments indicate that the models successfully capture the features of the process and the heuristics perform well. Implementing our models at the company has resulted in 31% reduction in overtime and 13% reduction in total resource costs.

Keywords: product development, software development, workforce management, capacity planning, resource allocation, grouping index, integer programming.

1. Introduction

Consumer tax preparation software is a profitable niche in the software development industry, which has become a leading industry worldwide and particularly important in the United States. In 2010, *Software Magazine* reported that the top 500 software companies in the U.S. employed nearly 3.56 million people and earned $491.7 billion in total revenue. The Bureau of Labor Statistics projects that, through the next decade, software will be the third-fastest growing industry in the U.S. economy. Growth in the tax preparation software segment, specifically, has been boosted by IRS efforts to achieve an 80% rate for electronically filing major returns by 2013. In 2011 alone, sales of some consumer tax preparation applications grew by as much as 20%, with 40 million taxpayers using them to file (The Electronic Tax Administration Advisory Committee, 2011). And in this highly regulated domain, makers of these products face the formidable task of speedily integrating thousands of state and federal changes every year.

The idea for this project grew from conversations with a software engineer who is studying for an MBA. The engineer, who works for one of the largest tax preparation software providers in the U.S., characterized the market as fiercely competitive, subject to a short and fixed sales window, and annually faced with obsolescence. Three companies dominate this high-pressure market, racing one another every year to incorporate changes to laws and forms, test the new version, and release a bug-free product for the upcoming tax season. Many taxpayers want sufficient lead time to ensure accurate filing, and those eligible for refunds want to receive them as soon as possible. Therefore, releasing its product early helps the company maximize its market share, and delays can result in significant losses.

The development process for tax software is complex, consisting of multiples stages and incorporating thousands of revisions. Some changes are trivial while others command significant developer time. Adding to the challenge is the fact that state and federal authorities only begin announcing their changes in early August and continue into December, while mid-December marks the start of the tax software sales season. Within the development process itself, built-in feedback loops interrupt the workflow to resolve errors and bugs. To release the software on time and control development costs, therefore, the company must effectively manage the process and accurately determine staffing levels. After a period of observation, we concluded that the complexity of the process thwarts even the most experienced manager's efforts to organize and administer development. We found tasks to be assigned on an ad hoc basis and staffing levels subject to individual power and influence. The firm's bottom line suffered from significant overtime costs and yet the company still found it difficult to achieve a timely release.

Clearly, the large number of forms calls for a staffing plan driven by an effectively organized development effort that simplifies day-to-day operations management, improves coordination among different stages, and facilitates information flow. To that end, we propose a model that restructures the development effort by partitioning tax forms into multiple independent groups and that determines staffing levels throughout the process. Because making the two decisions simultaneously is hard, we employ a hierarchical approach in which we first form the groups and then allocate to them sufficient resources (we use the terms *resource* and *employee* interchangeably) to ensure timely completion. The decision to create groups is supported by studies in software development (e.g., Brooks 1975, Cusumano 1997) which demonstrate such benefits as increased effectiveness and enhanced quality arising from the division of tasks into groups.

We also propose a second model for managing the company's available resources when either hiring is not possible or new employees need time to get up to speed. This model minimizes the total amount of work by splitting available resources between one line for processing all federal forms and one line for processing all state forms.

Although we are applying the hierarchical planning framework to a tax software process, our analysis also can be applied to product development in other highly-regulated industries that necessitate the redesign of the same product, face tight deadlines, and have processing requirements similar to those we encounter here (e.g., Ahmadi et al. 2001). Aerospace, cellular communication, healthcare, and enterprise resource planning (ERP) face similar staffing quandaries. For example, every year the Centers for Medicare & Medicaid Services publishes regulatory updates to payment rules, standard assessments, and resource utilization group and case mix calculations. Producers of clinical and financial software must incorporate these changes before October in an efficient and cost-effective way. In all these disciplines, development teams work in parallel to complete the design tasks and share a common deadline for completion. Thus, task assignment and resource allocation are common issues.

The remainder of this paper is organized as follows. In Section 2, we review the relevant literature. In Section 3, we describe the characteristics of the problem, explain the modeling assumptions, and propose an approximation to capture the effect of feedback loops on the process completion time. We introduce the notation and formulate the models in Section 4 and describe their solution procedures in Section 5. In Section 6, we investigate the performance of the hierarchical approach and the solution procedures using numerical experiments. Section 7 talks about the implementation of our models in the company. The paper is concluded in Section 8 with a summary of the research.

2. Literature Review

The development process for tax preparation software has elements of reentrant flow shops (Graves et al. 1983) and product development projects. On one hand, the development process is repetitive because the software is produced each year and each version encompasses multiple jobs, all facing the same deadline. On the other hand, the changes vary significantly each year so that the company cannot just repeat the same process. Every version of the application, therefore, can be thought of as a project.

In addition to traditional project management techniques (Tavares 1998), several approaches focus on the duration of product development projects, including overlapping activities (Krishnan et al. 1997, Roemer et al. 2000, Roemer and Ahmadi 2004, Loch and Terwiesch 1998) and reducing the number and the size of loops in the process (Smith and Eppinger 1997, Carrascosa et al. 1998, Ahmadi et al. 2001) by changing the sequence of development activities and the flow of information among developers. In our setting, however, the sequence is fairly straightforward and offers minimal opportunity for modification or increased overlap. Therefore, we seek to achieve the desired duration of the development process by creating groups to facilitate the flow of the process and then staffing the stages properly.

Partitioning the development effort and allocating tasks among groups is a common software industry practice and substantially influences project duration, software quality, development cost, and reusability. Cusumano (1997) and Cusumano et al. (2003) survey techniques employed by leading software companies to partition and manage large projects. Such studies largely focus on ways to divide complex tasks into manageable modules (Shaw and Clements 2006). In our problem, we define an index for grouping tasks based on similarities between the amount of work they require and then staff the groups to ensure timely completion.

In software engineering, an interesting question is whether to assign two developers to work on a module, or to assign only one developer. Empirical evidence suggests that assigning multiple developers increases the time and effort to develop a module but it decreases the time and effort to integrate the module. Dawande et al. (2008) develop a mathematical model to find conditions under which one approach dominates the other. In our models, we assume that jobs can be divided among the developers, which is an approximation for tractability.

Browning and Ramasesh (2007) provide a comprehensive review of network-based process models for managing product development activities and name very few papers that are concerned with resource allocation in product development projects. Joglekar and Ford (2005) study ways to dynamically shift a finite pool of resources across different stages of the process using a procedure consisting of three steps, which is considerably simpler than ours. Though we are not concerned with dynamically reassigning resources, Joglekar and Ford intriguingly seems to suggest that complexity diminishes the value of dynamic resource allocation.

In software development, another class of resource allocation problems is concerned with optimally allocating resources among competing priorities. For example, Ji et al. (2005) explore optimal allocation between software construction and debugging to maximize quality. While Kumar et al. (2006) look at the trade-off between the benefits of adding a new feature and the risk of introducing new bugs with it.

Queuing network models also has been proposed for staffing projects. Adler et al. (1992) consider multiple product development projects that proceed through nodes representing departments or functional capabilities. Queuing models assume that the design facility is continually receiving design projects (Adler et al.) or maintenance requests (Asundi and Sarkar 2005, Kulkarni et al.

2009, Feng et al. 2006) each with its own deadline. Although we have multiple tasks in the process, they are all part of a single project.

Resource scheduling problems also arise during software execution (Hos and Shin 1997) where the challenge is to allocate elements of a job in real time to a set of resources so as to complete the job on time. Complexity in these problems stems from the nature of the precedence relationships and interdependencies among the tasks. In our problem, the flow patterns and precedence relationships are simple, and the challenge stems from the large number of tasks that have to be done.

Kekre et al. (2009) address an interesting problem with features similar to our work. They analyze the workforce management of multistage check-clearing operations at a major commercial bank. They use simulation-optimization to capture the tradeoff between efficiency and the risk of delayed checks resulting from excessive workforce reduction. Our problem involves staffing the stages of the process and deciding which group each form should be assigned to. Because the number of forms is very large, numerous scenarios exist for resource allocation and group assignment, even for three to five groups. Therefore, simulation-optimization is not well-suited to our problem, though one could use the technique to explore different staffing scenarios for the fixed assignment of forms to groups.

3. Problem Characteristics and Modeling Assumptions

Maintaining a tax preparation application encompasses multiple complex processes throughout the year, a number of which are concentrated into the quarter before the impending tax cycle. Some processes, such as maintaining the software engine at the core of the application, proceed independently of revisions to the tax code. Five major processes dominate the workflow (see Figure 1).

1. The Image Development Group (IDG) evaluates all tax form and document changes and creates an image of each form.
2. Calculation (CALC) elucidates and tests the computations that underlie each form. The most important stage, CALC also carries the highest amount of work.

3. Electronic Filing (EF) develops electronic versions of the forms, employing functions and macros based on the structure of each form and the fields it contains.
4. The Interview team designs the user interface that guides the consumer through the software.
5. Integration and Final Test incorporate the forms into the application and put each component through exhaustive trials. Integration also designs the buttons, menus, and toolbars. Final Test sends each error back to the team that introduced it.

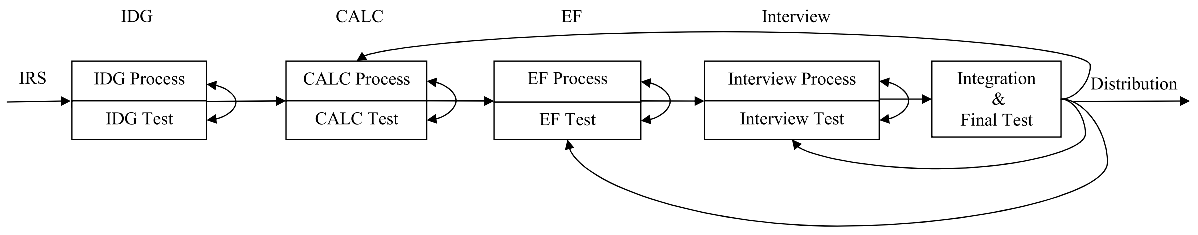


Figure 1 **Tax Software Process Line**

The backward arcs in Figure 1 show the feedback from the final test to upstream stages. If an error is found in a form during the final test, it may be returned to CALC, EF, or Interview for correction. Each stage, in turn, consists of two substages. The first substage is for processing forms and the second is for testing forms internally. The internal tests may send the forms back to their corresponding process for reprocessing.

Besides annual Internal Revenue Service (IRS) changes, each state taxing authority adopts separate tax laws and creates different forms. The number of changes that must be incorporated into the application is large, historically 3,000-5,000, and the taxing authorities release them dynamically starting in August and continuing all the way until mid-December. As stated earlier, some forms contain small changes while others contain changes that are time-consuming to implement, producing considerable variation in processing times. Thus, workflow management and coordination becomes a formidable task.

3.1. Modeling Assumptions

To effectively control this development effort, we need to partition the forms into manageable groups, staff each group, and develop rules for sequencing the tasks. Unfortunately, the volume of work, dynamic arrivals, feedback loops, and variability in processing times make it nearly impossible to identify good sequencing rules. Hence we develop models that support tactical decisions of grouping and staffing and do not consider operational issues such as sequencing and scheduling.

We make two simplifications while developing the tactical model. We have observed that at no point is the system idle due to lack of work. This observation permits us to ignore detailed patterns of dynamic arrival of forms. We can base grouping and staffing decisions on a forecast of amount of work and assume that forms arrive in a deterministic manner. If the forecast is significantly revised during the season, the company can always revisit the models and change staffing levels. This simplification was validated by managers based on their past experience and by our computational experiments reported in section Section 6 which are based on historical data. The Second simplification concerns feedback loops which impact the process in several ways. They increase the amount of work at each stage, introduce additional uncertainty into processing time requirements, and may cause a stage to wait while a downstream stage is creating a rework loop. Given our interest in completion time, the last impact is the most problematic. The likelihood of inserted idle times depends on the initial amount of work – when the amount of work is high, inserted idle times are very unlikely. In absence of inserted idle times, we may be able to approximate the process with loops with one without loops, provided we suitably modify the processing times at each stage.

3.2. Approximating Feedback Loops

Figure 2 shows an alternate depiction of the development process that separates the sub-stages and Figure 3 shows an approximating process with no feedback loops. Note that IDG process and its internal test are eliminated because the final test never sends a form back to the IDG for rework. However, the approximation we describe here will be separately applied to the feedback loop from IDG internal test to IDG proces.

In the approximating process, the processing time distribution of a form at each stage is determined by the the original processing time distribution of the form and the distribution of the number of times the form revisits the stage. In the original process, let $\rho_k(i,n)$ represent the probability that the number of times form i visits stage k is n and let $f_k(i,t)$ denote the processing time distribution of form i at stage k. In the approximating process, the processing time distribution for form i at stage k is defined as $h_k(i,t) = \sum_n \rho_k(i,n) f_k^{(n)}(i,t)$, where $f_k^{(n)}(i,t)$ is the n–fold convolution of $f_k(i,t)$. In the unabridged version of the paper (Iravani et al. 2011), we show that for a two-stage system with deterministic processing times the completion time in the no-loops process converges to that of the original process as the number of forms increases. Also, our numerical experiments in Section 6 show that the approximation performs well.

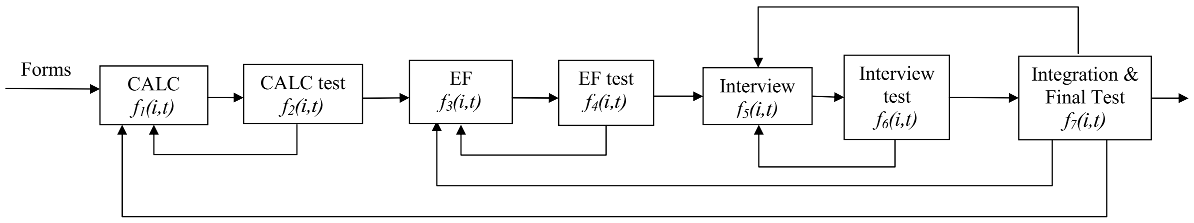


Figure 2 Process with Feedback Loops

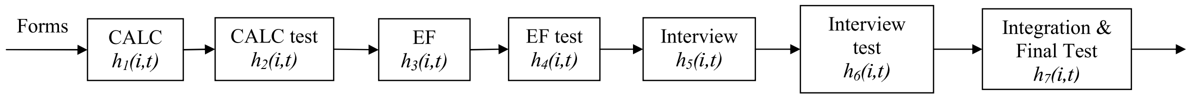


Figure 3 Approximate No-loops Process

4. Models

Based on the foregoing assumptions, we formulate a monolithic model that simultaneously partitions tax forms into groups–company managers determine their total number–and staffs the groups. We find that the monolithic model is strongly NP-hard and solvers such as Industrial Lingo are ineffective in solving even moderate-size problems that involve more than 100 forms. Thus, we adopt a hierarchical approach to formulating problems that calculate the similarity between processing times for forms, assign forms to groups based on the similarity index, and staff the groups to release the software on time. This approach is similar to the classic hierarchical production planning approach in Hax and Candea (1984). One can picture the groups of forms as families of similar items. While Hax and Candea focus on disaggregating the production plan of product types to product families and items, we are concerned with staffing the production process. Because our hierarchical staffing models assume the ability to hire additional resources when needed to finish processing forms before the deadline, we also include a model for reallocating current employees. We formulate the models as follows:

1. Monolithic grouping and resource allocation problem (GRAP)
2. Hierarchical framework (see Figure 4)
   (a) Grouping index problem (GIP)
   (b) Grouping problem (GP)
   (c) Resource allocation problem (RAP)
3. Process line separation problem (PLSP) to reallocate existing resources

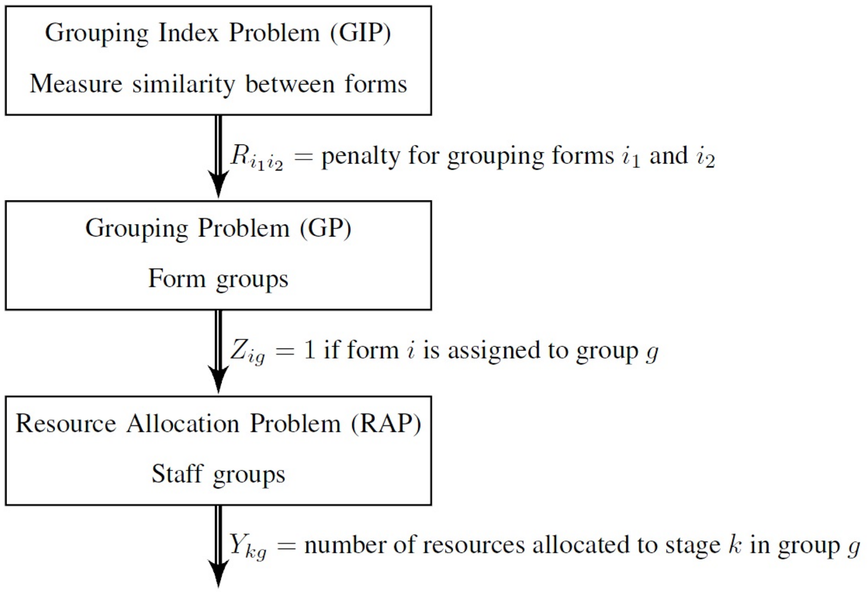

Grouping Index Problem (GIP)
Measure similarity between forms
$R_{i_1 i_2}$ = penalty for grouping forms i_1 and i_2
Grouping Problem (GP)
Form groups
$Z_{ig} = 1$ if form i is assigned to group g
Resource Allocation Problem (RAP)
Staff groups
Y_{kg} = number of resources allocated to stage k in group g

Figure 4 **Steps of the hierarchical framework for grouping and resource allocation**

4.1. Grouping and Resource Allocation Problem

Formulating the GRAP entails one approach to determine the upper bound for optimal resources (GRAP_1) and another to obtain the lower bound (GRAP_2). For the upper bound, we allocate resources in such a way that the sum of the maximum effective processing time *across all stages* for all forms is less than the deadline. For the lower bound, we allocate resources in such a way that the total effective processing time *at each stage* is less than the deadline.

Let P_{ik} denote the processing time for form i at stage k, which in fact represents the total time needed to perform *a number of divisible tasks* for this form (see Table 1 for notation). If Y_{kg} is the number of resources allocated to stage k of group g, and form i is processed in group g, then the effective processing time of form i at stage k is roughly $\frac{P_{ik}}{Y_{kg}}$. Let $T_i = \max_k \frac{P_{ik}}{Y_{kg}}$ be the maximum effective processing time for form i. A simple yet tractable approximation for the time to complete all the forms in this group is $\sum_{i=1}^{I} T_i$. We find that this estimate, motivated by Proposition 1, continues to be accurate in our numerical experiments even with feedback loops. Appendix 1 provides proofs for all propositions.

PROPOSITION 1. *Consider a flow shop with K single resource stages processing I jobs (forms) with processing times P_{ik}. The time to complete these jobs is at most $\sum_{i=1}^{I} max_k \{P_{ik}\} + (K - 1)max_i\{max_k \{P_{ik}\}\}$.*

Table 1 Notations for the GRAP

Parameters	
k	Index for process stages $1, \ldots, K$.
g	Index for groups $1, \ldots, G$ to be formed.
i	Index for forms $1, \ldots, I$.
P_{ik}	Processing time of form i at stage k.
w_k	Cost of allocating one resource to stage k.
D	The deadline by which all forms have to be compiled into the software.
Variables	
Y_{kg}	Number of resources (employees) allocated to stage k of group g.
T_i	Maximum effective processing time of form i if it is assigned to group g. In mathematical terms, $T_i = \max_k\{P_{ik}/Y_{kg}\}$.
Z_{ig}	1 if form i is allocated to group g; 0 otherwise.

As the number of jobs becomes large relative to the number of stages, $(K-1)\max_i\{\max_k \{P_{ik}\}\}$ becomes relatively small and the completion time asymptotically approaches $\sum_{i=1}^{I} \max_k \{P_{ik}\}$. In our problem, each group processes at least 1,000 forms so that, in our models, we use $\sum_{i=1}^{I} T_i$ as an approximation for the time to complete all forms. Thus, we formulate GRAP$_1$ as

$$(\text{GRAP}_1) \quad \text{Min} \sum_{k=1}^{K}\sum_{g=1}^{G} w_k Y_{kg} \tag{1}$$

$$s.t.$$

$$T_i \geq \frac{P_{ik} Z_{ig}}{Y_{kg}} \qquad \forall i, \quad \forall k, \quad \forall g, \tag{2}$$

$$\sum_{i=1}^{I} T_i Z_{ig} \leq D \qquad \forall g, \tag{3}$$

$$\sum_{g=1}^{G} Z_{ig} = 1 \qquad \forall i, \tag{4}$$

$$Y_{kg} \geq 1 \text{ and integer} \quad \forall k, \quad \forall g \tag{5}$$

$$Z_{ig} \in \{0,1\} \qquad \forall i. \tag{6}$$

Objective function (1) minimizes total resource cost. Constraint (2) defines the maximum effective processing time for form i based on the group to which it is assigned. Constraint (3) ensures that all forms are completed by the deadline. Constraint (4) guarantees that each form is assigned to one group only. Finally constraints (5) and (6) ensure that resources are positive integers and Z_{ig} is a binary variable.

In GRAP$_2$, we require that the total effective time to process all forms at each stage be less than the due date. In a flow shop, the maximum of the total effective processing time at each stage is a lower bound for completion time. Consequently, the solution to GRAP$_2$ represents the minimum number of resources needed to meet the deadline. The formulation for GRAP$_2$ is the same as GRAP$_1$, except that we replace constraints (2) and (3) with:

$$\sum_{i=1}^{I} \frac{P_{ik} Z_{ig}}{Y_{kg}} \leq D, \quad \forall k, \quad \forall g. \tag{7}$$

We now establish the complexity of the GRAP.

Proposition 2. *GRAP$_1$ and GRAP$_2$ are strongly NP-hard.*

4.2. Grouping Index Problem

The first component of our hierarchical framework produces a means of measuring the similarity between forms by articulating and solving the GIP. The idea behind grouping forms is the notion that if form i_1 and form i_2 have proportional processing times in each stage, i.e., $\frac{P_{i_1k}}{P_{i_2k}}$ is the same for all k, then it is suitable to assign these forms to the same group. Suppose that forms i_1 and i_2 are to be processed in one group by Y_k resources at stage k and define

$$\Lambda_{i_1} = \max_k \left\{ \frac{P_{i_1k}}{Y_k} \right\}, \quad \Lambda_{i_2} = \max_k \left\{ \frac{P_{i_2k}}{Y_k} \right\}.$$

For these forms, we define the *balance loss* as total idle time arising from the difference between effective processing times at bottleneck and non-bottleneck stages:

$$\sum_{k=1}^{K} \left\{ \left(\Lambda_{i_1} - P_{i_1k}/Y_k\right) + \left(\Lambda_{i_2} - P_{i_2k}/Y_k\right) \right\} Y_k = \left(\Lambda_{i_1} + \Lambda_{i_2}\right) \sum_{k=1}^{K} Y_k - \sum_{k=1}^{K} \left(P_{i_1k} + P_{i_2k}\right) \tag{8}$$

The smaller the balance loss, the more suitable i_1 and i_2 are to be placed in the same group. We define the GIP for states i_1 and i_2 as

$$\text{(GIP)} \quad \text{Min } \left(\Lambda_{i_1} + \Lambda_{i_2}\right) \sum_{k=1}^{K} Y_k \tag{9}$$

s.t.

$$\Lambda_{i_1} \geq \frac{P_{i_1k}}{Y_k} \qquad \forall k, \tag{10}$$

$$\Lambda_{i_2} \geq \frac{P_{i_2k}}{Y_k} \qquad \forall k, \tag{11}$$

$$Y_k \geq 1 \qquad \forall k. \tag{12}$$

The GIP addresses the following question: *Assuming unlimited resources, what is the minimum balance loss we will incur if we assign forms i_1 and i_2 to the same group?* Note that (9) is the variable part of (8). Also, resources are allowed to take real values because the GIP is not concerned with resource allocation. The GIP can be solved by defining $\beta = \frac{\Lambda_{i_1}}{\Lambda_{i_1}+\Lambda_{i_2}}$, and finding Y_k from (10) and (11) as follows:

$$\text{Min } \left(\Lambda_{i_1} + \Lambda_{i_2}\right) \sum_{k=1}^{K} Y_k \equiv \text{Min}_{\Lambda_{i_1}, \Lambda_{i_2} > 0} (\Lambda_{i_1} + \Lambda_{i_2}) \sum_{k=1}^{K} \max\left(P_{i_1k}/\Lambda_{i_1}, P_{i_2k}/\Lambda_{i_2}\right)$$

$$\equiv \text{Min}_{\Lambda_{i_1},\Lambda_{i_2}>0} \sum_{k=1}^{K} \max\left(P_{i_1k}/[\Lambda_{i_1}/(\Lambda_{i_1}+\Lambda_{i_2})], P_{i_2k}/[\Lambda_{i_2}/(\Lambda_{i_1}+\Lambda_{i_2})]\right) \quad (13)$$

$$\equiv \text{Min}_{0<\beta<1} \sum_{k=1}^{K} \max\left(P_{i_1k}/\beta, P_{i_2k}/(1-\beta)\right). \quad (14)$$

The function $\max(P_{i_1k}/\beta, P_{i_2k}/(1-\beta))$ is convex in β. It is also unimodal because its value goes to infinity when β approaches either boundary of $(0,1)$. Because the objective function is the sum of convex unimodal functions we can easily find the optimal solution. Let $\Lambda^*_{i_1}, \Lambda^*_{i_2}, Y^*_k$ be the optimal solution of (9). We define the penalty for placing forms i_1 and i_2 in the same group to be $R_{i_1i_2} = d_{i_1i_2} - \min_{b\neq i_1} d_{i_1b} + d_{i_2i_1} - \min_{b\neq i_2} d_{i_2b}$ in which $d_{i_1i_2} = \Lambda^*_{i_1}\sum_{k=1}^{K} Y^*_k - \sum_{k=1}^{K} P_{i_1k}$, $d_{i_2i_1} = \Lambda^*_{i_2}\sum_{k=1}^{K} Y^*_k - \sum_{k=1}^{K} P_{i_2k}$, and d_{i_1b} (d_{i_2b}) denotes the value of $d_{i_1i_2}$ $(d_{i_2i_1})$ when (9) is solved for i_1 (i_2) and $b\neq i_1$ $(b\neq i_2)$. Going back to (8), one can see that $d_{i_1i_2}(d_{i_2i_1})$ is the proportion of idle times attributable to form $i_1(i_2)$. The grouping index is defined for every pair of forms. Using the sum of pairwise grouping indices entails double counting when a group includes more than two states. The terms $\min_{b\neq i_1} d_{i_1b}$ and $\min_{b\neq i_2} d_{i_2b}$ are subtracted to alleviate that effect.

4.3. Grouping Problem

Now we use the GIP to assign forms to groups, which were defined by company process managers, by formulating the GP. The lower the $R_{i_1i_2}$, the better it is to have these forms in the same group. By defining

$$X_{i_1i_2} = \begin{cases} 1 & \text{if forms } i_1 \text{ and } i_2 \neq i_1 \text{ are assigned to the same group,} \\ 0 & \text{otherwise} \end{cases}$$

and $P_i = \sum_{k=1}^{K} P_{ik}$, we can formulate the GP for any pair of forms

$$\text{(GP) Min} \sum_{i_1=1}^{I-1} \sum_{i_2=i_1+1}^{I} R_{i_1i_2} X_{i_1i_2} \quad (15)$$

s.t.

$$X_{i_1i_2} \geq Z_{i_1g} + Z_{i_2g} - 1 \qquad \forall g,\ \forall i_1, i_2,\ i_1 \neq i_2 \quad (16)$$

$$\sum_{g=1}^{G} Z_{ig} = 1 \qquad \forall i, \quad (17)$$

$$\sum_{i=1}^{I} P_i Z_{ig} \leq Q \qquad \forall g, \quad (18)$$

$$X_{i_1 i_2} \in \{0,1\} \qquad \forall i_1, i_2,\ i_1 \neq i_2, \tag{19}$$

$$Z_{ig} \in \{0,1\} \qquad \forall i,\ \forall g. \tag{20}$$

where $Q = (1+\delta)\sum_{i=1}^{I}\sum_{k=1}^{K} P_{ik}/G$. Objective function (15) minimizes the total penalty for grouping forms. Constraint (16) is a logical constraint that links $Z_{i_1 g}$ and $Z_{i_2 g}$ to $X_{i_1 i_2}$. Constraint (17) ensures each form is assigned to a group. Constraint (18) states that the total processing times for the forms in each group should not exceed the average total processing time per group by more than a predefined fraction. Ideally, we want δ to be zero, but this may make the problem infeasible.

On other hand if δ is too large, the constraint becomes redundant. Although the value of δ need not be unique, our numerical investigations indicate that 0.25 is reasonable for 3 to 5 groups. However, when the number of groups increases, δ should be increased to ensure feasibility. We can now attest to the complexity of the GP.

PROPOSITION 3. *The GP is strongly NP-complete.*

4.4. Resource Allocation Problem

For the last component of the hierarchical framework, we solve the RAP to establish staffing levels that allow each group to finish processing their assigned forms on time at minimum expense. Procedures for formulating and solving the RAP are the same for all groups. Therefore, instead of defining i_g as the index for forms assigned to group g, with a little abuse of notation we use i in this section to index forms that are assigned to a group. RAP$_1$ is formulated as

$$(\text{RAP}_1)\ \ \text{Min} \sum_{k=1}^{K} w_k Y_k \tag{21}$$

s.t.

$$T_i \geq \frac{P_{ik}}{Y_k} \qquad \forall k,\ \forall i, \tag{22}$$

$$\sum_{i=1}^{I} T_i \leq D \tag{23}$$

$$Y_k \geq 1 \text{ and integer}, \qquad \forall k. \tag{24}$$

We obtain RAP$_2$ from RAP$_1$ by replacing (22) and (23) with $\sum_{i=1}^{I} P_{ik}/Y_k \leq D$ for all k. The following proposition states that RAP$_1$ is a hard problem.

Proposition 4. *RAP_1 is binary NP-hard.*

4.5. Process Line Separation Problem

The final model addresses the PLSP to help the company manage existing resources. The three models in the hierarchical framework assume that the company has already decided to acquire additional resources if necessary. As a strategic decision, hiring new employees requires financial resources, time to interview candidates, and time to train new hires. In a crunch, the company may need to reconfigure current resources as it prepares new employees to join the teams. Thus, we propose a model that distributes existing resources to two major process lines: one line for processing all federal forms and one line for processing all state forms.

Let M_k be the current number of employees at stage k. Also, let I_S and I_F denote the number of federal and state forms indexed by i_s and i_f. We use Y_{ks} and Y_{kf} to denote the resources in stage k working on state and federal forms. The PLSP is defined as

$$\text{(PLSP)} \quad \text{Min} \sum_{i_s=1}^{I_S} T_{i_s} + \sum_{i_f=1}^{I_F} T_{i_f} \tag{25}$$

$$s.t.$$

$$T_{i_s} \geq \frac{P_{i_s k}}{Y_{ks}} \qquad \forall k, \;\; \forall i_s, \tag{26}$$

$$T_{i_f} \geq \frac{P_{i_f k}}{Y_{kf}} \qquad \forall k, \;\; \forall i_f, \tag{27}$$

$$Y_{ks} + Y_{kf} \leq M_k \qquad \forall k, \tag{28}$$

$$Y_{ks}, Y_{kf} \geq 1 \text{ and integer} \qquad \forall k. \tag{29}$$

Objective function (25) minimizes the total maximum amount of work for all forms. Constraints (26) and (27) define the maximum amount of work for state and federal tax forms. Constraint (28) shows resource availability.

Proposition 5. *PLSP is binary NP-hard.*

Unlike the previous models, we employ only the first approach to formulate the PLSP because $\sum_{i=1}^{I} T_i$ is an upper bound on the time to process all forms. Therefore, by dividing the resources, we minimize the upper bound on the completion time. On the other hand, if we were to formulate and solve PLSP under the second approach, we would minimize a lower bound, which has no value.

5. Solution Procedures

The three models constituting the hierarchical approach provide an approximate solution for the very difficult GRAP. Of the three, we needed the GIP solution in order to define $R_{i_1i_2}$ with which we formulate the GP. The remaining problems require more elaborate solution procedures.

5.1. GP Solution

To solve the GP, we propose a decomposition procedure that provides both a lower bound and feasible solutions. When we relax (16) by positive Lagrangian multipliers $\gamma_{i_1i_2g}$, we get

$$L(\gamma) = \text{Min} \sum_{i_1=1}^{I-1}\sum_{i_2=i_1+1}^{I}\left(R_{i_1i_2} - \sum_{g=1}^{G}\gamma_{i_1i_2g}\right)X_{i_1i_2} + \sum_{g=1}^{G}\sum_{i_1=1}^{I}\left(\sum_{i_2=1}^{i_1-1}\gamma_{i_2i_1g} + \sum_{i_2=i_1+1}^{I}\gamma_{i_1i_2g}\right)Z_{i_1g}$$
$$-\sum_{g=1}^{G}\sum_{i_1=1}^{I-1}\sum_{i_2=i_1+1}^{I}\gamma_{i_1i_2g}$$
$$s.t.$$
$$(17),(18),(19),(20)$$

For a vector of Lagrangian multipliers, γ, with $\gamma \geq 0$, $L(\gamma)$ can be decomposed into two problems:

$$L_1(\gamma) = \text{Min} \sum_{i_1=1}^{I-1}\sum_{i_2=i_1+1}^{I}\left(R_{i_1i_2} - \sum_{g=1}^{G}\gamma_{i_1i_2g}\right)X_{i_1i_2}$$
$$s.t.$$
$$(19),$$

$$L_2(\gamma) = \text{Min} \sum_{g=1}^{G}\sum_{i_1=1}^{I}\theta_{i_1g}Z_{i_1g}$$
$$s.t.$$
$$(17),(18),(20),$$

in which $\theta_{i_1g} = \sum_{i_2=1}^{i_1-1}\gamma_{i_2i_1g} + \sum_{i_2=i_1+1}^{I}\gamma_{i_1i_2g}$.

In $L_1(\gamma)$, the optimal value of each $X_{i_1i_2}$ is equal to 0 if its coefficient in the objective function is positive; otherwise it is equal to 1. $L_2(\gamma)$ is a packing-by-cost variation of the Bin Packing Problem in which a set of items should be packed in bins (groups) of the same volume to minimize the total packing cost. We solve the Bin Packing subproblem using a dynamic program in which $V_i(U_1, U_2, ..., U_G)$ denotes the minimum cost of assigning form i to one of the groups with sufficient capacity, given that the remaining capacity of bin g is U_g and forms 1 to $i-1$ are already assigned. The value functions can be calculated using the following recursive relation:

$$V_i(U_1, U_2, ..., U_G) = \min_{\{g|U_g \geq P_i\}} \{\theta_{ig} + V_{i+1}(U_1, U_2, ..., U_g - P_i, ..., U_G)\} \quad \forall i. \tag{30}$$

To ensure feasibility, we set $V_i(U_1, U_2, ..., U_G) = +\infty$, if $U_g < P_i$ for all g. The optimal solution to $L_2(\gamma)$ is $V_1(Q, Q, ..., Q)$ and the complexity of the procedure is $O(IQ^G)$.

Since $L(\gamma)$ is a lower bound on (15), we solve the following dual problem to find the best of such lower bounds:

$$\begin{aligned} &\text{Max } L(\gamma) \\ &s.t. \\ &\quad \gamma_{i_1 i_2 g} \geq 0, \quad \forall i_1, i_2, \forall g. \end{aligned}$$

A subgradient optimization algorithm (Fisher 1981, 1985) is used to solve the dual problem.

In addition to obtaining a lower bound on the objective function, we also use the decomposition procedure to find feasible solutions to the GP. In each iteration of the subgradient optimization algorithm, we can construct a feasible solution to the GP by using the optimal values of Z_{ig} in $L_2(\gamma)$ and setting $X_{i_1 i_2} = \max_{1 \leq g \leq G}\{Z_{i_1 g} + Z_{i_2 g} - 1\}$. These feasible grouping scenarios become inputs to the RAP for finding the optimum resource allocation.

We close this section by pointing out that the grouping procedure can also be applied to states instead of forms. To calculate $R_{j_1 j_2}$ for states j_1 and j_2, one should replace $P_{i_1 k}$ and $P_{i_2 k}$ with $\sum_{i_{j_1}} P_{i_{j_1} k}$ and $\sum_{i_{j_2}} P_{i_{j_2} k}$ in which $P_{i_j k}$ denotes the processing time for form i of state j at stage k.

5.2. RAP Solution

RAP$_1$ can be transformed to a Shortest Path Problem. First, we find lower bound Y_k^{min} and upper bound Y_k^{max} for Y_k^*, the optimal solution, to limit the size of the network. A lower bound on $Y_1^*, \ldots, Y_K^*$ can be found by summing both sides of (22) over i and combining it with (23). Thus, $Y_k^* \geq Y_k^{min} = \left\lceil \sum_{i=1}^{I} P_{ik}/D \right\rceil$, for which $\lceil x \rceil$ is the smallest integer larger than or equal to x. For an upper bound, set $P_{ik} = \max_k \{P_{ik}\}$ for all k to inflate processing times to the maximum processing time over all stages. Then, if $Y_k = \overline{Y} = \sum_{i=1}^{I} \max_k \{P_{ik}\} / D$ for all k, the deadline will be met. Therefore, $\sum_{k=1}^{K} w_k \overline{Y} \geq \sum_{k=1}^{K} w_k Y_k^*$ which means $Y_k^* \leq Y_k^{max} = \left\lceil \left(\overline{Y} \sum_{k=1}^{K} w_k - \sum_{r \neq k} w_r Y_r^{min} \right) / w_k \right\rceil$.

The network for RAP$_1$ has K layers and each node is represented by (k, Y_k, E) where $E = \sum_{i=1}^{I} T_i - D$ given resources $Y_1, \ldots, Y_K$. The network is constructed using the following steps:

Step 0. Generate Start (0) and Finish (F) nodes.

Step 1. Generate nodes $(1, Y_1, E)$ starting from $Y_1 = Y_1^{min}$ and increasing it by 1 while setting $Y_r = Y_r^{min}$ for $r > 1$. The cost of arc $(0) \to (1, Y_1, E)$ is $(Y_1 - Y_1^{min})\,\omega_1$. Stop adding new nodes if $Y_1 = Y_1^{max}$ or $E \leq 0$. If $E \leq 0$ occurs first, connect the last node to node F with cost 0.

Step 2. In layers $k = 2, \ldots, K$, generate the children of nodes in layer $k-1$ with $E > 0$ in the same manner as Step 1. To be more specific, the value of E in node (k, Y_k, E) is computed by setting $Y_r = Y_r^{min}$ for $r > k$ and setting Y_r for $r \leq k$ equal to their values on the path $(1, Y_1, E) \to (2, Y_2, E) \to \cdots \to (k-1, Y_{k-1}, E) \to (k, Y_k, E)$. The cost of arc $(k-1, Y_{k-1}, E) \to (k, Y_k, E)$ is $(Y_k - Y_k^{min})\,\omega_k$. In layer K, the cost of $(K, Y_K, E) \to F$ will be a very large number if $E > 0$ and 0 otherwise.

Step 3. Compute the shortest path of the network. The optimal solution to RAP$_1$ is the sum of the shortest path and the cost of allocating $Y_k^{\min}$ to each stage. The optimal resource configuration can be determined by traversing the shortest path.

The network is of order size complexity $O\left((\max_k \{Y_k^{max} - Y_k^{min}\})^K \right)$. Appendix 2 provides an example of the network construction. Finally, the solution to RAP$_2$ is $Y_k^* = \left\lceil \sum_{i=1}^{I} P_{ik}/D \right\rceil$.

5.3. PLSP Solution

To solve the PLSP, we propose a heuristic solution that relaxes the integrality of Y_{kg} and then we show its worst-case performance. We let $\widetilde{Y}_{ks}$ and $\widetilde{Y}_{kf}$ be the solution to continuous PLSP, and we find an integer solution using the following algorithm:

Step 1. For all k, set $Y_{ks} = \lfloor \widetilde{Y}_{ks} \rfloor$ and $Y_{kf} = \lfloor \widetilde{Y}_{kf} \rfloor$, for which $\lfloor x \rfloor$ is the greatest integer less than or equal to x and define $\overline{M}_k = \max(0, M_k - Y_{ks} - Y_{kf})$.

Step 2. Find the first stage k with $\overline{M}_k > 0$. First compute TS_k, the objective function of PLSP for $(Y_{ks} + 1, Y_{kf})$. Then compute TF_k, the objective function of PLSP for $(Y_{ks}, Y_{kf} + 1)$. If

$TS_k < TF_k$, let $Y_{ks} := Y_{ks} + 1$; otherwise, let $Y_{kf} := Y_{kf} + 1$. Update $\overline{M}_k$ and repeat this step until $\overline{M}_k = 0$. Then find the next stage with $\overline{M}_k > 0$ and repeat this step.

Let ϕ^H be the objective function value of the above heuristic. The next proposition provides a bound on the worst-case performance of the heuristic.

Proposition 6. *If ϕ^* denotes the optimal value of PLSP, then* $\frac{\phi^H}{\phi^*} \leq 2$.

6. Numerical Experiments on Performance

Through extensive numerical experiments, we explore model and solution performance prior to incorporating them into the company's development processes. First, we show that the Lagrangian heuristic provides a very good solution to the GP. Second, we compare the performance of the hierarchical framework to the monolithic approach for small problems. These two experiments, though limited in the size of problems solved, indicate that the hierarchical framework is an excellent approach to the original problem. Third, we use detailed scheduling to check the performance of the proposed resource configurations. Finally, we look at the effect of dynamically arriving forms and feedback loops on the performance of our solutions.

We select experimental parameters based on our observations of the company. Historically, approximately 50% of the forms are easy, 20% are time consuming, and the remaining 30% fall somewhere in the middle. The categorization is based on processing time requirements. Table 2 shows the processing time intervals (in hours) for each category. The last row presents the resource costs in tens of thousands of dollars. The process starts on August 1st and continues through December 15th. Employees work 8 hours per day, Monday through Friday.

Table 2 Processing time intervals for form categories and resource costs at each stage

Form categories	IDG process	IDG test	CALC process	CALC test	EF process	EF test	Interview process	Interview test	Integration	Final Test
Easy	[2 5]	[1 2]	[3 6]	[1 2]	[2 5]	[0.5 1]	[1 3]	[0.5 1]	[0.5 1.5]	[1 2]
Fairly hard	[5 12]	[1.5 2.5]	[8 12]	[2 3]	[4 7]	[0.5 1.5]	[2 5]	[0.5 1.5]	[1 1.5]	[2 3]
Hard	[12 20]	[2 3]	[12 24]	[3 4]	[6 9]	[1 1.5]	[4 7]	[1 1.5]	[1.5 2]	[3 4]
Cost	50	10	60	15	40	10	35	10	50	15

6.1. Lagrangian Heuristic Performance

To evaluate the quality of the Lagrangian heuristic in solving the grouping problem (GP), we set up the experiment as follows: (1) the number of groups, G, is set to 3, 4, and 5; (2) the number of processing stages, K, is set to 8, 9, and 10; and (3) the number of forms is set to 30, 40, and 50. We have generated 10 problems for each category of problem type and reported the average ratio of the Lagrangian heuristic to the lower bound (LH/LB). Note that at each iteration of the subgradient method, we generate a heuristic solution. We report the result of the best heuristic solution over the 35 iterations of the subgradient method.

Table 3 Performance of the Lagrangian heuristic for the GP

NO.	G	K	I	Ave(LH/LB)	Wrst(LH/LB)
1		8	30	1.023	1.041
2		8	40	1.019	1.039
3		8	50	1.025	1.057
4		9	30	1.034	1.066
5	3	9	40	1.017	1.038
6		9	50	1.019	1.042
7		10	30	1.036	1.079
8		10	40	1.028	1.059
9		10	50	1.041	1.081
10		8	30	1.038	1.077
11		8	40	1.027	1.066
12		8	50	1.043	1.084
13		9	30	1.036	1.068
14	4	9	40	1.023	1.092
15		9	50	1.028	1.088
16		10	30	1.018	1.062
17		10	40	1.039	1.071
18		10	50	1.028	1.066
19		8	30	1.042	1.089
20		8	40	1.045	1.095
21		8	50	1.032	1.082
22		9	30	1.029	1.079
23		9	40	1.041	1.094
24	5	9	50	1.031	1.097
25		10	30	1.048	1.089
26		10	40	1.026	1.088
27		10	50	1.029	1.079
Average				1.031	1.072

Table 3 shows our computational results. For each category, we report two statistics for each of the ten problems solved: the average ratio, Ave(LH/LB), and the worst ratio, Wrst(LH/LB). The average ratio of the best upper bound to the lower bound is quite good. On average, the heuristic is only 3.1 percent above the lower bound. Note that this is a conservative estimate of the quality of the Lagrangian heuristics since we do not have the optimum solution. Even in the worst scenario, our procedure is on average 7.2 percent above the lower bound over the 270 problems solved.

6.2. Hierarchical Framework Performance

In this experiment, we evaluate the quality of the proposed hierarchical framework, the GP and RAP$_1$, and compare it to the monolithic approach for small problems. One drawback to this experiment is that we were not able to solve large problems using the monolithic approach; therefore, we cannot make any claims regarding the quality of the hierarchical approach for large problems. We use the following parameters: (1) the number of groups is set to 2, 3, and 4; (2) the number of stages is assumed to be 8, 9, and 10; and (3) the total number of forms ranges from 50 to 100.

Table 4 shows our computational results. We report Ave(HA/MO), which is the average ratio of the total cost of resources allocated using the hierarchical approach to that using the monolithic approach. In each category, ten problems were solved to optimality. The grand average of the problems solved indicates that the hierarchical problem is at most 3.58 percent above the monolithic approach.

Table 5 reports the average total time in seconds it takes to solve the hierarchical models (GIP, GP, and RAP) for $G = 3, 4, 5$, $K = 8, 9, 10$, and $I = 500, 750, 1000$. The computation time for each combination of G, K, and I shows the average time for solving 10 problem instances. It is evident from the data that the hierarchical approach has a satisfactory runtime.

6.3. RAP Performance

In this section, we evaluate the performance of the two proposed resource allocation problems, RAP$_1$ and RAP$_2$. The parameters used are as follows: (1) the number of stages, K, is set to 8, 9, and 10; and (2) The total number of forms is set to 500, 750, and 1000. A total of 270 problems were solved with 30 problems in each category. To report the results, we first solved RAP$_1$ and RAP$_2$

Table 4 The hierarchical approach vs. the monolithic approach

NO.	G	K	$I = 50$ Ave(HA/MO)	$I = 60$ Ave(HA/MO)	$I = 70$ Ave(HA/MO)	$I = 80$ Ave(HA/MO)	$I = 90$ Ave(HA/MO)	$I = 100$ Ave(HA/MO)
1		8	1.0408	1.0298	1.0355	1.0227	1.0050	1.0433
2		8	1.0373	1.0252	1.0104	1.0183	1.0475	1.0280
3		8	1.0305	1.0236	1.0084	1.0301	1.0292	1.0451
4		9	1.0215	1.0254	1.0411	1.0267	1.0453	1.0544
5	3	9	1.0416	1.0157	1.0126	1.0393	1.0521	1.0038
6		9	1.0063	1.0427	1.0381	1.0412	1.0113	1.0394
7		10	1.0251	1.0096	1.0411	1.0462	1.0439	1.0024
8		10	1.0074	1.0429	1.0309	1.0278	1.0411	1.0239
9		10	1.0240	1.0099	1.0328	1.0014	1.0325	1.0462
10		8	1.0347	1.0395	1.0067	1.0288	1.0070	1.0308
11		8	1.0377	1.0159	1.0423	1.0431	1.0017	1.0155
12		8	1.0093	1.0455	1.0319	1.0368	1.0543	1.0327
13		9	1.0051	1.0232	1.0090	1.0035	1.0384	1.0234
14	4	9	1.0071	1.0209	1.0043	1.0248	1.0070	1.0546
15		9	1.0257	1.0408	1.0290	1.0494	1.0504	1.0299
16		10	1.0003	1.0431	1.0414	1.0212	1.0158	1.0551
17		10	1.0076	1.0059	1.0190	1.0309	1.0116	1.0087
18		10	1.0260	1.0289	1.0203	1.0229	1.0163	1.0395
19		8	1.0342	1.0270	1.0200	1.0450	1.0394	1.0452
20		8	1.0337	1.0380	1.0111	1.0340	1.0234	1.0435
21		8	1.0426	1.0347	1.0354	1.0171	1.0111	1.0578
22		9	1.0369	1.0215	1.0164	1.0292	1.0124	1.0339
23		9	1.0310	1.0332	1.0095	1.0006	1.0171	1.0598
24	5	9	1.0140	1.0238	1.0136	1.0397	1.0211	1.0346
25		10	1.0160	1.0226	1.0121	1.0148	1.0092	1.0303
26		10	1.0082	1.0423	1.0036	1.0240	1.0511	1.0558
27		10	1.0350	1.0445	1.0423	1.0251	1.0336	1.0292
Average			1.0237	1.0287	1.0229	1.0276	1.0270	1.0358

and obtained the resources allocated, Y_k. Then, we computed the completion time of each form by scheduling the tax forms using the resource configurations obtained from solving RAP_1 and RAP_2 and following the shortest total processing time (STPT) rule. We are interested in evaluating the completion time of the forms for each case. In the next section we examine the effect of dynamic arrival and feedback loops on system performance.

Table 5 Computation time in seconds for the hierarchical approach

NO.	G	K	I	Average total computation time
1	3	8	500	41
2		8	750	82
3		8	1000	106
4		9	500	60
5		9	750	85
6		9	1000	101
7		10	500	60
8		10	750	67
9		10	1000	94
10	4	8	500	65
11		8	750	78
12		8	1000	116
13		9	500	52
14		9	750	89
15		9	1000	95
16		10	500	42
17		10	750	82
18		10	1000	103
19	5	8	500	54
20		8	750	84
21		8	1000	101
22		9	500	45
23		9	750	69
24		9	1000	112
25		10	500	60
26		10	750	84
27		10	1000	104
Average				79

Table 6 shows the results of the experiments for static instances in which it is assumed that all forms are available at the beginning of the planning horizon and that all processing times are known. In this case, we report the average ratio of time needed to complete the total amount of work over the system deadline given the resource profile from RAP_1 and RAP_2. As expected, RAP_1 always meets the system deadline and completes the processing of forms within 95 percent of the deadline on average. However, this is achieved at a higher resource cost. On average, it costs 5.4 percent more to obtain the additional resources used under RAP_1. Although RAP_2 produces mixed results, forms are completed after the deadline on average only 2.3 percent of the time . The experimental results verify the appropriateness of the lower and upper bound approach for system

Table 6 Performance of resource allocation models for static instances

			Static case		
			$\sum_{i=1}^{I} T_i / D$		Ratio of optimal costs
NO.	K	I	RAP$_1$	RAP$_2$	RAP$_2$/RAP$_1$
1		500	0.966	1.012	0.935
2	8	750	0.959	1.024	0.951
3		1000	0.957	1.037	0.938
4		500	0.947	0.977	0.939
5	9	750	0.938	1.025	0.965
6		1000	0.963	1.017	0.909
7		500	0.922	1.011	0.935
8	10	750	0.975	1.067	0.961
9		1000	0.925	1.039	0.979
Average			0.950	1.023	0.946

resource configuration.

6.4. Effects of Dynamic Arrivals and Feedback Loops on Performance

Now, we look at the effects of dynamic arrivals and feedback loops on system performance. Generating arrivals based on the three-year average of historical arrival patterns, we found, that the dynamic arrival of IRS forms does not dramatically change the performance of the system (see Table 7). Even though in some cases RAP$_1$ requires overtime to complete the process, the maximum amount of overtime needed is only 3.7 percent and on average RAP$_1$ meets the deadline. Regarding RAP$_2$, on average it requires only 3.5 percent more time to complete the forms. These results indicate that assuming complete availability of the forms is a good approximation approach and that one does not need to incorporate the dynamic characteristics of the tax forms explicitly.

Our results from examining the effect of approximating the feedback loops with the convolution of processing time distributions are presented in Table 7 under feedback loops. By comparing the ratios for feedback loops with Table 6, we see that the average ratio of the total completion time of RAP$_1$ and RAP$_2$ to the deadline remains almost unchanged. This indicates that our approximation for feedback loops performs well.

Finally, we look at the simultaneous effect of dynamic arrivals and feedback loops. The results are reported under the last two columns of Table 7. We notice that the average of the ratio increases to 1.044 for RAP$_1$ and to 1.062 for RAP$_2$, which means that employees need to work between 3.8

Table 7 Performance of resource allocation models under dynamic arrival and feedback loops

			Dynamic arrival		Feedback loops		Dynamic arrival and feedback loops	
			$\sum_{i=1}^{I} T_i/D$		$\sum_{i=1}^{I} T_i/D$		$\sum_{i=1}^{I} T_i/D$	
NO.	K	I	RAP$_1$	RAP$_2$	RAP$_1$	RAP$_2$	RAP$_1$	RAP$_2$
1		500	0.995	1.042	0.949	1.021	1.052	1.047
2	8	750	0.988	1.056	0.936	1.025	1.051	1.043
3		1000	0.996	1.042	0.947	1.013	1.020	1.081
4		500	0.988	1.018	0.957	1.016	1.052	1.072
5	9	750	1.018	1.057	0.977	1.011	1.037	1.070
6		1000	1.003	1.033	0.931	1.012	1.051	1.051
7		500	1.004	1.024	0.953	1.013	1.054	1.078
8	10	750	0.991	1.024	0.946	1.025	1.035	1.070
9		1000	0.980	1.017	0.946	1.008	1.048	1.045
Average			0.996	1.035	0.949	1.016	1.044	1.062

to 9.8 percent overtime. Since we assumed five working days per week and eight hours per day, these percentages translate to at most near four hours per week. Currently, the employees work almost every Saturday and at least one hour overtime on weekdays. This means a total of 13 hours of overtime per week, which is more than 5 times the overtime of our models. This difference shows how our models can help the company reduce its overtime expenses significantly.

To examine these simultaneous effects further, we used the Wilcoxon signed–rank test (Lawler et al. 1985) to test the null hypothesis that completion times obtained from our models and completion times obtained from simulation for dynamic arrival and feedback loops belong to the same population. Conducting the test with 270 observations and at a 5% level of significance, we could not reject the null hypothesis that the two populations were the same.

In summary, these numerical experiments indicate that the hierarchical procedure proposed for forming groups and allocating resources is an excellent approximation to the NP-hard monolithic problem. Moreover, our treatment of two important features of the process, dynamic arrivals and feedback loops, does not significantly affect the performance of our models and heuristics. Because the system is sufficiently congested, assuming that an estimate of the amount of work is available at the beginning of the planning horizon is not unreasonable. Also, using convolution of processing time distributions is a good approximation for the effect of feedback loops on the completion time.

7. Implementation

Directed by the vice president of Operations, the company implemented our models during the summer and early fall of 2010 as a part of ongoing process improvement efforts. Prior efforts at process improvement made the company receptive to our analytical solutions, and, after showing our initial analysis of the problem and the potential benefits to the company, we were given the green light to go ahead. Process managers were asked to fully cooperate with us and provide us with the necessary data. The VP of Operations proved critical to project success by initiating the implementation and removing internal obstacles as it proceeded. Her trust and full support made it possible to test our solutions in a real-world situation.

The managers were interested in evaluating and implementing the models in two phases. The idea behind phase 1 was to see how much the company could potentially benefit from the models without altering the workforce level. In phase 2, our models were implemented during the 2010 production season when the company could hire and relocate its employees.

Phase 1 entailed analyzing the benefits of optimally dividing the existing workforce into two designated groups: one for all state forms and one for all federal forms. We were given 2009 company data for the number of resources at each stage, M_k, and length of time it took to process each form at each stage, P_{ik}. Because P_{ik} already included all processing and reprocessing times, we did not apply our approximation procedure for feedback loops in this phase. We used the PLSP to allocate resources to these two predefined groups in order to minimize the total maximum amount of work needed to process all forms. Next, we incorporated the actual arrival pattern of forms in 2009 and the solution from the PLSP into a scheduling simulator that evaluated the total time needed to process the forms in each of the two groups. The simulator computed the project completion time based on the exact allocation of tasks to each resource at each stage. We compared simulator results with actual total amount of work in 2009 and found that the company could have reduced the total amount of work by 23.5%.

The objective of phase 2 was to fully implement our models throughout the 2010 tax season. For this purpose, we had to estimate the number of federal and state forms, the arrival pattern of the forms, and the processing times of forms at each stage. We also had to determine the number of groups to be formed. First, we used a linear regression model to estimate the number of federal and state forms to be processed for the 2010 tax year. To generate the arrival patterns of the forms, we used the average cumulative arrivals of forms over the last three years (2007-2009) and randomly assigned arrival dates to them such that their cumulative arrivals matched the historical percentage of arrivals.

To estimate processing time distributions at each stage, we designed a questionnaire (see Appendix 4). It requested estimates of the minimum, average, and maximum processing times needed to complete the forms at each stage. It also asked for the percentage of forms that take less than 25%, between 25% and 50%, between 50% and 75%, and above 75% of the maximum processing time. Having found that employees tend to inflate processing times, we asked managers to adjust and fine tune estimates based on their own judgment and experience. For each internal test and Final Test, we asked respondents to estimate the percentage of forms that fail testing and return for rework. After collecting the estimates, we followed the procedure in Section 3.2 to calculate the no-loop approximate processing times needed to complete the forms at each stage. At the end of the season, we found that the actual total processing times, $\sum_{i=1}^{I}\sum_{k=1}^{K} P_{ik}$, was 3.6% higher than our estimates.

To decide the number of groups, we discussed managerial aspects of organizing them, and the process managers elected to use two groups for processing all forms. We then used GRAP_1 and GRAP_2 to generate a range for the number of employees needed at each stage for each group. Because GRAP_1 and GRAP_2 provide lower and upper bounds, the company decided to make hiring and relocation decisions based on their average.

Using the new configuration of processing groups and comparing it with the 2009 resource deployment, the company relocated approximately 12% of the 237 employees to new job assignments and hired 8 new employees (3% increase in workforce). Company employees can be classified into three major skill categories: programmers, testers, and accountants. They cannot be relocated across the categories, but they can be relocated within them. Of the 12% who were relocated, 8.5% were relocated within the same skill category, and 3.5% came from the same skill category in a different product line.

To measure the savings obtained by implementing the hierarchical model, the company recorded cumulative overtime used prior to taking the software to market. Comparing the data with the 2009 tax year, the company found that it enjoyed a 25.7% reduction in overtime and an 11.3% reduction in the total workforce cost. The company also found that, even though hiring 8 new employees was costly, the hiring and relocation decisions helped the company reduce overtime payments which ultimately reduced the total workforce cost. When we considered the total amount of work before and after the first release (subsequent upgrades), the savings were even higher. In 2009, the total overtime was 59,439 hours, and the ratio of total overtime to regular hours was 22.5%. In 2010, the total overtime declined by 31.6% to 40,656 hours, and the ratio of total overtime to regular hours was 15.4%. The total workforce cost in 2010 was 13.6% lower than that of 2009, which roughly translates into $960,000. The company did not allow us to report the absolute value of workforce cost in 2009 and 2010, however. To the best of our knowledge, the financial savings were not the result of external factors such as demand change, employee turnover, change in skills, or changes in the structure of the software. In fact, the actual amount of work in 2010 was 1.8% higher compared with 2009, meaning that the company had to do even more work in 2010. Also, the structure of the software, its features, and interface did not change. Therefore, we can claim that the savings were the result of using our decision-making tools.

In addition to saving money and completing the software on time, our models helped the company resolve some long-standing disagreements among the functional managers. Managers were particularly amenable to the proposed solutions since they did not involve issuing any pink slips. Also, establishing two processing lines created some healthy competition to complete tasks earlier and with less overtime.

Motivated by the savings in overtime and total resource costs, the company has decided to implement the models every tax year. Managers are also considering expanding the modeling framework to other product lines in the future. One challenge to implementation was estimating processing times. In order to increase the accuracy of estimates used as inputs to the models, the company is currently setting up a system to better record processing times and rework iterations.

8. Concluding Remarks

We studied a software development process at a large company that faces a tight deadline for releasing its tax preparation software so that it retains its market share and avoids losses. Dynamic arrivals, variable processing times, feedback loops, and high task volume make process management a formidable undertaking. The same challenges are faced by many companies servicing other domains that require them to routinely upgrade their software applications.

We used an approximation to capture the effect of feedback loops on the completion time. Then we introduced a hierarchical framework to help the company manage the development process more effectively. The framework focused on sorting tax forms to dedicated groups and finding the right staffing levels to meet the release deadline. The computational experiments supported our modeling assumptions and attested to the excellent quality of the hierarchical framework and the solution procedures.

Implementing our models gave the company a 31% reduction in overtime. It also reduced the total workforce cost by 13% or around $1 million. The company successfully completed the software on time even though the amount of work to do was not less than the previous year. In the future, we would like to study other product lines in the company and possibly expand our models to manage more processes. Because some employees with certain skills (e.g., programmers) can work in different software development processes, a consolidated workforce management system can help the company more efficiently utilize its employees.

Acknowledgments

The authors sincerely thank the UCLA Harold and Pauline Price Center for Entrepreneurial Studies for supporting this research.

References

Adler, P., A. Mandelbaum, V. Nguyen, E. Schwerer. 1992. From project to process management in engineering: strategies for improving development cycle time. *Sloan School of Management,* MIT.

Ahmadi, R., H. Matsuo. 2000. A mini-line approach to pull production. *Eur. J. Oper. Res.* **125** 340–358.

Ahmadi, R., T. A. Roemer, R. H. Wang. 2001. Structuring product development processes. *Eur. J. Oper. Res.* **130** 539–558.

Asundi, J., S. Sarkar. 2005. Staffing software maintenance and support projects. *Proceedings of 38th Hawaii International Conference on System Sciences.* Hawaii, 1–8.

Brooks, F. P. 1975. *The Mythical Man-Month: Essays on Software Engineering.* Addison–Wesley, Reading, MA.

Browning, T. R., R. V. Ramasesh. 2007. A survey of activity network-based process models for managing product development projects. *Prod. Oper. Management.* **16**(2) 217–240.

Carrascosa, M., S. D. Eppinger, D. E. Whitney. 1998. Using the design structure matrix to estimate product development time. *ASME Design Automation Conference,* Atlanta, GA.

Cusumano, M. A. 1997. How MicroSoft makes large teams work like small teams. *Sloan Management Review.* **39**(1) 9–20.

Cusumano, M., A. MacCormack, C. F. Kemerer, B. Crandall. 2003. Software development worldwide: The state of the practice. *IEEE Software,* **20** 28–34.

Dawande, M., M. Johar, S. Kumar, V. S. Mookerjee. 2008. A comparison of pair versus solo programming under different objectives: An analytical approach. *Inf. Syst. Res.* **19**(1) 71–92.

Electronic Tax Administration Advisory Committee. 2011. Annual Report to the Congeress. Accessed on September 26, 2011 at `http://www.irs.gov/pub/irs-pdf/p3415.pdf`

Feng, Q., V. S. Mookerjee, S. P. Sethi. 2006. Optimal policies for the sizing and timing of software maintenance projects. *Eur. J. Oper. Res.* **173** 1047–1066.

Feng, Q., V. S. Mookerjee, S. P. Sethi. 2008. Application development using fault data. *Prod. Oper. Manag.* **17**(2) 162–174.

Hax A. C., D Candea. 1984. *Production and Inventory Managemnet.* Prentice–Hall, Englewood Cliffs, NJ.

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco.

Graves S. C., H. C. Meal, D. Stefek, A. H. Zeghmi. 1983. Scheduling of re-entrant flow shops . *J. of Oper. Management* **3**(4) 197–207.

Hos, C. J., K. G. Shin. 1997. Allocation of periodic task modules with precedence and deadline constraints in distributed realtime systems. *IEEE Transactions on Computers.* **46**(12) 1338–1356.

Iravani, F., S. Dasu, R. Ahmadi. 2011. A hierarchical framework for organizing a software development process. *working paper.* University of California at Los Angeles.

Ji, Y., V. S. Mookerjee, S. P. Sethi. 2005. Optimal software development: A control theoretic approach. *Inf. Syst. Res.* **16**(3) 292–306.

Joglekar, N. R., A. A. Yassine, S. D. Eppinger, D. E. Whitney. 2001. Performance of coupled product development activities with a deadline. *Management Sci.* **47**(12) 1605–1620.

Kekre, S., N. Secomandi, E. Sönmez., K. West. 2009. Balancing risk and efficiency at a major commercial bank. *Manufacturing Service Oper. Management.* **11** 160–173.

Krishnan, V., S. D. Eppinger, D. E. Whitney. 1997. A model-based framework to overlap product development activities. *Management Sci.* **43**(4) 437–451.

Kulkarni, V. G., S. Kumar, V. S. Mookerjee, S. P. Sethi. 2009. Optimal allocation of effort to software maintenance: A queueing theory approach. *Prod. Oper. Management.* **18**(5) 506–515.

Kumar, S., Y. Ji, S. P. Sethi, D. H. Yeh. 2006. Dynamic optimization of software enhancement effort. *Proceedings of 16th Workshop on Information Technologies and Systems (WITS).* Milwaukee, WI, 133–138.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinooy Kan, D. B. Shmoys. 1985. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Wiley & Sons, Chichester, NY.

Loch, C. H., C. Terwiesch. 1998. Communication and uncertainty in concurrent engineering. *Management Science.* **44**(8) 1032–1048.

Roemer, T., R. Ahmadi, R. Wang. 2000. Time-cost trade-offs in overlapped product development. *Oper. Res.* **48**(6) 858–865.

Roemer, T., R. Ahmadi. 2004. Concurrent crashing and overlapping in product development. *Oper. Res.* **52**(4) 606–622.

Shaw, M., P. Clements. 2006. The golden age of software architecture. *IEEE Software.* **23**(2) 31–39.

Smith, R. P., S. D. Eppinger. 1997. A predictive model of sequential iteration in engineering design. *Management Sci.* **43**(8) 1104–1120.

The 2010 Software 500. *Software Magazine.* Accessed on February 2, 2011 at `http://www.softwaremag.com/editors-desk/2010-software-500-another-good-year-for-outsourcers/`

Tavares, L. V. 1998. *Advanced Models for Project Management.* Kluwer Academic Publishers, Norwell, MA.

Terwiesch, C., C. H. Loch. 1999. Measuring the effectiveness of overlapping development activities. *Management Science.* **45**(4) 455–465.

Appendix 1. Proofs

Proof of Proposition 1. It is not difficult to see that the optimum schedule is a permutation schedule; i.e., the sequence of processing jobs is the same on all machines. An upper bound on the actual makespan is obtained by inflating the processing time of each job at each stage to its maximum processing time across the stages. In this case, the longest job and all the jobs after it are processed in stages $2, \cdots, K$ with no inserted idle time. Therefore, the makespan is equal to the sum of the processing times plus the transition of the longest job on machines $2, \cdots, K$, which is $\sum_{i=1}^{I} \max_k \{P_{ik}\} + (K-1)\max_i\{\max_k \{P_{ik}\}\}$. Note that the upper bound is independent of the sequence of jobs. $\square$

Proof of Proposition 2. We show that GRAP$_1$ and GRAP$_2$ are as hard as the 3-partition problem, which is known to be strongly NP-hard (Garey and Johnson 1979). Assume that we are given a general instance of the 3-partition problem consisting of an index set $A = (1, 2, ..., 3m)$, positive elements a_i for $i = 1, 2, ..., 3m$, and a positive integer B such that $B/4 < a_i < B/2$ and $\sum_{i=1}^{3m} a_i = mB$. We now introduce a specific instance of GRAP$_1$ and GRAP$_2$ as follows: $K = 1$, $G = m$, $I = 3m$, $P_{ik} = a_i$ for all i, k, $D = B$ and $w_k = 1$ for all k. We shall show that the optimal solution of GRAP$_1$ and GRAP$_2$ takes value m if and only if the $3m$ elements of A can be partitioned into m disjoint subsets $A_1, A_2, ..., A_m$ such that $\sum_{i \in A_r} a_i = B$ for $r = 1, ..., m$. If the 3-partition problem has a solution, then it is easy to see that the elements of each subset A_r could be assigned to one of the m groups and that the total amount of work for each group would be equal to D. In this case, each group would have one resource assigned to it: $Y_{kg} = 1$ for all k and g and $\sum_{k=1}^{K} \sum_{g}^{G} Y_{kg} = m$. If the 3-partition problem does not have a solution, then there is at least one subset A_r such that the total processing time in that group would be more than D. To meet the deadline, D, more than one resource would have to be assigned to this group. All other subsets would require one unit of resource, making the total resources greater than m. □

Proof of Proposition 3. We shall show that the recognition version of the GP is as hard as the 3-partition problem, which is known to be strongly NP-complete. Assume that we are given a general instance of the 3-partition problem consisting of an index set $A = (1, 2, ..., 3m)$, positive elements a_i for $i = 1, 2, ..., 3m$, and a positive integer, B, such that $B/4 < a_i < B/2$ and $\sum_{i=1}^{3m} a_i = mB$. We now introduce a specific instance of the GP as follows: $G = m$, $I = 3m$, $P_i = a_i$ for all i, $R_{i_1 i_2} = 0$ for all i_1, i_2, and $Q = B$. We shall show that the GP has a feasible solution if and only if the $3m$ elements of A can be partitioned into m disjoint subsets $A_1, A_2, ..., A_m$ such that $\sum_{i \in A_r} a_i = B$ for $r = 1, \ldots, m$. If the 3-partition problem has a solution, then it is easy to see that the elements of each subset A_r could be assigned to each of the m groups and that the total processing time of each group would be equal to B. If the 3-partition problem does not have a solution, then there is at least one subset A_r such that the total processing time of that group would be more than B and it is easily seen that the GP has no feasible solution. □

Proof of Proposition 4. We shall show that RAP$_1$ is as hard as the equal-size, equal-number-of-items partition problem, which is known to be binary NP-hard. Assume that we are given a general instance of the equal-size, equal-number-of-items partition problem consisting of an index set $A = (1, 2, ..., m)$, in which m is even and elements a_i for $i = 1, 2, ..., m$ are positive. Consider the following instance of RAP$_1$ as follows: $K = m$, and $P_{ik} = 2a_i$ if $i = k$; otherwise $P_{ik} = a_i$. Also, $D = \frac{3}{2}\sum_{r \in A} a_r$, $w_k = 1$ and $1 \leq Y_k \leq 2$ for all k. Finally, we consider an objective value of $D = \frac{3}{2}\sum_{r \in A} a_r$. Note that in any feasible solution to RAP$_1$, there exists a subset of stages, B, such that $Y_k = 2$ for all $k \in B$ and $Y_k = 1$ for all $k \notin B$. If there exists a partition, $B \subset A$, such that $\sum_{r \in B} a_r = \sum_{r \in A-B} a_r$ and $|B| = m/2$, then setting $Y_k = 2$ for all $k \in B$ and $Y_k = 1$ for all $k \in A - B$ generates a feasible solution to RAP$_1$ with the objective value of $D = \frac{3}{2}\sum_{r \in A} a_r$. If no partition exists and there is a solution to RAP$_1$ such that its objective value is less than or equal to $D = \frac{3}{2}\sum_{r \in A} a_r$, then it is easy to see that there should exist a subset $C \subset A$ such that $|C| < |A|/2$.

Setting $Y_k = 2$ for all $k \in C$, and $Y_k = 1$ for all $k \in A - C$, does not provide a feasible solution to RAP$_1$. $\square$

Proof of Proposition 5. We show that PLSP is as hard as the partition problem. Consider the following instance of PLSP as follows: $I_S = I_F = m$, $K = m$, $m_k = 3$, and $P_{ik} = 2a_i$ if $i = k$; otherwise, $P_{ik} = a_i$. Also, $D = \frac{1}{2}\sum_{r \in A} a_r$ and consider an objective value of $D = \frac{3}{2}\sum_{r \in A} a_r$. If there exists a partition $B \subset A$, such that $\sum_{r \in B} a_r = \sum_{r \in A-B} a_r$ and $|B| = m/2$, then in each feasible solution to PLSP, each stage in the federal or state process line will use either one or two resources. Let B be the set of stages in the state process line that use two resources and let $A - B$ be the set of stages in the federal process line that use two resources. Then, the total maximum amount of work is given by $2\sum_{r \in B} a_r + \sum_{r \in A-B} a_r$ for the state process line and $2\sum_{r \in A-B} a_r + \sum_{r \in B} a_r$ for the federal process line. Consequently, the objective value will be $\frac{3}{2}\sum_{r \in A} a_r$. Conversely, if no partition exists, we assume by contradiction that there is a solution to PLSP with the objective value of $\frac{3}{2}\sum_{r \in A} a_r$. Then, we would have:

$$2\sum_{r \in A-B} a_r + \sum_{r \in B} a_r \leq \frac{3}{2}\sum_{r \in A} a_r,$$

$$2\sum_{r \in B} a_r + \sum_{r \in A-B} a_r \leq \frac{3}{2}\sum_{r \in A} a_r,$$

which completes the proof. $\square$

Proof of Proposition 6. Let ϕ^f be the value of PLSP when $Y_{ks} = \left\lfloor \widetilde{Y}_{ks} \right\rfloor$, $Y_{kf} = \left\lfloor \widetilde{Y}_{kf} \right\rfloor$. Similarly, let ϕ^c be the value of PLSP when $Y_{ks} = \left\lceil \widetilde{Y}_{ks} \right\rceil$ and $Y_{kf} = \left\lceil \widetilde{Y}_{kf} \right\rceil$. Since the total maximum amount of work is non-decreasing in the number of resources, we can write:

$$\frac{\phi^H}{\phi^*} \leq \frac{\phi^f}{\phi^c} = \frac{\sum_{i=1}^{I_S} \max\{P_{i1}/\lfloor Y_{1s} \rfloor, ..., P_{iK}/\lfloor Y_{Ks} \rfloor\} + \sum_{i=1}^{I_F} \max\{P_{i1}/\lfloor Y_{1f} \rfloor, ..., P_{iK}/\lfloor Y_{Kf} \rfloor\}}{\sum_{i=1}^{I_S} \max\{P_{i1}/\lceil Y_{1s} \rceil, ..., P_{iK}/\lceil Y_{ks} \rceil\} + \sum_{i=1}^{I_F} \max\{P_{i1}/\lceil Y_{1f} \rceil, ..., P_{iK}/\lceil Y_{Kf} \rceil\}}.$$

For each i_s, define $\eta_i = \dfrac{\max\{P_{i1}/\lfloor Y_{1s} \rfloor, ..., P_{iK}/\lfloor Y_{Ks} \rfloor\}}{\max\{P_{i1}/\lceil Y_{1s} \rceil, ..., P_{iK}/\lceil Y_{Ks} \rceil\}}$. Also, let k_1 and k_2 be the indices of the stages that determine the maximum amount of work of form i in the numerator and denominator, respectively. If $k_1 = k_2$, then $\eta_i \leq 2$. If $k_1 \neq k_2$, then we have $P_{ik_1}/\lfloor Y_{k_1 s} \rfloor \geq P_{ik_2}/\lfloor Y_{k_2 s} \rfloor$, and $P_{ik_1}/\lceil Y_{k_1s}\rceil \leq P_{ik_2}/\lceil Y_{k_2s}\rceil \Rightarrow \lceil Y_{k_2s}\rceil \leq P_{ik_2}\lceil Y_{k_1s}\rceil/P_{ik_1}$. Thus, $\eta_i = P_{ik_1}\lceil Y_{k_2s}\rceil/P_{ik_2}\lfloor Y_{k_1s}\rfloor \leq \lceil Y_{k_1s}\rceil/\lfloor Y_{k_1s}\rfloor \leq 2$. The inequality also holds if η_i is defined for federal forms. Therefore, $\eta_i \leq 2$ for all forms which means $\phi^H/\phi^* \leq 2$. □

Appendix 2. An Example of the Shortest Path Algorithm for RAP$_1$

Figure 5 illustrates a small example of the network construction for RAP$_1$. Due to limited space, we only considered five stages and did not generate all the nodes to $Y^{\max}$. The data are shown in Table 8. Five forms with processing times are listed in columns two through six. The seventh column shows the cost of hiring one employee for each stage. The eighth and ninth columns show the lower and upper bounds for each stage. The deadline is 20, hence $\overline{Y} = 3.85$. The shortest path is shown with dashed arcs. Thus the optimal solution is $(3,4,4,4,4)$, which incurs 675 units of cost.

Table 8 Data for the network example

Stage	Form 1	Form 2	Form 3	Form 4	Form 5	w_k	Y_k^{Min}	$Y_k^{\max}$
1	10	12	10	14	6	25	3	8
2	14	9	10	17	7	30	3	7
3	14	10	13	12	11	45	3	6
4	13	16	15	10	5	35	3	7
5	12	13	14	7	15	40	4	7

Appendix 3. Questionnaire for collecting processing time estimates

Figure 6 shows the questionnaire we designed to obtain estimates of the processing time distributions and the percentage of forms that require rework after internal tests. We asked the same questions for all stages. Figure 7 shows the questions that we asked about Integration & Final Test to estimate the distribution of the destination of feedback loops from the final test as well as the percentage of forms that require one or more rounds of rework.

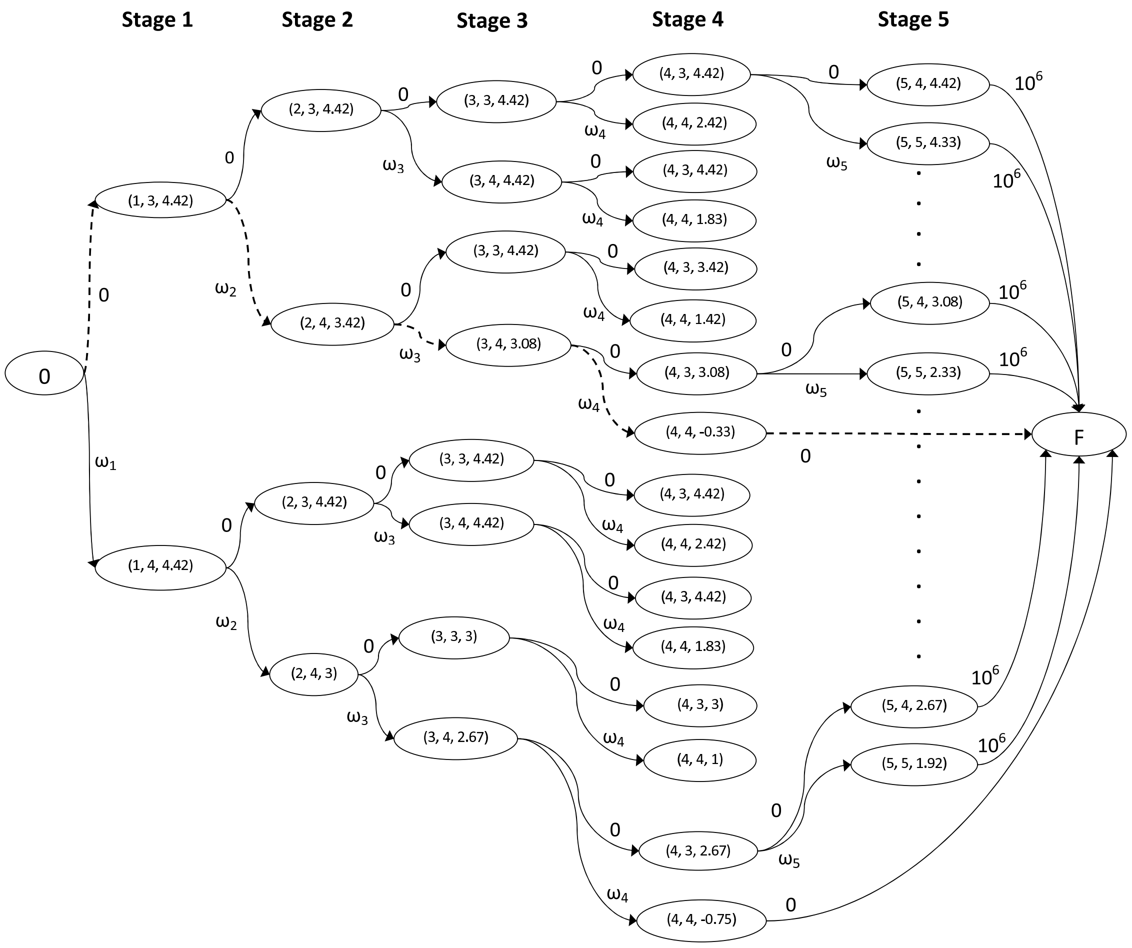
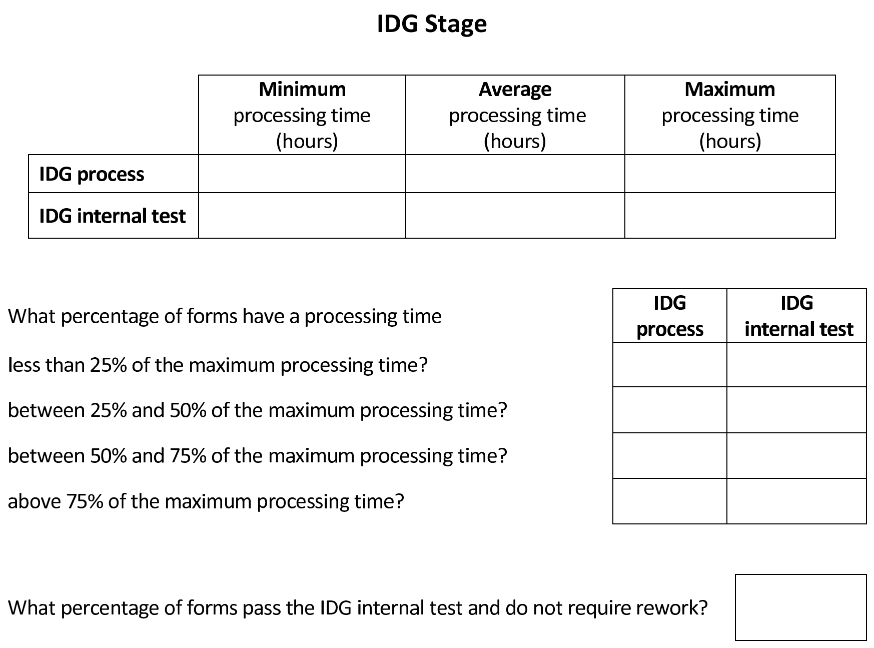


Figure 5 An example of constructing a network for RAP$_1$



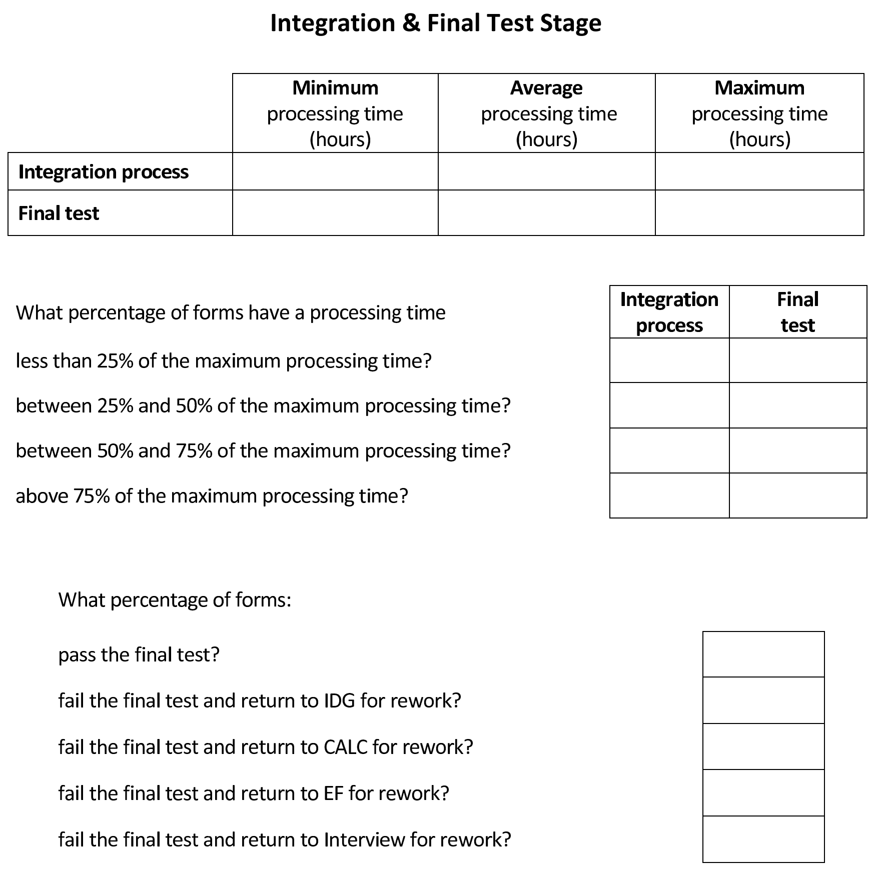
IDG Stage

	Minimum processing time (hours)	**Average** processing time (hours)	**Maximum** processing time (hours)
IDG process			
IDG internal test			

What percentage of forms have a processing time	**IDG process**	**IDG internal test**
less than 25% of the maximum processing time?		
between 25% and 50% of the maximum processing time?		
between 50% and 75% of the maximum processing time?		
above 75% of the maximum processing time?		

What percentage of forms pass the IDG internal test and do not require rework?	

Figure 6 Question for Estimating the Processing Time Distribution and Rework Probabilities at Each Stage



Integration & Final Test Stage

	Minimum processing time (hours)	**Average** processing time (hours)	**Maximum** processing time (hours)
Integration process			
Final test			

What percentage of forms have a processing time	**Integration process**	**Final test**
less than 25% of the maximum processing time?		
between 25% and 50% of the maximum processing time?		
between 50% and 75% of the maximum processing time?		
above 75% of the maximum processing time?		

What percentage of forms:

pass the final test?	
fail the final test and return to IDG for rework?	
fail the final test and return to CALC for rework?	
fail the final test and return to EF for rework?	
fail the final test and return to Interview for rework?	

Figure 7 Question for Estimating the Rework Probability Distribution for the Final Test