AMERICAN UNIVERSITY OF BEIRUT

OPTIMIZING THE EXCHANGE OF PARTIAL INFORMATION IN INTEGRATED PRODUCT DEVELOPMENT WITH MULTIPLE ACTIVITIES

by
JOE GERGES JABBOUR

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master in Engineering Management
to the Engineering Management Program
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
August 2012

AMERICAN UNIVERSITY OF BEIRUT

OPTIMIZING THE EXCHANGE OF PARTIAL INFORMATION IN INTEGRATED PRODUCT DEVELOPMENT WITH MULTIPLE ACTIVITIES

by

JOE GERGES JABBOUR

Approved by:

Dr. Ali Yassine, Associate Professor Engineering Management Program	Advisor
Dr. Bacel Maddah, Associate Professor Engineering Management Program	Advisor
Dr. Walid Nasr, Assistant Professor Suliman S. Olayan School of Business	Member of Committee

Date of thesis defense: August 28, 2012

AMERICAN UNIVERSITY OF BEIRUT

THESIS RELEASE FORM

I, Joe Gerges Jabbour

☐ authorize the American University of Beirut to supply copies of my thesis to libraries or individuals upon request.

☐ do not authorize the American University of Beirut to supply copies of my thesis to libraries or individuals for a period of two years starting with the date of the thesis.

_____
Signature

_____
Date

ACKNOWLEDGMENTS

First and foremost I offer my sincerest gratitude to my advisors, Dr Ali Yassine and Dr. Bacel Maddah, who have supported me throughout my thesis with their patience and knowledge whilst allowing me the room to work in my own way. I attribute the level of my Masters degree to their encouragement and effort and without them this thesis, too, would not have been completed or written.

I would also like to thank the Engineering Management Department at the Faculty of Engineering and Architecture at AUB for funding my research and study. Also the management at Consolidated Contractors Company who were flexible and understanding to the demands of the masters degree.

Finally, I thank my parents for supporting me throughout all my studies at University. It is through their uncountable sacrifices and their continuous encouragement that I was able to complete this work.

AN ABSTRACT OF THE THESIS OF

Joe Gerges Jabbour for Master in Engineering Management
Major: Engineering Management

Title: Optimizing The Exchange of Partial Information In Integrated Product Development With Multiple Activities.

Integrated product development (IPD) is the overlapping of nominally sequential activities while considering downstream concerns, in an effort to reduce the time-to-market of the product. The IPD practice helps companies to either pre-empt the competition and beat them to market, or respond quickly to changes in the market and quickly align their products with evolving customer needs.

Working with incomplete information, downstream activities are subject to the risk of rework; thus, entailing more development costs. In extreme cases these costs could surpass expected benefits. Prior research has shown that it may not be optimal to always consider partial information. On the other hand, research also showed that never considering partial information could entail high rework costs. Therefore, there is a need for an optimal policy to manage exchange of partial information in an IPD environment.

In this thesis, we formulate a dynamic programming model to manage upstream partial information flow in a multi-activity IPD process ($m > 2$ activities). The multi-activity aspect is novel as all previous research considers two-activity models. We then resort into solving the dynamic program directly and performing an extensive Monte Carlo simulation study to analyze the behavior of the optimal policy. The simulation results suggest several important insights regarding the timing and frequency of considering partial information in an IPD environment. The study showed that upstream activities would consider more information in IPD environments, and they do so earlier. Most notably, we observe a reverse bullwhip effect in IPD environments where the effect of variability of information is dampened rather than amplified downstream. Finally, we present a decomposition heuristic to easily approximate the decision policy, by solving a sequence of two-activity models. The heuristic performs very well yielding near-optimal results at significantly lower computer storage requirement. This enhances the applicability of the research to real-world problems involving a large number of activities.

CONTENTS

ILLUSTRATIONS

Figure Page

TABLES

CHAPTER 1

INTRODUCTION AND MOTIVATION

In today's fast moving markets, one of the major factors affecting a company's competitive edge is the time it requires to develop new products. The shorter the time-to-market of the new products, the faster the company is able to respond to changes in the market and the better it can preempt the competition in an effort to acquire market share.

The development process of any new product is made up of a set of diverse activities. Traditionally, these activities would be done sequentially where the first activity would complete its tasks and then send complete information to mid-stream activities. These mid-stream activities would rely on this information to do their tasks and then send complete information downstream. Downstream activities would then use this information to complete their tasks. Figure 1 shows an example of a three-activity product development process, where the first activity is market research (A) , the second is product Design (B) and the third is Process Design (C).

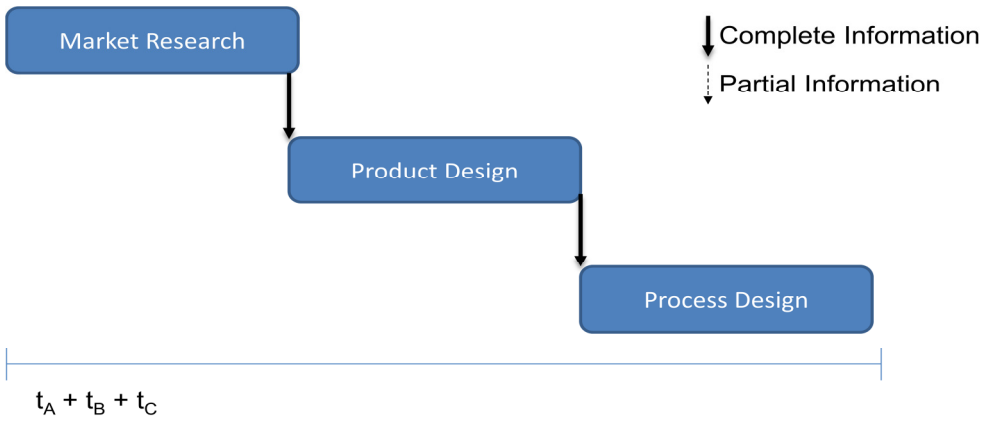

Figure 1: Sequential Product Development process with 3 activities.

The total time-to-market of this product would be $t_A + t_B + t_C$ which is the time to complete all the activities sequentially. There are two ways that the time-to-market could be reduced. The first way is to crash the three activities, by adding many resources to them in an effort to reduce the time of every activity individually, and thus decreasing the total time required. This method could become costly in regards to the number of resources required and the correct management of these resources. The second approach, as shown in Figure 2, would be to overlay these activities and thus reduce the time-to-market of the product to $t_A + t_B + t_C - t_{A\cap B} - t_{B\cap C} + t_{rework}$,where - $t_{A\cap B}$ and $t_{B\cap C}$ are the overlap times of A&B and B&C respectively. In this overlap setting activities B and C would have to work with partial information, and then suffer rework costs as information is changed upstream.

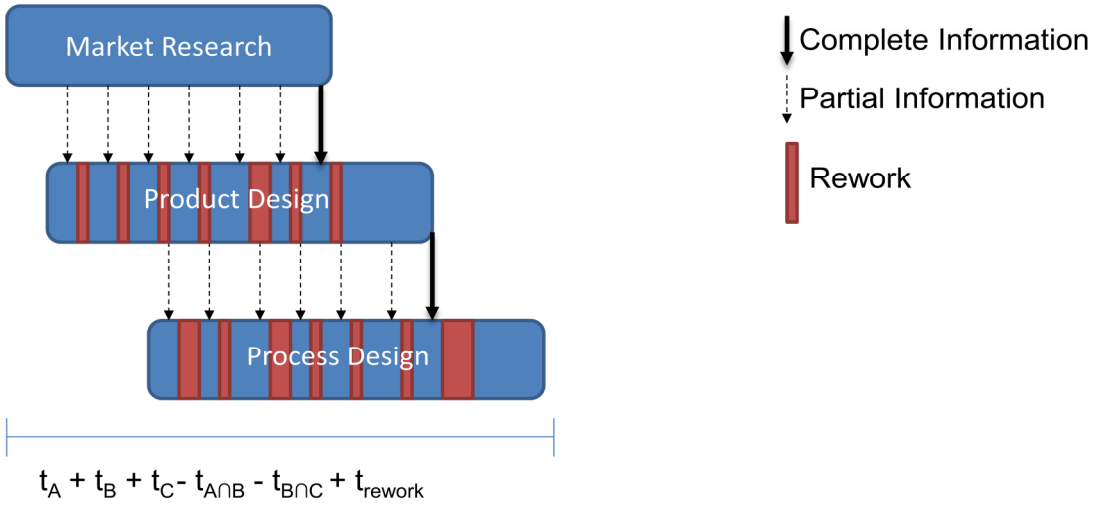

Figure 2: Integrated Product Development with three activities.

Integrated Product Development (IPD) is the overlaying of nominally sequential activities in an effort to decrease the time-to-market of a product. Overlapping will hence cause activities to work with incomplete information and hence will induce rework. Considering the worst case scenario is when the total integrated time would be

greater than the sequential time. That is why a decision policy is needed to indicate when integration is profitable and if it is profitable to manage the exchange of information in an optimal manner thus reducing the time-to-market of the product. Since overlapping and interaction increase the need for coordination among activities, IPD compensates using organizational mechanisms (such as cross-functional teams) and technical mechanisms (such as PIM/PDM tools).[1] The influence of organizational mechanisms on integration and coordination have been studied extensively in the product development literature (e.g., Hauptman and Hirji, 1999; McDonough, 2000; Hoegl et al., 2004); however, rigorous analysis concerning technical mechanisms within a development environment is scarce, despite the consensus on its pivotal role (Nambisan, 2003; Yassine et al., 2004; Banker et al., 2006; Nambisan, 2009).

In IPD environments, individual decisions are not made in isolation, but are impacted by information generated and consumed by other development participants. When an individual participant reacts to newly arrived information, it will modify the requisite information for other dependent participants. This will create a complex chain of interdependencies, where the decision of a single participant has the potential to propagate throughout the development organization involving many other participants (Yassine et al., 2003). We specifically consider the archetypal scenario where a development participant is capable of accessing, at any time, unreliable, but related, development information, which has the potential to change as the development endeavor progresses. The participant has to decide what the appropriate action should be

[1] PIM stands for Product Information Management, which is also known as PDM (Product Data Management). PIM software involves a database system that manages product-related information including engineering drawings, project plans, assembly diagrams, product specifications, analysis results, correspondences, bills of material, and many others (Liu and Xu, 2001). They are successful at managing the access and control of finalized information; however, they fail to handle the evolving nature of incomplete information that characterizes IPD environments; particularly at the early stages of development (Banker et al., 2006; Bardhan, 2007).

in response to this information. For instance, it can choose to ignore the information and continue with its original mission, or, it can incorporate the information to modify its work appropriately in light of this new, but partial, information. The trade-off involved here is that acting upon such information may improve the quality of its work; however, there is a risk of disrupting its progress (to check this partial information) and then discovering that this newly available information is irrelevant.

In this thesis, we consider an multi-activity, $m \geq 2$, development scenario where each activity has the option of either to consider the partial information being fed to it by upstream activities or to disregard it. The trade-off is between disrupting the flow of the activity and incorporating information that may be valid or not, and postponing the processing of information to face more costly rework later. We formulate a multi-activity DP to develop the optimal decision policy for all possible stages, in an effort to minimize the overall IPD completion time by controlling the amount of rework performed, taking into consideration the quality of partial information exchanged and the cost of performing rework. We then resort to simulation to study the dynamics of the multi-activity model and draw out a set of insights. The study showed that upstream activities would consider more information in IPD environments, and they do so earlier. Most notably, we observe a reverse bullwhip effect in IPD environments where the effect of variability of information is dampened rather than amplified downstream.

The remainder of this thesis is organized as follows. In Chapter 2 we briefly review some of the literature related to this topic. In Chapter 3 we present our multi-activity formulation and in Chapter 4 we simulate different scenarios of an IPD model using the optimal decision policy of chapter 3 to draw out a set of insights that could be useful in a regular working environment. In chapter 5 we present a decomposition heuristic that

gives a near-optimal decision policy with minimal additional rework. Finally we conclude in chapter 6 by summarizing our work and proposing future extensions.

CHAPTER 2

LITERATURE REVIEW

There has been extensive research on concurrent engineering and integrated product development. The work done has been along two directions. The first direction is towards determining the optimal amount of overlap between tasks to reduce development lead time. This work was led by Krishnan et al. (1997), Terwiesch (1998) and Joglekar et al. (2001). The second direction is towards determining the optimal communication policy. Our work is along this second direction to complement the efforts started by Ha and Porteus (1995), Yassine et al. (2008, 2012) and Lin et al. (2010).

Krishnan et al. (1997) investigated the overlapping of two nominally sequential dependent Product Development activities. They developed an overlapping framework for these activities based on a downstream rework formulation that depends on upstream information evolution and downstream sensitivity. The two constructs, upstream evolution and downstream sensitivity are at the heart of the IPD study and are extensively used in this thesis. Upstream Evolution is used to refer to the refinement of the upstream generated information, from its preliminary form to a final value (Krishnan et. al 1997). i.e. the way information changes from its initial form to a final complete form to be passed down at the end of the upstream activity. Downstream sensitivity is the relationship between the duration of downstream rework and the magnitude of the change in the upstream information value (Krishnan et. al 1997). Activities are considered highly sensitive to upstream changes in information when small changes in the upstream information leads to large rework downstream. On the contrary, when

large changes in upstream information leads to small rework duration, this means that the downstream activity is slightly sensitive to changes in upstream information.

Loch and Terwiesh (1998) offered a two activity analytical model for overlapping activities to minimize the time-to-market. They derived an optimal communication policy affected by the uncertainty of changes in information generated by upstream activities. The information changes upstream are released to downstream activities as batches of data that would induce rework on the downstream tasks. The model also took into account the dependence of the downstream activity on the upstream activity. The occurrence of these information changes was modelled as non-stationary Poisson arrival process with a variable rate defined along the duration of the upstream phase. This non-linear program had three decision variables: the pre-communication intensity, the expected communication frequency and the amount of overlap. Pre-communication intensity is the total number of information exchange meetings that are planned to be held during the development process. Expected communication frequency is the expected rate of occurrence of these meetings. The objective function assess the benefits of the overlapping period against the additional cost of rework generated from uncertainty in the information and the cost of frequent meetings.

Lin (2010) followed a similar approach by assuming a non-homogeneous Poisson process for the upstream changes occurrences and the dependency function. The proposed model is also a non-linear program (NLP). However, in this case there are two decision variables: start time of downstream work and functional interaction duration. To derive estimates of the rates of the non-homogenous Poisson process, project engineers first examined the documents which contain all the details of the changes, such as the root causes, the severity, and the closure date. After that, project engineers

jointly estimated the rates of engineering changes for different points in time. Similarly, the rework impact on downstream rework was also estimated using interviews with experienced engineers in the company.

Yassine et al. (2008) defined four forms of upstream evolution. These forms are shown in Figure 3. Concave evolution is when activities evolve the most during early stages of activity's duration. Convex evolution is when the activities evolve the most during late stages. Linear evolution is when an activity evolves at a constant pace al throughout its lifetime. Finally, S-shaped evolution is when an activity evolves the most in the middle stages. In the same paper, they developed a dynamic programming model that provides optimal timing and frequency of information exchange for a single activity in order to minimize it development cost. The model considers discrete times divided into equal periods where the upstream activity would send information. At the beginning of each period the downstream activity is faced with the decision of either considering or not considering the information. If the new information is used then a rework penalty is incurred. Otherwise, the team would proceed with its normal course of development. Then, the trade off becomes how to divide the rework packages along the detailed design time-line such that the total development cost is minimized.

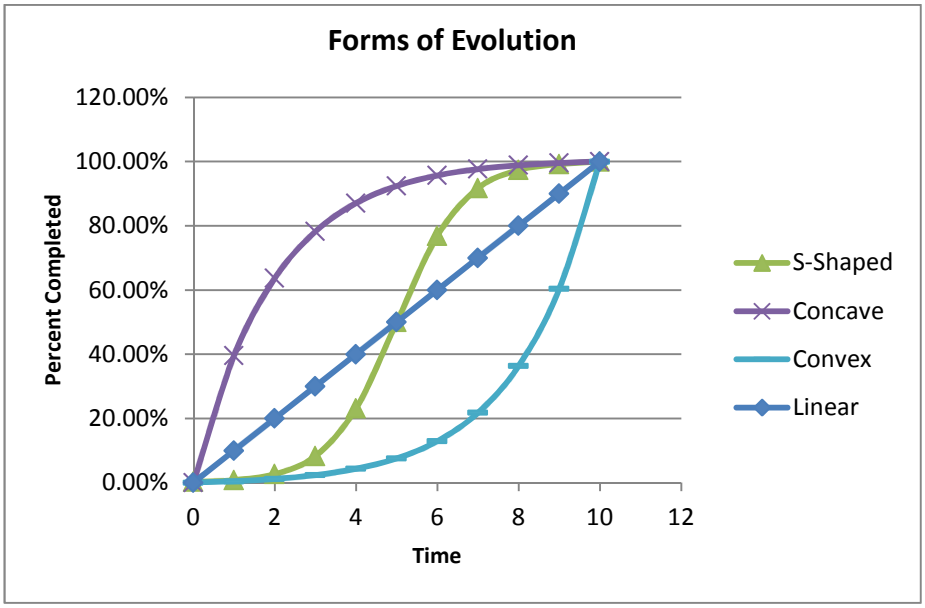

Figure 3: Forms of evolution as described by Yassine et. al (2008)

In a sequel paper, Yassine et al. (2012), considered uncertainty in the quality of exchanged information and derived a threshold policy for the two activity model. They also analyzed a three-activity model. However, the complexity of the three activity model made it difficult to find a threshold solution. They resorted to simulation, instead, in order to understand the dynamics of the problem. The simulation was done in two steps, first all possible scenarios of the integrated model are enumerated for given parameters and stored in memory. In the second step, the interaction between the midstream activity (B) and the downstream activity (C) is simulated using values obtained from the first step to decide on whether to consider the information or not. The simulation resulted in three insights: (i) Activity B considered more information in the integrated model than in isolation in an effort to help (C) in reducing its rework; (ii) Activity (B) in the integrated model tends to consider more information earlier in order to improve the quality of the information as early as possible; and (iii) when the fixed costs of considering information for either activity (B) or (C) were increased up to a

certain extent where the integration became costly and prohibitive, which implies that (B) behaves the same in integration and isolation. In this thesis, we extend the model from a 3 activity model to an n-activity model. The extended model adds integer formulation to the DP, making it flexible and extendable to n-activities. We have also changed the form of the model parameters linking them to upstream evolution and downstream sensitivity. The most important contribution in our study in integrated product development, is that our model accounts for downstream concerns. All models in the literature have not included this major part in their model's frame. This the main feature that distinguishes the overlapping models presented in earlier studies with the integrated product development that we present in this thesis. We also confirm the insights presented in Yassine et al. (2012) for the four activity model over a wider set of parameter changes. We draw additional insights on the dynamics and behaviour of the three and four activity model.

CHAPTER 3

MODEL AND ASSUMPTIONS

In this chapter, we develop a dynamic programming model to minimize the overall time of the IPD process of *m* activities. The model is similar to the DP formulation devised by Yassine et al. (2012), yet it is extended to *m* activities. In section 3.1, we develop the function describing the quality of upstream information for any activity *k,* $p^k(i, \boldsymbol{r^{k-1}})$. In section 3.2 we introduce the function defining the variable amount of rework performed by activity *k,* in state r^k, $\alpha^k(i, r^k)$. Finally, we present our multi-activity DP model in section 3.3.

All activities can be executed sequentially after the completion of the preceding activity, in which case the overall duration of the development process becomes the sum of the nominal durations of all activities. Alternatively, as we assume in this thesis, in an integrated development environment, all downstream activities can be executed concurrently, over a given time interval related to upstream activities completing a minimum amount of work, as discussed in Yassine et al. (2012). We divide the duration of any activity into two parts: "nominal" activity duration and the "rework" activity duration. The nominal duration of an activity is the time needed for the activity to complete its assigned work assuming that it is either independent of all other activities (i.e., does not need information from other activities), or all its requisite information is available at its start time. The rework duration is the extra time needed to perform rework in case the activity starts with missing or incomplete requisite information. The nominal duration of an activity is assumed to be known and fixed; however, rework

duration is uncertain and depends on the fraction of nominal work performed prior to the arrival of the incomplete information. Thus, the minimization of the completion time of the development process is equivalent to the minimization of the cumulative rework durations performed by the downstream activity. Figure 4 shows a diagram of the proposed model for $m = 4$ activities.

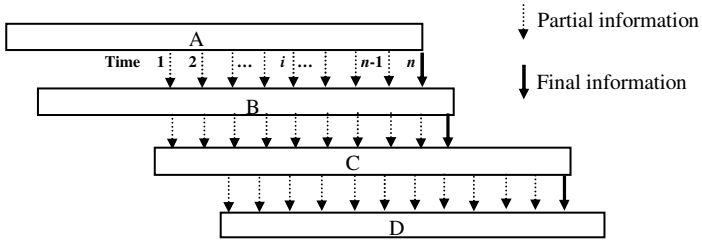

Figure 4: IPD model with $m = 4$ activities

We adopt a discrete time framework with n time periods. The upstream activities continuously (i.e., at every time interval $i = 1, \ldots, n$) send information to the downstream activities. The downstream activities have the ability to consider the information sent or to ignore it. At every time i, the decision maker has two options: either not to consider the information at this time which will lead to an accumulation of rework and costly rework at later time, or to consider the information at this time with a risk of rejecting it (i.e., not performing rework) due to low information quality. It is mandatory that at the last time interval (denoted by n) the decision maker considers the information in order to finalize the integration of the downstream activity with the upstream activity. This assumption is legitimate since the relation between all activities is of the finish-to-start type. When an activity decides to consider information, it is incurred a fixed cost. This cost reflects the preparations needed to consider the information. For example, when an engineer does some changes in the design of a chip, he sends the new CAD files to be considered by manufacturing. The costs incurred by the manufacturing department to

look over these CAD files and see if there are any changes to be made is the fixed cost of considering information. We denote f^k, the fixed cost for considering information by activity k, $1 \leq k \leq m$. Whenever information is considered, there is a chance that the exchanged information is not valid and does not induce rework to the activity that is considering the information; on the contrary the considered information could induce rework. The probability defining the quality of the data and its ability to induce rework is denoted by $p^k(i, \boldsymbol{r^{k-1}})$, being the probability that the information available at time i for activity k is valid. This probability is also a function of the vector $\boldsymbol{r^{k-1}} = \{r^{k-1}, r^{k-2}, \dots\dots\dots\dots r^2\}$ where r^j denotes the last time activity j performed rework. A detailed explanation of the function $p^k(i, \boldsymbol{r^{k-1}})$ is given in the next section (3.1). In the case where the information is valid, the amount of rework performed could be divided two parts: fixed costs of performing rework, denoted by β^k for any activity k, and a variable cost component denoted by $\alpha^k(i, r^k)$. The variable cost of rework for activity k, $\alpha^k(i, r^k)$ is dependent on the time i and $r^k$the last time the activity performed rework. The function $\alpha^k(i, r^k)$ is described in detail in section 3.2.

3.1 Quality of Exchanged Information $p^k(i, \boldsymbol{r^{k-1}})$

The earlier the downstream activities consider upstream information, the less rework is required. However, considering the information at early stages has a higher probability of rejection since the quality of information might not be high enough to justify performing rework.

The function $p^k\left(i, \boldsymbol{r^{k-1}}\right)$ of any activity k refers to the probability that the information provided by upstream activities is in its finalized form and will not be changed. Naturally, $p^k\left(i, \boldsymbol{r^{k-1}}\right)$ would depend on the activities originating the

information, mainly activity $k-1$, if we are assuming sequential processes, and is not affected by activity *k*. We start our definition of the function $p^k(i, \boldsymbol{r^{k-1}})$ by considering that k only receives information from $k-1$ and that $p^{k-1}(i, \boldsymbol{r^{k-2}})$ is known. Any activity has two main sources of information: Endogenous information and exogenous information, ie information supplied from the upstream activity $k-1$. Endogenous information is the part of information that is available at time 0. $p^k(i, \boldsymbol{r^{k-1}})$ also depends on the exogenous information, that is influenced by the evolution of the activity $k-1$; as such one would expect that the quality of exogenous information to increase at the same rate as the evolution of the activity $k-1$. We would assume that $p^k(i, \boldsymbol{r^{k-1}})$ would be proportional to evolution function of activity $k-1$. Figure 5 shows the sources of information of any activity in the IPD process, Endogenous information is fed to the activities at time 0 and partial information is sent downstream at all time periods.

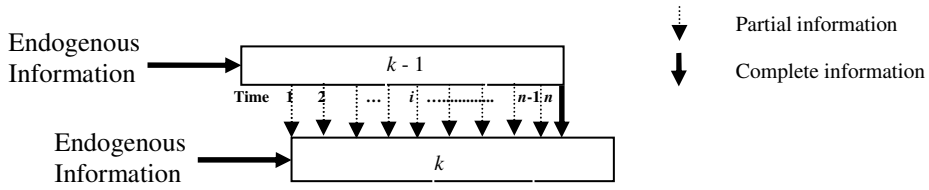

Figure 5: Sources of information for activity *k*.

The above definition of $p^k(i, \boldsymbol{r^{k-1}})$ is correct if activity $k-1$ is the first activity and it is only creating information. On the other hand, midstream ($k-1$) activities start their work with preliminary information, and will have to work with information that is still not finalized. If midstream activities do not consider any information or do not do any rework all throughout their evolution, they are not able to achieve a full evolution. This means that the quality of information they generate will deteriorate if the activity misses information from upper stream activities ($k-2$). Midstream activities could

enhance the quality of their information by increasing the frequency of considering information from upstream activities. That is why a more adequate form of $p^k(i, \boldsymbol{r^{k-1}})$ for activities receiving information from middle stream activities, would have to take into account the evolution of the activity $k-1$ and the last time $k-1$ did rework. It would also have to take into consideration the quality of information supplied to the midstream activity. The quality of the information that $k-1$ is using is a function of the information being supplied by the activity $k-2$, $p^{k-1}(i, \boldsymbol{r^{k-2}})$ to $k-1$ and the last time activity $k-1$ received good information, denoted by r^{k-1}. We suggest the following form of the probability function $p^k(i, \boldsymbol{r^{k-1}})$:

$$p^k\left(i, \boldsymbol{r^{k-1}}\right) = g\left(\mathcal{E}_i^{k-1}\right) * \left(\varsigma^{k-1} + (1-\varsigma^{k-1})\, p^{k-1}\left(r^{k-1}, \boldsymbol{r^{k-2}}\right)\right) \qquad (1)$$

where $g\left(\mathcal{E}_i^{k-1}\right)$ is an increasing function of the evolution of the activity $k-1$, and ς^{k-1} is the fraction of endogenous valid information available to activity $k-1$. The two functions are multiplied to mimic the fact that *k-1* would improve the quality of its information as it evolves. The function $\mathcal{E}_i^{k-1}$ is the evolution of activity $k-1$. It is a normalized function and belong to the range [0,1].

The function $\mathcal{E}_i^{k-1}$ could have a form of any of the evolution functions discussed in Chapter 2. See Figure 3 in that chapter. These forms are adopted from Yassine et al (2008). The degrees of convexity or concavity of the functions below could be varied to match the IPD situation that the decision maker is faced with.

The function $p^k(i, \boldsymbol{r^{k-1}})$ is an increasing function in r^{k-1} and i. For example, if Activity C gets its information from the upper stream Activities B and A having a convex evolution then we would have the following forms of $p^C(i, \boldsymbol{r^B})$ as a function

of time i and r^B the last time B did rework. In this case the vector $\boldsymbol{r^B}$ reduces to r^B and hence $p^C(i, r^B)$ has the a shape similar to the one shown below in Figure 6. In Figure 6 we show different curves for $p^C(i, r^B)$ as a function of time form $5 \leq i \leq 10$ and $0 \leq r^B \leq 4$. We plot these values to show the impact of time and the last time B considered information on the quality of information used by C.

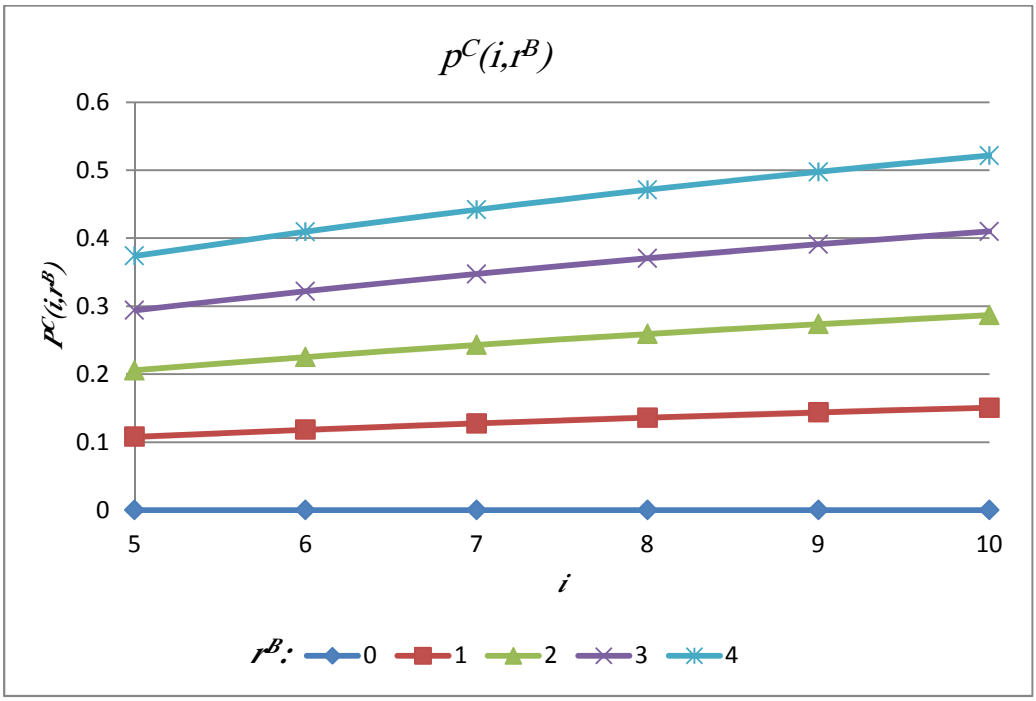

Figure 6: Form of $p^C(i,r^B)$ as a function of i and r^B.

We can see that the function increases in both i and r^B. If B never considers information r^B would always be 0.

3.2 Potential rework $\alpha^k(i, r^k)$

Delaying the evaluation of upstream information carries some penalty due to (i) the increase of *nominal* work complexity as the development evolves, and (ii) the increase in the *rework* complexity of the unfinished work performed downstream

without input from upstream, as delaying the consideration of information will increase the backlog of activities requiring input from upstream. In other words, the more we delay considering the information, the more we delay potential rework, and thus the more rework we have to perform due to the increase in complexity of both the nominal work and rework. Therefore, the potential rework function at time *i*, $\alpha^k(i, r^k)$ is divided into two components:

- Rework caused by the increase of nominal work complexity $\mathrm{h}^k(\mathrm{E}_i^k)$:

This component of the rework is a function of the time *i*; reflects the weight of the finished work on the rework to be done. As *i* increases more work is done and hence when new information is considered at later stages more rework has to be performed. The function $\mathrm{h}^k(\mathrm{E}_i^k)$ is increasing in *i* since one would assume that more work done would require a larger amount of rework. The function $\mathrm{h}^k\left(\mathrm{E}_i^k\right)$ is best interpreted as function of the evolution of the Activity *k*. The Figure below illustrates the need for the ideas discussed above. In the Figure 7, Case 1 is when a downstream activity does rework at later stages. Alternatively, case 2 is when the same activity does rework at an early stage. The amount of work completed is reflected by the size of the bar.

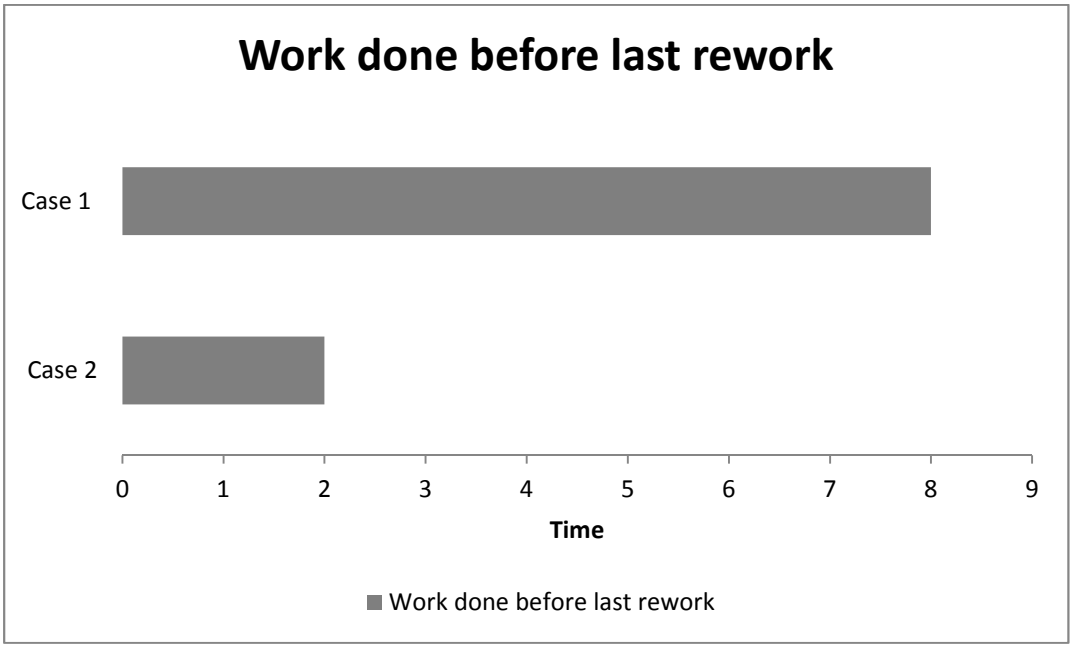

Figure 7: Effect of amount of work completed on rework.

- Rework caused by the increase in the *rework* complexity of the unfinished work performed downstream without input from upstream $\gamma^k\left(\varsigma^k, \mathcal{E}_i^{k-1} - \mathcal{E}_{r^k}^{k-1}\right)$

$\gamma^k\left(\varsigma^k, \mathcal{E}_i^{k-1} - \mathcal{E}_{r^k}^{k-1}\right)$: is a function of the time i, the amount of endogenous information and the last time rework was done r^k. This factor helps the model differentiate between two states where the process has just done rework and where the process has not done any rework in a long time. As such the function $\gamma^k\left(\varsigma^k, \mathcal{E}_i^{k-1} - \mathcal{E}_{r^k}^{k-1}\right)$ should reflect the amount of accumulated rework as a result of not considering information for a long period of time. Figure 8 illustrates the two states discussed above. Case 1 shows the amount of rework, the white portion, accumulated from period 8, the last time the activity did rework. Case 2 shows the amount of rework, the white portion, accumulated from period 2, the last time the activity did rework.

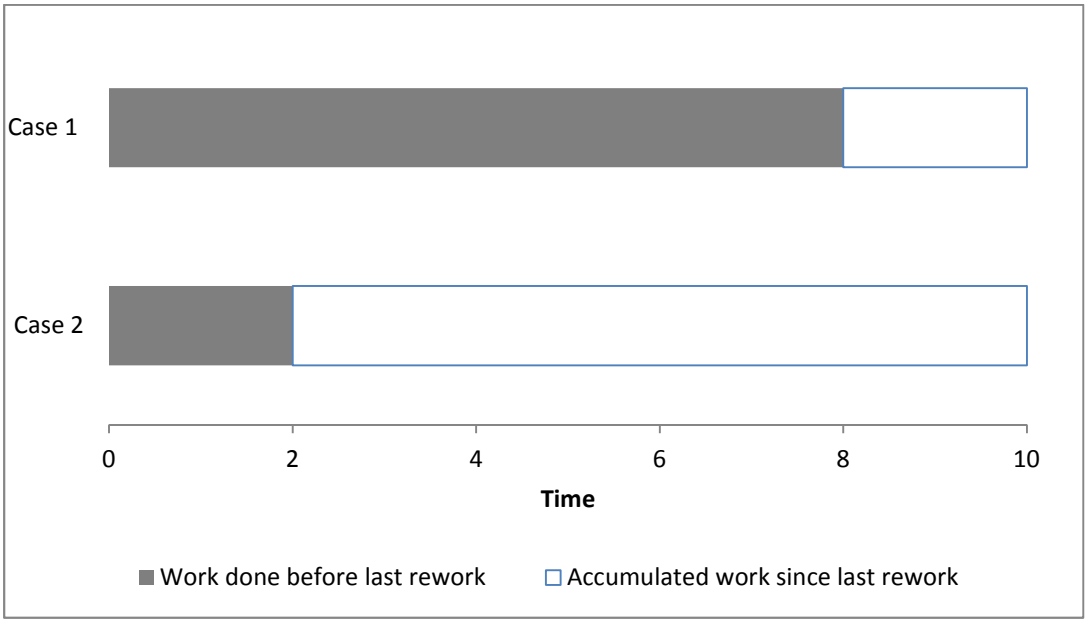

Figure 8: Effect of amount of information missed on rework.

The function $\gamma^k\left(\varsigma^k, \mathcal{E}_i^{k-1} - \mathcal{E}_{r^k}^{k-1}\right)$ is then increasing with the amount of information missed. The rate at which this function increases is related to the sensitivity of the function to information sent from upper-stream activities. The sensitivity of the downstream activity is influenced by the type of the activity and by the proportion of information that the activity takes from exogenous sources. That said, one would assume that as the amount of endogenous information increases downstream activities would become less sensitive to upstream information changes. An activity that has all the information that it needs at time 0 would be insensitive to changes in upper-stream activities as it is not affected by them. When $\gamma^k\left(\varsigma^k, \mathcal{E}_i^{k-1} - \mathcal{E}_{r^k}^{k-1}\right)$ increases rapidly then a small change in the information upstream leads to large rework downstream. In this case Activity k is highly sensitive to upstream changes. Alternately, when $\gamma^k\left(\varsigma^k, \mathcal{E}_i^{k-1} - \mathcal{E}_{r^k}^{k-1}\right)$ increases slowly then large changes in information upstream induces small amount of rework. In this case, activity k is said to be slightly sensitive to changes upstream. At any point the largest amount of rework would be $\mathrm{h}^k(\mathcal{E}_i^k)$, which

means that all work that has been done needs to be redone again. As such a natural form of the function $\gamma^k\left(\varsigma^k, \varepsilon_i^{k-1} - \varepsilon_{r^k}^{k-1}\right)$ would belong to the range [0,1].

Thus the function $\alpha^k(i, r_k)$, that quantifies the amount of rework done whenever valid information is communicated to downstream activities is:

$$\alpha^k(i, r^k) = \mathrm{h}^k(\varepsilon_i^k) \times \gamma^k\left(\varsigma^k, \varepsilon_i^{k-1} - \varepsilon_{r^k}^{k-1}\right) \qquad (2)$$

As discussed earlier the function $\mathrm{f}^k(\varepsilon_i^k)$ would be a function of the evolution function of the activity *k*, it could have any of the forms discussed in Yassine et. al (2008). See Figure 3 in chapter 2.

The function $\gamma^k\left(\varsigma^k, \varepsilon_i^{k-1} - \varepsilon_{r^k}^{k-1}\right)$ would be an increasing function in the amount of information that has been accumulating from the last time the activity *k* did rework. As discussed earlier this is related to the rate of evolution of the activity $k-1$. As such at any point in time $\gamma^k\left(\varsigma^k, \varepsilon_i^{k-1} - \varepsilon_{r^k}^{k-1}\right)$ is increasing in $\varepsilon_i^{k-1} - \varepsilon_{r^k}^{k-1}$. This function is also decreasing in ς^k. The Figures below show possible forms for the function $\gamma^k\left(\varsigma^k, \varepsilon_i^{k-1} - \varepsilon_{r^k}^{k-1}\right)$ in two cases $\varsigma^k = 0$ (Figure 9) and $\varsigma^k = 25\%$ (Figure 10).

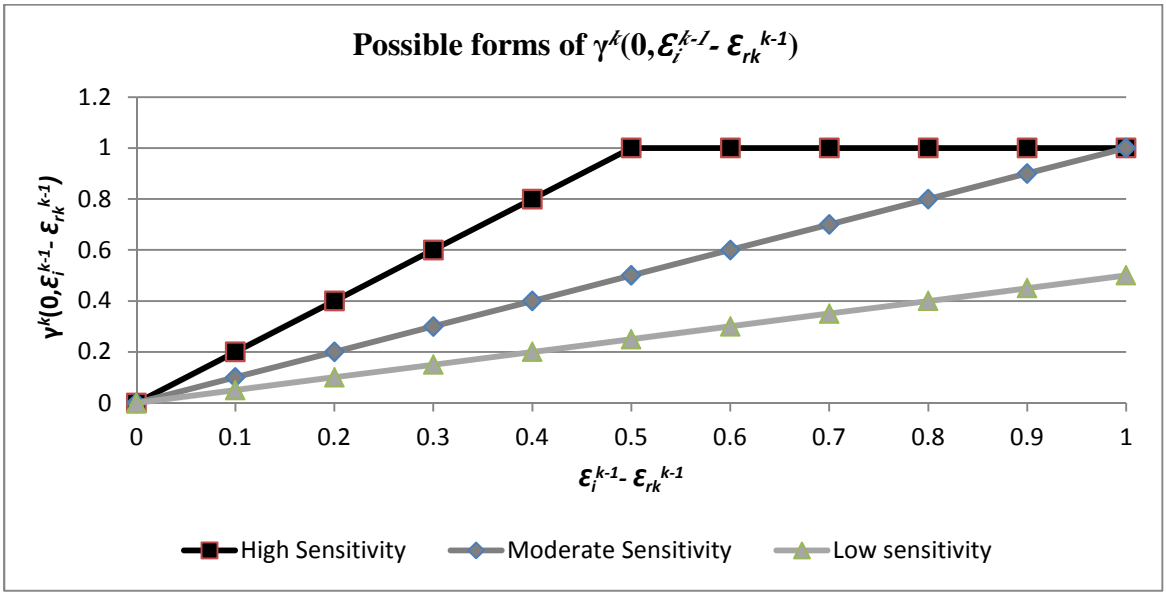

Figure 9: Possible forms of the sensitivity function $\gamma^k\left(\varsigma^k, \varepsilon_i^{k-1} - \varepsilon_{r^k}^{k-1}\right)$ for

$$\varsigma^k = 0\%.$$

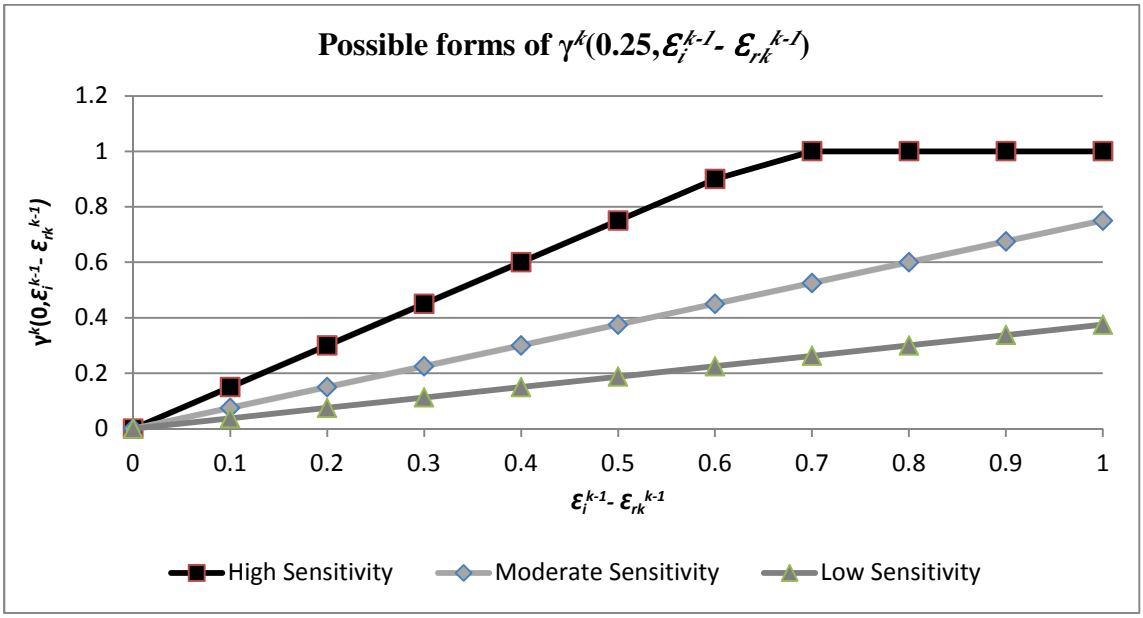

Figure 10: Possible forms of the sensitivity function $\gamma^k\left(\varsigma^k, \mathcal{E}_i^{k-1} - \mathcal{E}_{r^k}^{k-1}\right)$ for

$$\varsigma^k = 25\%$$

3.3 The Multi-Activity DP:

At every time period i = 1,..., n, and depending on the amount of rework previously performed as measured by the last times rework is done defined by the "state" vector $\boldsymbol{r^k} = \{r^k, r^k, \ldots\ldots\ldots\ldots r^2\}$, the model examines 2^{m-1} cases to minimize the total rework for all the activities. The cases represent all possibilities of the decision. For example, when m = 3, then we have three activities where the first activity generates the information, a midstream activity, and a downstream activity that receives the information. The last two activities will have to decide whether to consider the information from the upstream activities or not. So the set of cases would be 2^{3-1} = 4, the first case being that no activity would consider information, the second case would be that only the second activity considers the information, and the third case would be that the third activity alone considers the information, and finally the fourth case would be that both the second and third activities consider the information. Our general m-

activity formulation evaluates each decision for the corresponding state, r^k, and stage, i, and chooses the least costly alternative.

Let $J^k = 1$ if activity k considers information and 0 otherwise and let $Z^k = 1$ if activity k does rework and 0 otherwise. Note that J^k are our decision variables where Z^k are independent Bernoulli random variables with parameters $p^k(i, \boldsymbol{r^{k-1}})$. Then, the minimum expected rework between time i and the end of the concurrent development, n, $R_i(\boldsymbol{r^k})$, when the system is in state $\boldsymbol{r^k} = \{r^k, r^k, \ldots\ldots\ldots\ldots r^2\}$, is given by the following DP optimality equation.

$$R_i(\boldsymbol{r^m}) = \min_{\substack{J^k=0,1,\\ k=2,\ldots,m}} \sum_{k=2}^{m} J^k f^k + \mathbf{E}^2\left[\mathbf{E}^3\left[\ldots \mathbf{E}^m\left[\sum_{k=2}^{m} J^k Z^k\{\alpha^k(i, r^k) + \beta^k\} + R_{i+1}(\boldsymbol{r^k} + \mathbf{J} \wedge \mathbf{Z} \wedge (\mathbf{i} - \boldsymbol{r^k}))\right]\right]\ldots\right] \quad (3)$$

$$R_n(\boldsymbol{r^k}) = \sum_{k=2}^{m} (\alpha^k(n, r^k) + \beta^k + f^k) \quad (4)$$

where $\mathbf{E}^k[.]$ is the expectation operator over Z^k, $\mathbf{E}^k[g(x, Z^k)] = (1\text{-}p^k(i, \boldsymbol{r^{k-1}}))\ g(x, 0) + p^k(i, \boldsymbol{r^{k-1}})\ g(x, 1)$, for any function $g(.)$ and scalar x, $\mathbf{J} = (J^2, \ldots, J^m)$, $\mathbf{Z} = (Z^2, \ldots, Z^m)$, and "$\wedge$" is a type of vector product, $\mathbf{v} \wedge \mathbf{w} = (\mathbf{v}^2\mathbf{w}^2, \ldots, \mathbf{v}^m\mathbf{w}^m)$, for any two vectors $\mathbf{v}$ and $\mathbf{w}$, and $\mathbf{i} = (1, \ldots, n)$.

This m-activity formulation reflects that at every time i, either the information is not considered ($J^k = 0$) at no additional cost, or information is considered ($J^k = 1$), then (i) a fixed cost of f^k is incurred, and (ii) a rework cost of $\alpha^k(i, r^k) + \beta^k$ is incurred (with probability $p^k(i, \boldsymbol{r^{k-1}})$ and the system moves to state i or no rework is performed and the system remains in state r^k . The DP formulation in (1) could be better understood when observing the decision tree below. The decision tree below is the expansion of the above formulation for a specific case, where m=3 activities. We also show how the

multi-activity expands into the regular 3-activity model in Appendix D. Reading this appendix will help the reader to better understand the compacted version of the model.

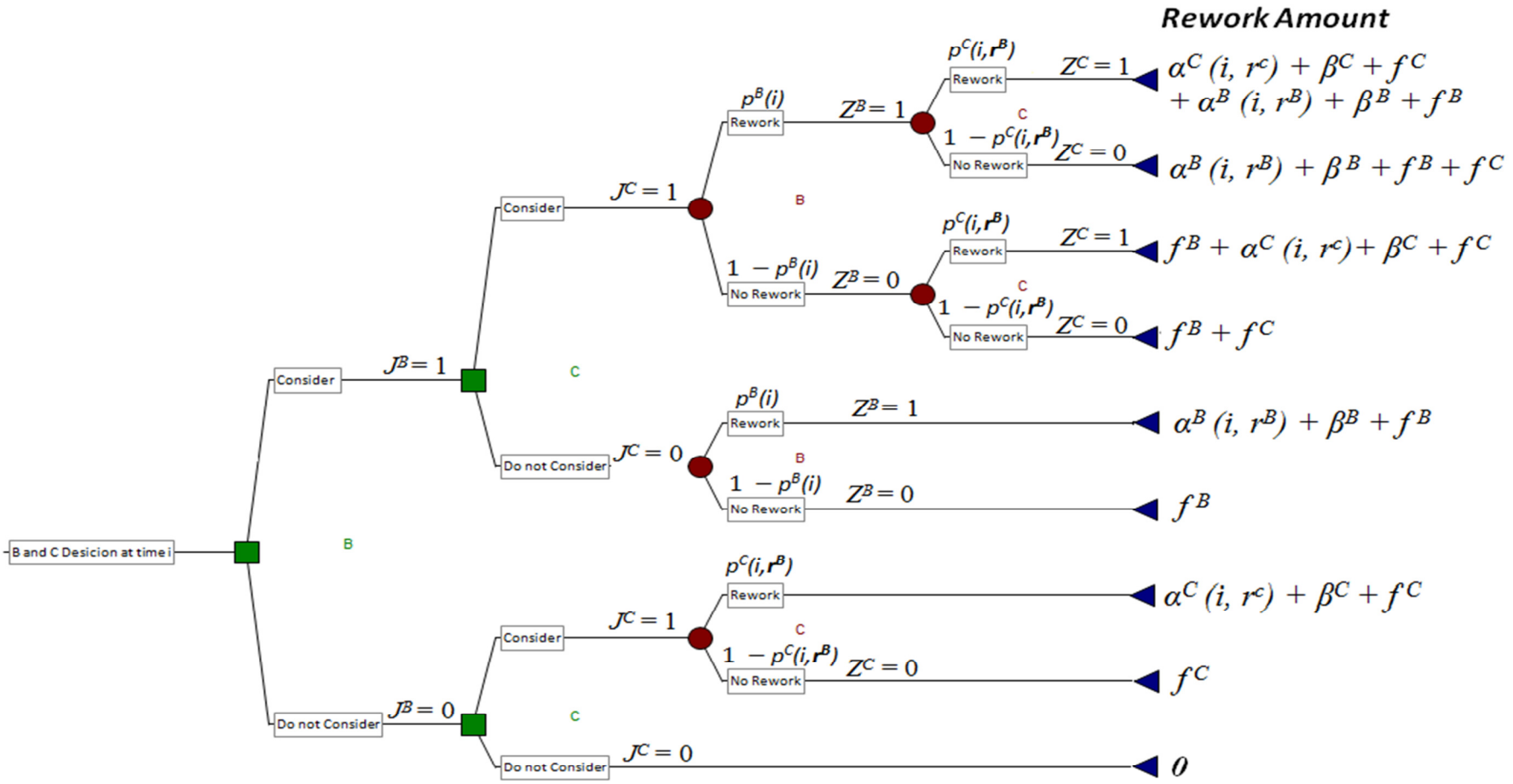

Figure 11: The Decision tree for m=3 activities.

The DP formulation in (3) provides a compact, mathematically elegant, and computationally effective way (i.e., easy to program through iterative nested loops) to manage the flow of information in an IPD environment with any number of activities $m \geq 2$. We provide in Appendix A, a pseudo code that can used to compute the decision matrix. For every stage, and all the states the system could be in. The code quantifies the costs of all possible decisions at that state and stage and chooses the decision with the least cost and records it a decision matrix. The code utilizes the multi-activity model described in equation 3 to do its calculations. This is one of the main contributions of the current study. This contribution is significant given today's computing power where large storage and efficient retrieval is possible. Note that to utilize the DP in (1) in practice, one needs to store the corresponding decisions **J** at every time i, and in every state $\boldsymbol{r}^k$**,** in a large hyper matrix. Then, as the system evolves dynamically over time, starting from state $\boldsymbol{r}^k$ **=** (0**, …,**0) at time i = 1, one retrieves the optimal decision from the matrix. We acknowledge that the calculation of the decision matrix could become tedious and somewhat prohibitive as more activities are added. For instance, when $m = 10$ activities the model would have to compare $2^{10} = 1024$ possible decisions and will have to do this comparison for $\sum_{i=1}^{n-1} i - 1^{m-1} = 387{,}420{,}489$ possible states if $n = 10$ time periods. The calculation of this decision matrix could be done using the pseudo-code found in appendix A. We have been able to calculate such matrices in a respectively short period of time using a regular desktop PC for three and four activities. For larger IPD problems we propose a decomposition heuristic in Chapter 5.

CHAPTER 4

COMPUTATIONAL RESULTS AND INSIGHTS

In order to draw out a series of insights or 'rules of thumb' that would help decision makers without having to calculate the decision matrix every time, we have simulated a set of cases that would cover a large set of the cases in real IPD environments.

4.1 Experimental Setup:

The numerical experiment was built in Anylogic[2] simulation environment for the two, three and four activities models. The experiment is made up of two parts. First a decision matrix is computed for all possible stages and states utilizing the formulation in Chapter 3. Then a Monte Carlo simulation is run, where we generate information qualities according to the probabilities $p^B(i)$, $p^C(i,\boldsymbol{r^B})$, and $p^D(i,\boldsymbol{r^C})$, when needed and using the stored policy, from step 1, to decide on considering the information or not. The analysis focuses on comparing the behavior of both activities B and C as more activities are added. Accordingly, we draw out some important insights that can be used in an IPD environment. We utilize a set of metrics that would help us in quantifying some of the behavioral changes.

The Percentage Variation of Information (PVI) stated below is used to compare the actions of the activities B and C as more activities are added.

[2] AnyLogic is a multi-method simulation modeling tool developed by XJ Technologies.

$$PVI^{k}_{m,m+\delta} = \frac{\sum_{i=1}^{n} \sum_{l=1}^{n_s} \left(J_{ikl}^{m+\delta} - J_{ikl}^{m} \right)}{\sum_{i=1}^{n} \sum_{l=1}^{n_s} J_{ikl}^{m}} \quad (5)$$

where k = B, C denote the activity, m is the number of integrated activities in the model, and n_s is the number of simulations, and J_{ikl}^{m} is the number of times activity k considers information at time i of simulation l in an m-activity model. We use m = 2 3, and 4 for B. That is, we compare the amount of information considered by B in the two, three, and four-activity models. For example, if $PVI^{B}_{2,3} > 0$, then B considers more information, on average, in the three-activity (A-B-C) model then in the two activity (A-B) model. For C, we use m = 3 and 4. In the simulation, J_{ikl}^{m} is determined based on an exhaustive a priori storage of the values of the decision policy.

We utilize the duration $\overline{T}^{k}_{m}$ to compare the timing of considering information of the of the activities B and C as more activities are added.

$$\overline{T}^{k}_{m} = \frac{\sum_{i=1}^{n} \left(\sum_{l=1}^{n_s} J_{ikl}^{m} \right) * i}{\sum_{i=1}^{n} \sum_{l=1}^{n_s} J_{ikl}^{m}} \quad (6)$$

where k = B, C denote the activity, m is the number of integrated activities in the model, and n_s is the number of simulations, and J_{ikl}^{m} is the number of times activity k considers information at time i of simulation l in an m-activity model. For example $\overline{T}^{3}_{B}$ is the average time activity B considers information in the three-activity model.

We define a base IPD process that has 10 periods (n =10) and the following data. For activity B the setup cost for considering the information and the costs of performing rework are $f^B = 0.01$ and $\beta^B = 0.02$, respectively. The values are the same for both activities C and D. All activities have the same "nominal" rework rates $\alpha^k(i, r^k)$ and quality of information probabilities $p^k(i, \boldsymbol{r^{k-1}})$, k = B, C, D. The simulation is run under a series of variations to cover the majority of possible cases a decision maker may be face with. These variations are as follows.

- **Variations in evolution types:**

We vary the forms of these functions to cover all shapes of evolution. We simulate convex, concave, S-shaped and Linear evolution. We also vary the degree of concavity and convexity of all the possible forms to cover most scenarios that a decision maker would come across.

We utilize the functions below to calculate the evolution of the activities as follows:

1. Concave-fast evolution: $\varepsilon_i = 1 + \frac{e^{-\frac{i}{\lambda}} - e^{-\frac{n}{\lambda}}}{e^{-\frac{n}{\lambda}} - 1}$ for λ=0.3,0.5,1,2,4,7 and 10.

   and thus having the following shapes:

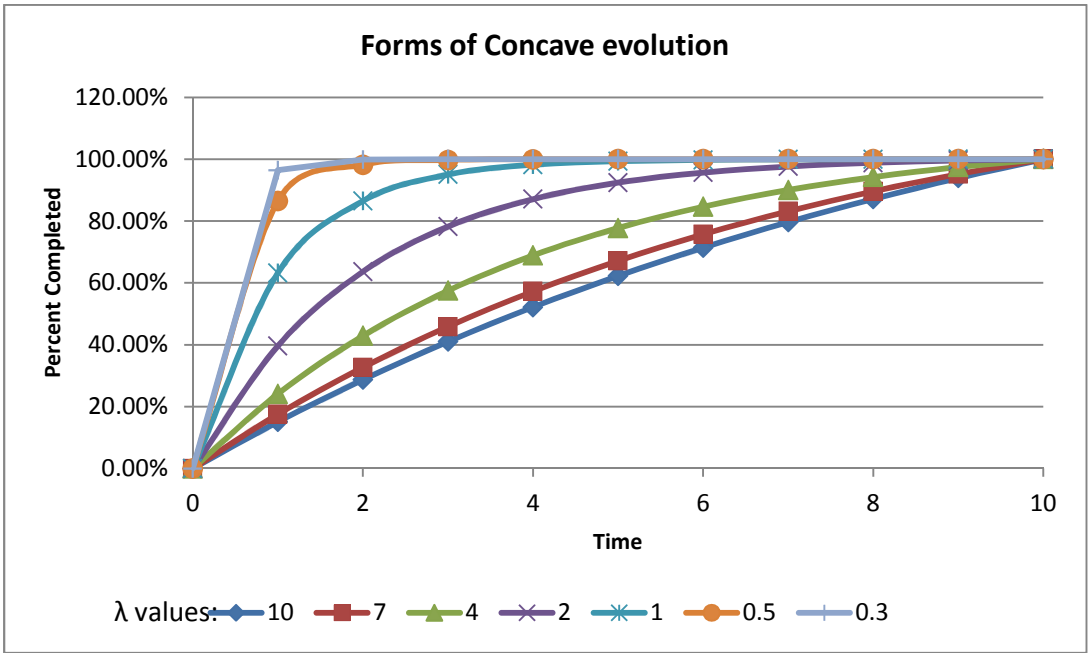

Figure 12: Simulated forms of concave evolution.

2. Convex-slow evolution: $\varepsilon_i = 1 + \frac{e^{-\frac{i}{\lambda}} - e^{-\frac{n}{\lambda}}}{e^{-\frac{n}{\lambda}} - 1}$ for λ=-0.3,-0.5,-1,-2,-4,-7 and -10.

and thus having the following shapes shown in Figure 13.

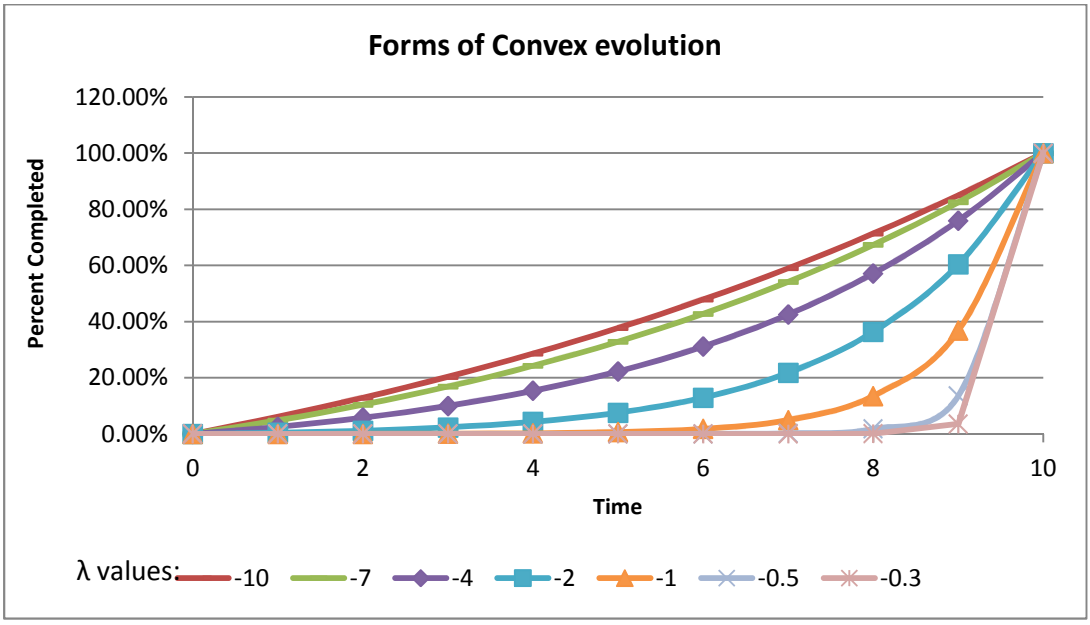

Figure 13: Simulated forms of convex evolution.

3. S-shaped evolution: $\varepsilon_i = \frac{1}{1+e^{-\lambda * i+6}}$ for λ=0.5, 0.8, 1, 1.2, 1.5, 2, 3.

and thus having the following shapes shown in Figure 14.

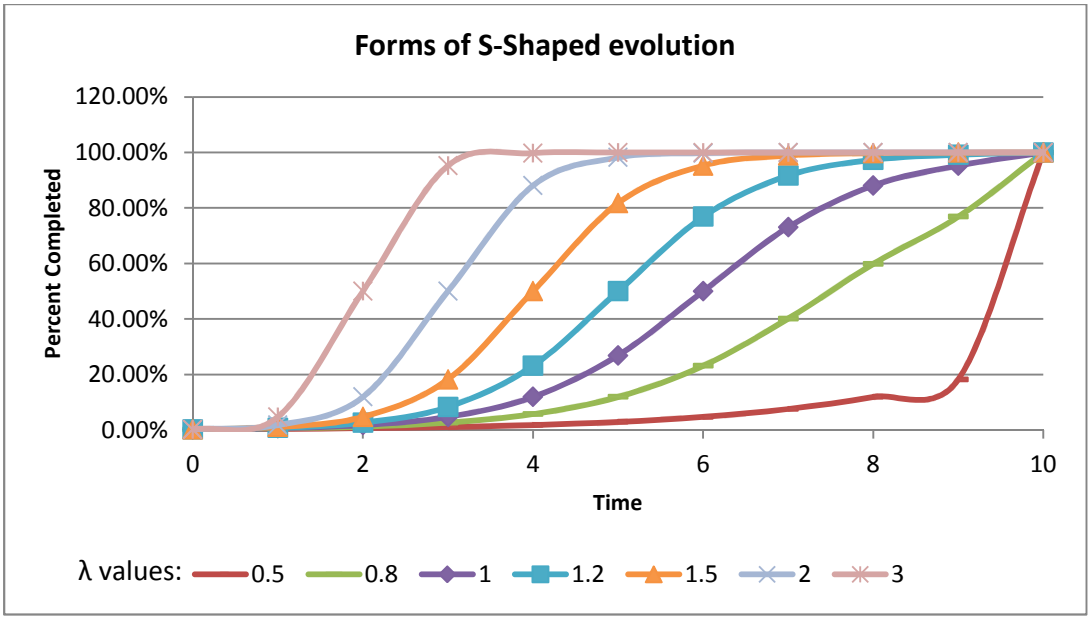

Figure 14: Simulated forms of S-shaped evolution.

4. Linear evolution: $\varepsilon_i = \lambda i$ for λ=0.1. This is the only form of evolution for the linear case since the activity has to be 100% complete at the i=n. The suggested form is shown in Figure 15.

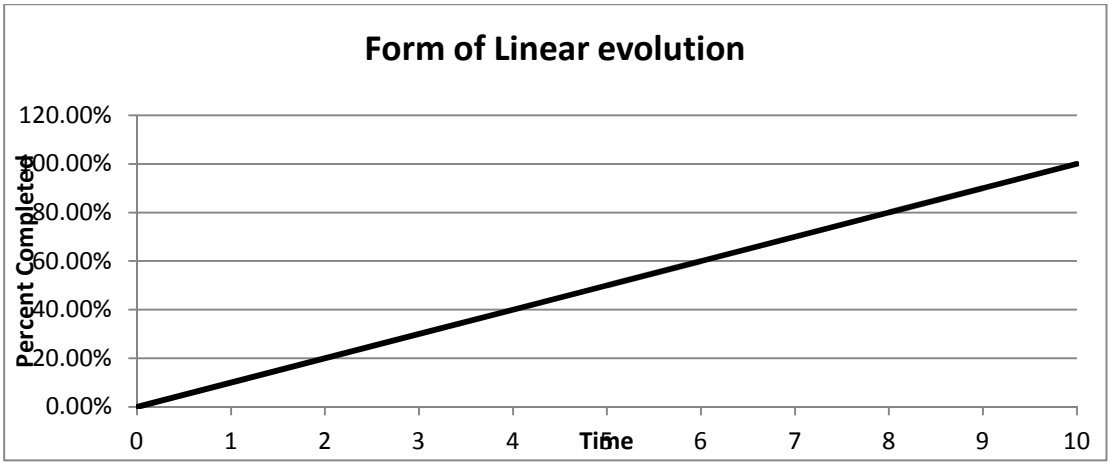

Figure 15: Simulated form of Linear evolution.

- **Variation in the sensitivity of downstream activities:**

As for the sensitivity function, our study is limited to linear sensitivity. The function below describes our sensitivity function:

$$\gamma^k\left(\varsigma^k, \varepsilon_i^{k-1} - \varepsilon_{r^k}^{k-1}\right) = \begin{cases} (1-\varsigma^k)\nu\left(\varepsilon_i^{k-1} - \varepsilon_{r^k}^{k-1}\right) & if\ x < \frac{1}{\nu} \\ 1-\varsigma^k & if\ x \geq \frac{1}{\nu} \end{cases}$$

ν=0.5 (Low),1 (Moderate),2 (High).

The three cases for linear sensitivity are shown in the Figure 16.

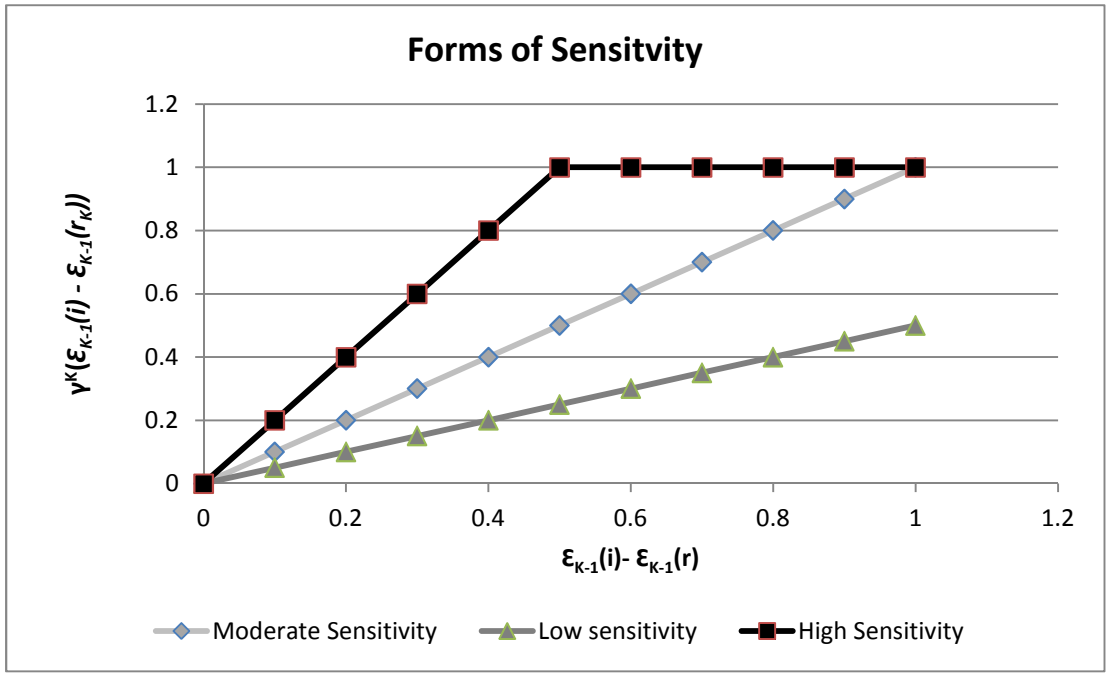

Figure 16: Simulated forms of downstream sensitivity to upstream information changes.

- **Variations in the amount of endogenous information**

We vary the amount of endogenous information (ς) over four values: 0%, 10%,40% and 70%. We assume that all activities would have the same amount of endogenous information.

- **Variations in the fixed costs of considering information and performing rework**

As for the variations of the fixed costs of considering information and performing rework. These costs could have low, medium or high values, and thus for the set of 3 activities would have 3^6=729 cases to compare. However, most of these cases

would not make sense, or whose logic would resemble the logic of other cases. To narrow down our scenarios we shall assume that as you go further downstream these values would have to increase which are summarized in Table 1. We considered various rates at which these costs can increase. For the Base Case there is no increase in these fixed costs. The values of these costs are also normalized, for instance a fixed cost of 0.01 would reflect a cost equivalent to 1% of the nominal work required to finish the activity[3]. Cases 1,2 and 3 represent cases were the fixed costs increase at different rates as one goes downstream, i.e. increasing in *k*. Case 4 examines the case where all the activities have no fixed costs. Case 5 represents the cases where we have no cost of performing rework. On the contrary Case 6 represents cases where activities have no costs of considering information. Case 7 is an exotic case and is used to represent cases where midstream activities have extremely high fixed costs. This case is used mimic cases where the midstream's activity's team could be working in a different country, such as outsourcing to China. This is much like the design of a project is done by specialist companies in different countries so any changes in the requirements of the design would have to be sent to the team, regularly some people would have to travel to explain the new details to the team. Thus ensuing huge costs.

Case	Change To Base Case	Description
Base Case	None	Fixed costs do not change as we go downstream.
Case 1	f_c=0.02, β_c= 0.03, f_D = 0.03, β_D = 0.04	f and β increase linearly at a rate of 0.01
Case 2	f_c=0.03, β_c= 0.05, f_D = 0.05, β_D =	f and β increase linearly at a rate of 0.02

[3] This assumption is not restrictive, yet we made this assumption to give a sense of meaning to fixed costs.

	0.08	
Case 3	f_c=0.05, β_c=0.06, f_D=*0.09*,β_D= 0.1	f and β increase linearly at a rate of 0.04
Case 4	f_B = 0, β_B = 0, f_c= 0, β_c= 0, f_D= 0, β_D = 0	All activities have no fixed costs.
Case 5	β_B = 0, β_c= 0, β_D = 0	All activities have no fixed costs for rework.
Case 6	f_B = 0, f_c= 0, f_D= 0	All activities have no fixed costs for considering information.
Case 7	f_c=0.15, β_c=0.15, f_D=*0.01*, β_D= 0.02	C has very high fixed costs

Table 1: Simulated cases for the variations of the fixed costs.

A Monte-Carlo simulation is run for n_s = 200,000 replications for each case of fixed costs, evolution type, λ, ς and ν. Thus amounting to 2,112 simulation scenarios. All the activities would have the same evolution type, λ, ς and ν. We record the PVI, $\overline{T}$ and amount of rework done by every activity. The large number of simulation runs per case yielded tight confidence intervals on all of the recorded metrics.

4.2 Simulation Experiments

We devised three simulation experiments. The first experiment studies the changes in the behavior of the upstream activities as more activities are added downstream. Our study is limited to the analysis of the two- activity (A-B), three- activity (A-B-C) and four- activity (A-B-C-D) models. The purpose of this experiment is to see if the addition of the downstream activities would affect the decisions of upper stream activities. We vary the evolution type and λ for all the activities and monitor how the upstream activities (B) changes its behavior as the changes occur. Specifically we monitor the

change in the number of times information is considered and the timing of considering the information. We also observe the effect of these changes on the total amount of rework done.

After we learn how the changes in downstream affects the upstream activity's behavior we set up the second experiment to better understand the effects of changes in the nature of upstream activities on downstream behavior. The experiment only takes into consideration the four activity (A-B-C-D) model. We vary the evolution type and λ of the first activity (A) and monitor how the downstream activities (B-C-D) change their behavior as the changes occur. Specifically we monitor the change in the number of times information is considered and the timing of considering the information. We also observe the effect of these changes on the total amount of rework done. The evolution type of activities (B-C-D) is not changed all throughout the experiments and is considered to be convex with λ=10. The same variations discussed in the simulation setup are repeated in cases, evolution type, λ, ς and ν, only varying the evolution type of the first activity (A). We record the amount of information considered by every activity, PVI metric for all downstream activities and the average time of considering information $\overline{T}$. We also monitor how the changes in the upstream activity A's evolution affect the total rework of the system and the rework done by each activity. We draw out a series of insights listed in the next section.

4.3 Results and Insights:

In this section, we first present a set of observations that to validate our model.

Observation 1: When endogenous information is high, midstream and downstream activities would consider less information. Integration would be very lucrative.

When endogenous information is high, then the amount of exogenous information used by downstream activities would be low. Thus the variability in the flow of information would decrease. Downstream activities would no longer have to consider more information when compared to the case where endogenous information is low. The improvement in the quality of information resulting from the evolution of upstream activities would be minimal, therefore activities will not consider information and incur fixed costs. As such we would observe a decrease in the amount of information considered. Figures 17 and 18 show how the increase in the endogenous information decreases the amount of information considered and the total rework.

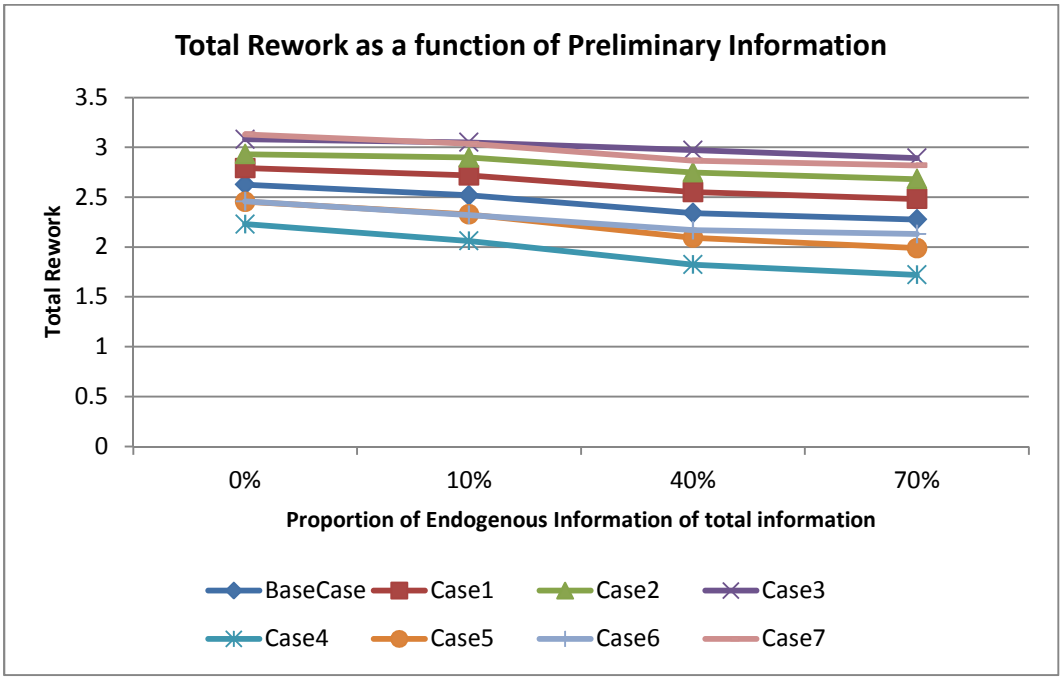

Figure 17: Effect of endogenous information on the amount of Total rework

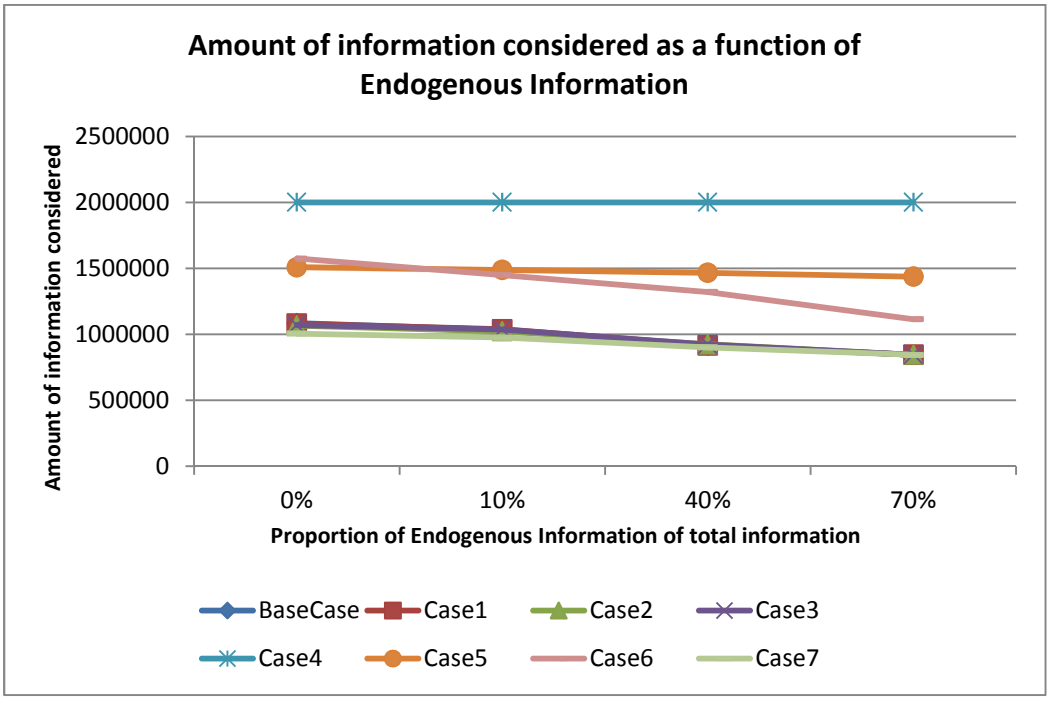

Figure 18: Effect of endogenous information on the amount of information considered.

Observation 2: In the absence of fixed costs, activities would always consider information.

This insight has been proven by Yassine et al. (2011) for the case of two activities with linear evolution, we confirm this phenomenon in the linear evolution case described in that study. We also generalize this finding over all possible forms of evolution. For a more detailed explanation as to why this phenomenon occurs, please refer to Yassine et al. (2011). The simulation results, for Case 4 across all possible iterations of the model parameters, show that the activities would consider information at any stage and state.

Observation 3: The faster the information evolves upstream the fewer the rework is done by downstream activities.

When upstream information evolves quickly, downstream activities will use valid information early on. As such the amount of accumulated rework at later stages would become minimal, thus reducing the total rework costs. Figures 19 and 20 show this phenomenon. For the Base Case with ς = 0% and ν=0.5. Figures 19 and 20, indicate that when upstream activities evolve slower downstream activities do more rework. This is evident as one goes over the Figures from left to right, going through the evolution types from the fastest :Concave-Quick with λ= 0.3 to the slowest: Convex-Slow with λ= -0.3 and form the fastest S-Shaped evolution at λ=3 to the slowest one at λ=0.5.

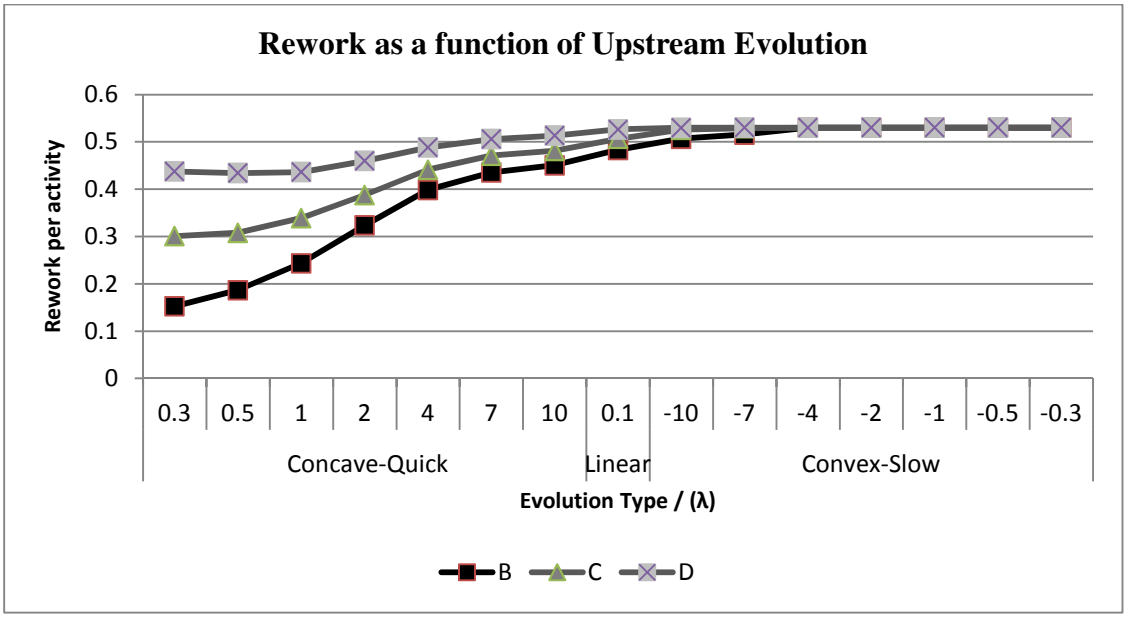

Figure 19: Amount of rework done by the downstream activities as a function of the upstream evolution, for quick, linear and convex evolutions.

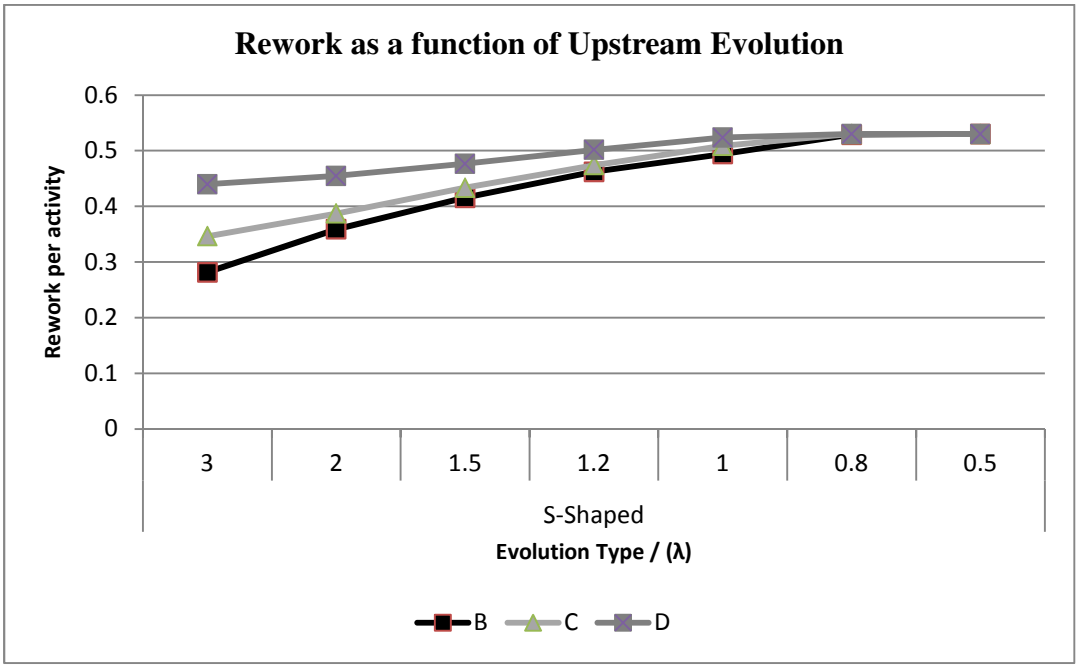

Figure 20: Amount of rework done by the downstream activities as a function of the upstream evolution, for S-shaped evolution forms.

Next, we present a set of useful managerial insights. These insights could be used as general guidelines in any IPD environment.

Insight 1: When upper stream activities evolve too fast or too slow, lower stream activities tend to consider less information than activities that evolve at medium rate.

As the information in the upstream activity tend to evolve fast during the early stages, lower stream activities tend to consider less information. This is due to the fact that as lower stream activities are working with incomplete data, the probability reflecting the chances that this data will induce rework, and thus taking these activities into a better state, is higher when the upper stream activities evolve faster. On the contrary when lower stream activities evolve slower this probability is lower and the chances for the activity considering the information to go to a better state, would be lower. That is why activities getting information from slow-evolving activities would

have to consider more information as the chances for them to actually benefit from that information is low, so they are most likely going to go to the next stage with accumulated rework and will have to consider information again. Another reason for this phenomenon would be that as upstream activities evolve fast during the early stages, the amount of rework accumulated in the last stages would decrease and as such these activities would prefer to consider the information at the terminal time n. At the other extreme and when upstream activities evolve the most during the last stages, activities would only benefit from considering information at the last stages because the information used at the early and middle stages would not be of good quality to consider; i.e. such activities would only have a few stages where the quality of information is good enough to use thus using less information all in all. Figure 21 shows the number of times activity B considers information in the base case where $\varsigma = 0\%$ and $\nu = 0.5$ for all activities. We ordered the evolution types and the λ's from fastest to slowest order of evolution. It is evident that as λ increases in the quick evolution, B tends to consider more information reaching a maximum at λ = 10. When moving to the linear part one could see that the amount of information considered is almost the same as Quick evolution with λ=10 and Slow evolution with λ= -10. At the Slow evolution section one could see that the amount of information considered decreases as the activities evolve later on. As for the S-shaped evolution type one could also see the phenomenon where as the amount of information considered is the most at middle stages.

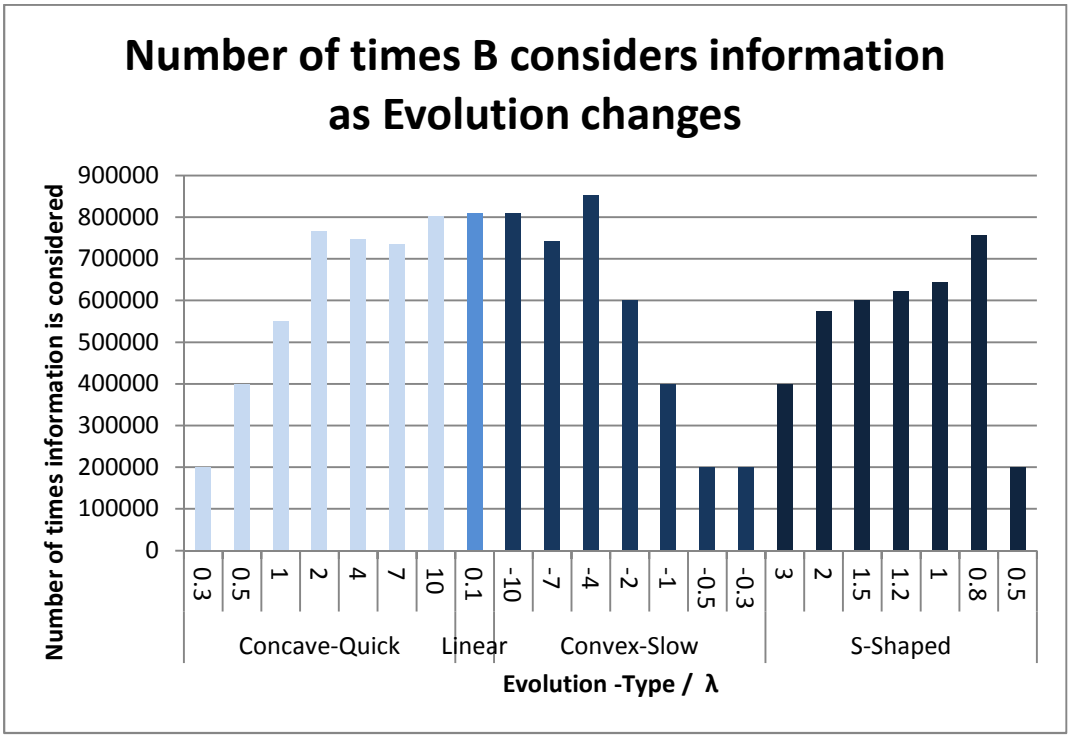

Figure 21: Number of times B considers information as a function of evolution type and λ.

Insight 2: As more activities are added, up-stream activities tend to consider more information.

When the size of the IPD system increases as more activities are added, the upper stream activities tend to consider more information. This is attributed to the fact that the upper stream activities would work on improving the quality of information delivered to downstream activities. The increase in the frequency of considering information enhances the information supplied by these upstream activities, as such the lower stream activities would have their concerns dealt with as early as possible and would thus work with better information leading to fewer accumulation of rework and less amount of times needed to consider information. Again the chances for the upstream activities to help downstream activities are in times where the evolution is moderate, neither very fast nor very slow. As the evolution of the upper stream activities is spread over many time intervals, downstream activities are bound to consider more

information and as such upper stream activities would work on enhancing the information for downstream activities to reduce the risks of sending bad information that would not lead to considering information without any rework. When the upstream activities evolve fast in early stages, extreme concave, the midstream activities would consider the information after the sharp increase in the quality of the information and as such would send information downstream with very good quality without having to consider more in later stages. In Figure 22, we could see how the midstream activity (B) behaves in the 3- and 4-activity scenarios. It is evident that when B considers more information to help C in the 3-activity model. It even uses more information to help C and D in the four activity model. $PVI_{2,3}^{B}$ is used to show the difference in the amount of information considered by activity B when in the two-activity model and three-activity integration model. The same is applied for $PVI_{2,4}^{B}$to show the difference in the behavior of B in the two- and the four-activity model. Figure 22 shows these results for the base case only. However, similar results are observed in cases 1,2, 4, 5 and 6. Cases 3 and 7 do not show this phenomenon due to high fixed costs as discussed in observation 2.

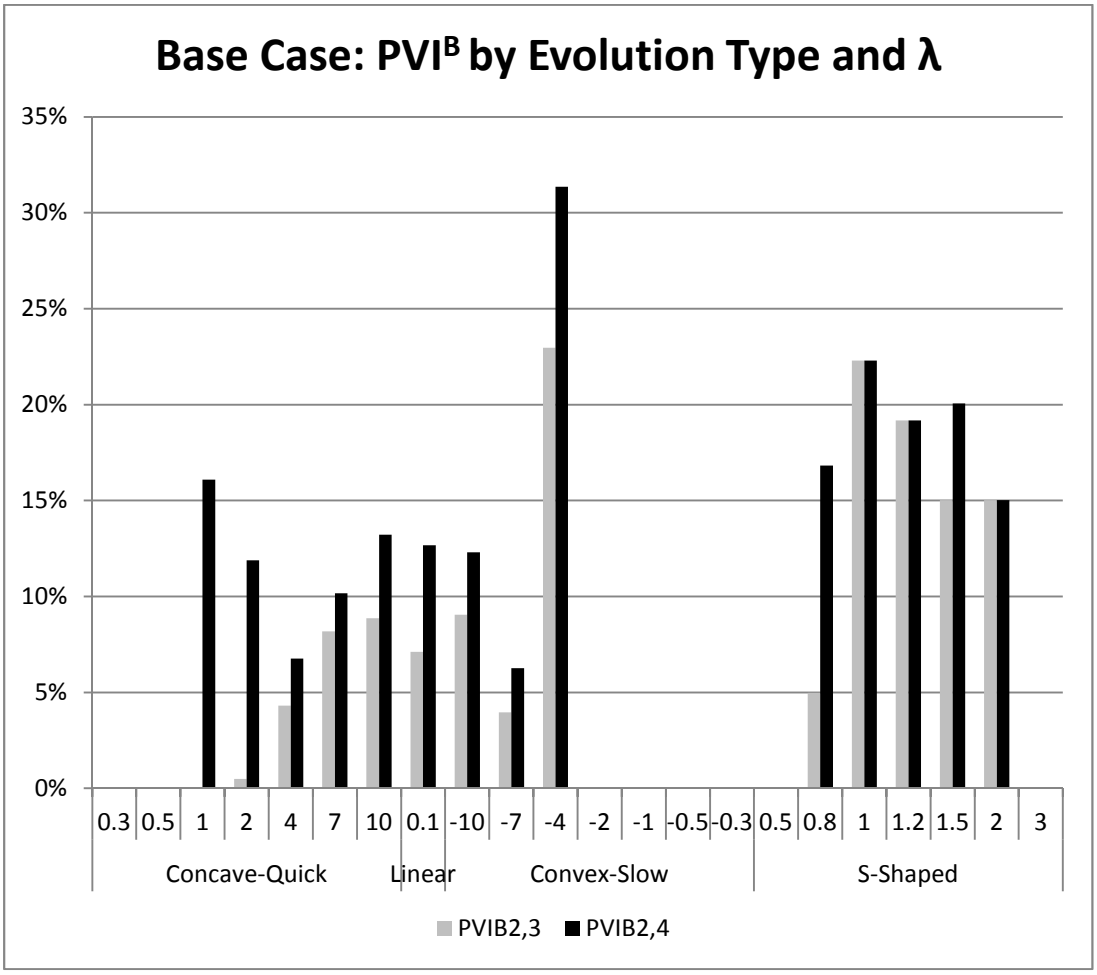

Figure 22: $PVI_{2,3}^{B}$ and $PVI_{2,4}^{B}$ as a function of evolution type and λ.

Insight 3: Midstream activities with high fixed costs block the integration.

When observing the behavior of activity B in Case 7, across all possible combinations for the model parameters, B behaves in the two-activity model the same as it behaves in the three- and four-activity model. i.e. $PVI_{2,3}^{B}$ and $PVI_{2,4}^{B}$ are zero for all the combinations. It is normal for B not to consider information to help C because the latter activity has high fixed costs. However, $PVI_{2,4}^{B}$ was also 0 for all the combinations; meaning that B did not even help D. Therefore the high fixed costs associated with midstream activity C blocked any possible help from B to D and thereby blocking all possible integration efforts. This insight is important in showing the effects

of outsourcing midstream activities in integrated product development. Outsourcing the work of midstream activities to other companies may entail large fixed costs for considering information and performing rework, especially when these companies are outside the country. That is why it is crucial to invest in modern communication software that would ease the communication between the participants, thereby decreasing these fixed costs and unblocking the integration process.

Insight 4: When more activities are added upstream activities consider more information earlier.

This insight confirms the findings of Yassine et al. (2011) for all forms of evolution. Since downstream activities (C and D) must consider the information at the terminal time period *n*, then the "help" of upstream activities (B) by improving the quality of information will be mostly needed in earlier time periods. Accordingly, we expect that (B) considers the information at earlier time periods in the integrated model. When observing the changes in the duration of upstream activity B and midstream activity C, we noticed that this metric would decrease as more activities are added, in cases where B or C consider more information. In Figure 23 below, we show $\overline{T}_3^B - \overline{T}_2^B$, where $\overline{T}_2^B$ and $\overline{T}_3^B$ are the duration of activity B in the two-activity and the three-activity models. If $\overline{T}_3^B - \overline{T}_2^B > 0$ then B would consider information earlier. We also show $\overline{T}_4^B - \overline{T}_2^B$ to reflect the difference in the timing of information considered by activity B in the four activity model and the two activity model. Figure 23 indicates that B would consider information earlier thus justifing our insight. The same pattern is observed for all the other cases but we choose to display the base case alone for presentation purposes.

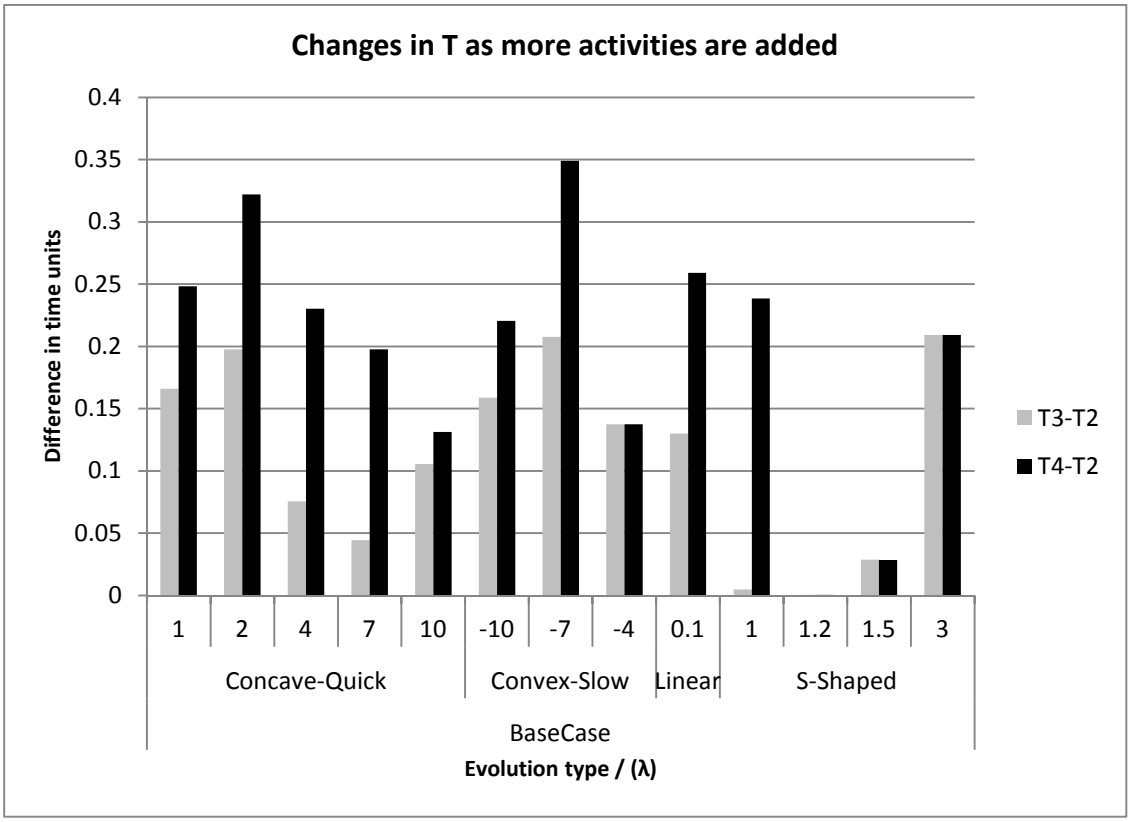

Figure 23: Changes in $\overline{T}$ when more activities are added.

Insight 5: Timing of considering information: The duration of downstream activities increases, i.e. downstream activities consider information later, as upstream activities evolve slower.

The average time of considering information $\overline{T}$ is influenced by the evolution type of the activity originating the information. As the evolution changes from quickest at evolution type Concave at λ=0.3 to the slowest at evolution type Convex at λ= -0.3 and from S-shaped starting quickest with λ=3 and slowest at λ=0.5, we observe that $\overline{T}_4^B$ increases along this variation, observation shown in Figures 24 and 25. This is in accordance with Insight 1, showing that downstream activities working with bad information are more likely to consider information again to reach a better status. This

will increase the average time of considering information. Moreover, activities working with bad information are more likely to choose not consider it. So activities would prefer to wait until the quality of information supplied is good enough to justify paying the fixed cost of considering the information.

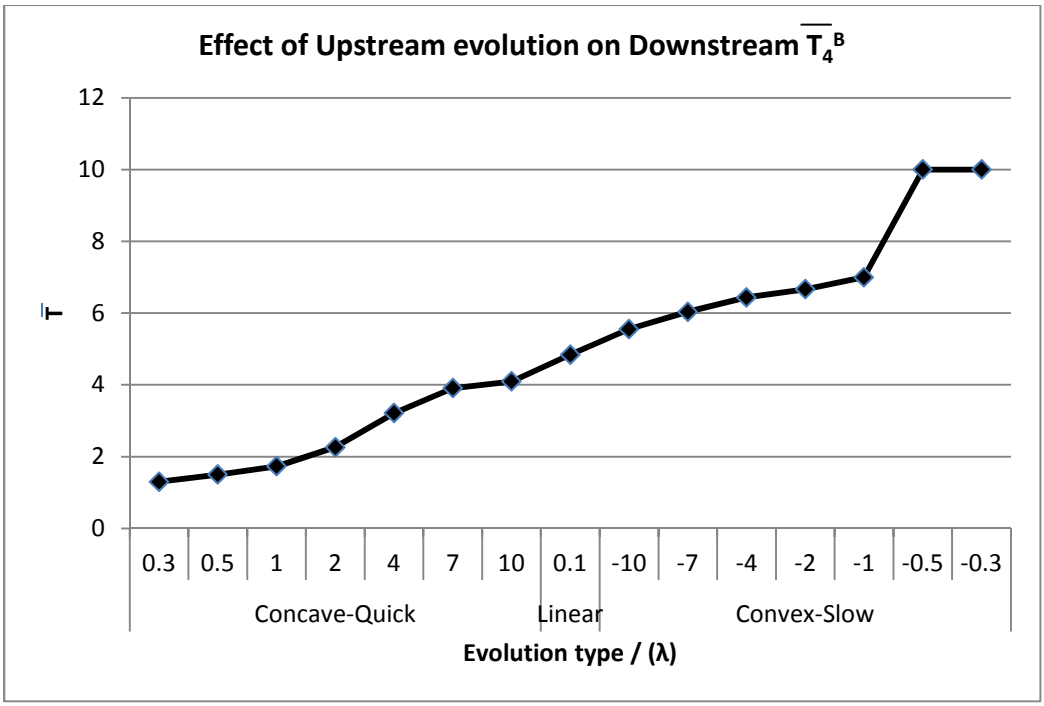

Figure 24: Effect of evolution of upstream activities on $\overline{T}$ as a function of evolution type and λ for Concave-Quick, Linear and Convex-slow evolutions.

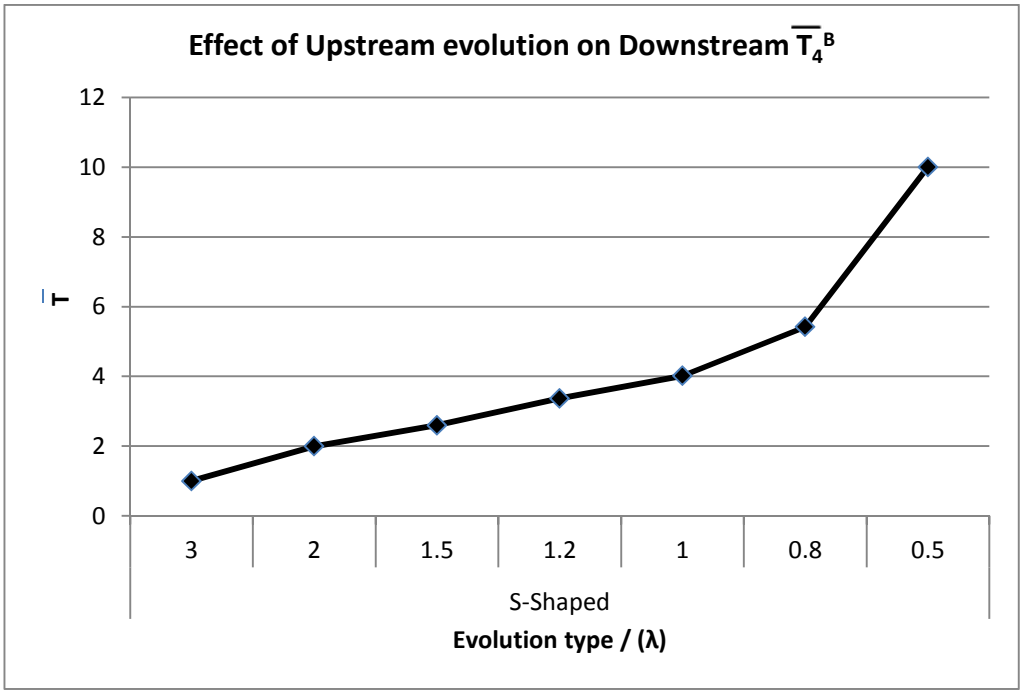

Figure 25: Effect of evolution of upstream activities on $\overline{T}$ as a function of evolution type and λ for S-shaped evolution.

Insight 6: The IPD Reverse Bullwhip effect: The effect of changes in upstream evolution on downstream rework is dampened as information is sent further downstream.

When observing Figures 19 and 20, it is noticed that activity B is most affected by the changes in A's evolution type going from a minimum of 0.15 to a maximum of 0.52. Activity C comes second where its minimum rework costs were 0.3 and maximum rework costs were 0.52. Activity D follows with a minimum of 0.42 and a maximum of 0.52. This observation suggests that upstream activity A's power to affect downstream rework is dampened as we move further downstream. Middle stream activities process the information they receive from upstream activities and will send information to downstream activities with a quality related to their evolution speed, so upstream's power to affect downstream activities is dampened by slow midstream activities. Hence

the observed reverse bullwhip effect is attributed to two factors: The first is the downstream activity's fixed costs of considering the information, forcing the downstream activities to disregard upstream changes at some stages in the IPD process. The second factor causing this reverse bullwhip effect is fact that the information supplied by the midstream activities to the downstream activities is a function of the information supplied by the upstream activities, relying on midstream evolution. Thus, the changes in upstream activity A's evolution type and hence the early improvement of the information is dampened by the ability of midstream activities to process this information and transmit it in usable form for the downstream activities. This delay would decrease the potential benefit that the downstream activities could attain thus resulting in more rework costs for the downstream activities.

CHAPTER 5

THE TWO-ACTIVITY CHAIN HEURISTIC

In this chapter, we propose a heuristic that could be used in any decision environment instead of the current n-activity formulation. The insights presented in Chapter 4 could be used as general guidelines. However, they do not present a precise decision policy. We demonstrate this heuristic for decision makers that are keen on making precise decisions, yet they do not want to suffer the hassle of working with the n-activity model, especially in terms of computer storage. This heuristic is an approximation of the multi activity model explained in Chapter 3. First, we present the heuristic. Then, we compare the performance of the heuristic with the multi-activity model, and a naive always consider information policy. We conclude with a discussion about the advantages and the disadvantages of this heuristic.

5.1 The Two-Activity Chain Heuristic Model

The proposed heuristic works in the following manner. A participant in the IPD environment would consider all the activities that precede his activity as one origin of information, whose quality of information is known ahead of time and considered static for any time *i*. The participant will also ignore all downstream activities and will act selfishly, disregarding any information that would enhance the quality of information to the downstream activities only without giving direct benefit to the concerned participant. Thus the participant does not have to know what is current status of all the upstream and downstream activities, his decision will totally be based on his own status. This act would help decrease the complexity of the decision for the participants at every

time *i*. Thus, activity *k* would rely on the decision policy of activity $k-1$, in turn $k-1$ would rely on the policy of $k-2$, thus the naming of the heuristic as the two-activity chain heuristic. The heuristic works in three steps:

Step 1: *Estimate the quality of information sent from all the upstream activities to the concerned participant (activity k) as a function of time i.*

An estimate of $p^k\left(i, \boldsymbol{r^{k-1}}\right)$ need be used. All the parameters used in the n-activity model, presented in Chapter 3, are used in the calculation of this heuristic.

We will start our explanation of the scheme to calculate $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}$ by assuming that $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-2}}$ is known, as such the function $p^{k-1}\left(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-2}}\right)$ could be calculated as described in Chapter 3 by substituting the values of $\boldsymbol{r}_{\boldsymbol{i}}^{\boldsymbol{k-2}}$ by the average vector $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-2}}$. The resulting function would only depend on the time *i* and the estimated vector $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-2}}$; it is independent of the dynamic state of the upstream activities. We also assume that a decision policy for the activity *k-1* is available and that it was computed using this heuristic thus depending only on r^{k-1}.

Calculating $\bar{r}_i^{k-1}$ and amending it to $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-2}}$ would be sufficient to get the estimation vector $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}$. At any time *i,* the activity *k-1* could be at any state $r^{k-1} \in [0, i-1]$ with a certain probability depending on the decision policy of the activity *k-1* and the quality of information this activity uses $p^{k-1}\left(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-2}}\right)$. At every time stage an activity could choose to consider information or not consider the information. This is decided by its decision policy. If it does not consider information it will move to the next stage with the same state. On the other hand if it chooses to consider the information it could move to the next stage with a state equal to the current stage if the information is valid. However, if the information is not valid it will remain in the same state. Figure 26

illustrates these ideas and shows how the state of the activity could change depending on the decision policy and the quality of information it is using.

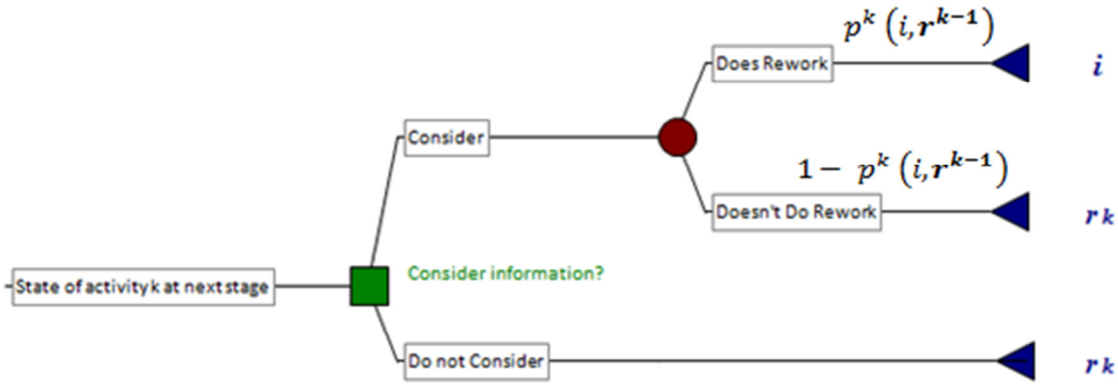

Figure 26: Decision tree for the activity *k* at time *i* showing the possible states that the activity could attain in the next stage.

Therefore at every time *i,* the activity could remain in its state or attain the state of the current time *i.* Once the activity considers information it may move to the next stage with $r^k = i$, no matter what its current state is, as long as it considers the information and receives valid information inducing rework. If it does not do any rework, it can only move to the next stage with its current state. Figure 27 shows these possible transitions for an activity from time 1 until time 4.

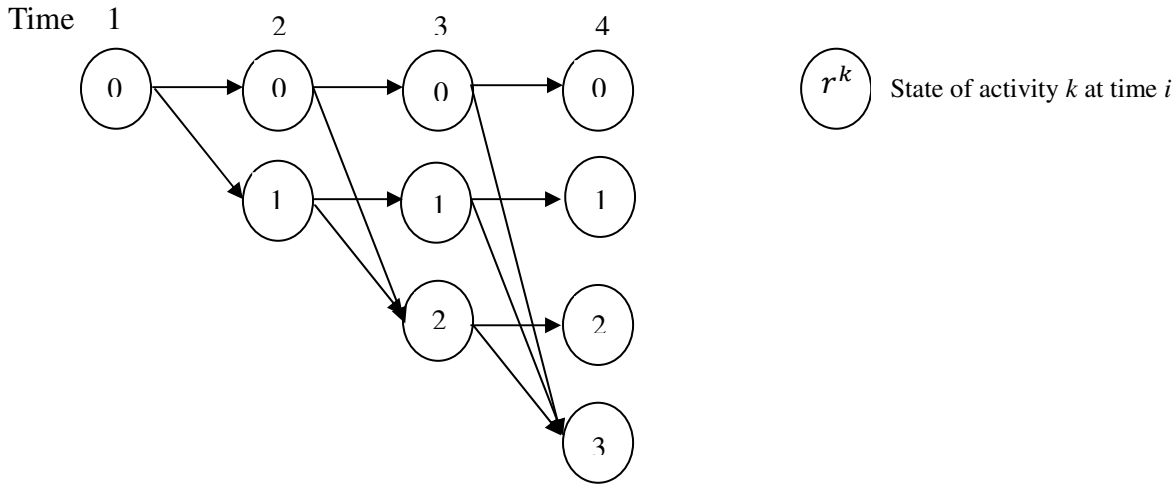

Figure 27: Possible paths of activity *k* from stage 1 to stage 4 depending on the decision policy and the quality of information used.

As illustrated in Figure 27, the probability that an activity would be in a certain state at a certain stage depends on its path, thus depending on the probability of being in its previous state at the previous stage $(i-1, r^k)$, its decision policy and the quality of information used. As such we compute the probability matrix $q^k(i, r^k)$ (6) that describes the probability that activity *k* would be in stage *i* with state r^k. After finding this probability matrix, we could calculate $\overline{r}_i^k$ as the weighted average of the possible states the activity *k* could attain at the time *i*. The function $q^k(i, r^k)$ is defined as follows:

$$q^k(i, r^k) = \begin{cases} 1 & i = r^k = 0 \\ q^k(i-1, r^k)\left(1 - D^k(i, r^k)p^k\left(i, \boldsymbol{r_i^{k-1}}\right)\right) & 0 \le r^k \le i-1, 1 < i \le n \\ \sum_{s=0}^{r^k-1} y^k(i-1, s)D^k(i, s)p^k(i, s) & r^k = i, 1 < i \le n \end{cases} \quad (6)$$
$$1 \le i \le n$$

Where the function $D^k(i, r^k)$ is the decision matrix defining the actions of activity *k* at time *i* and state r^k; $D^k(i, r^k) = 1$ when *k* considers information and 0 otherwise. Although $q^k(0,0)$ does not exist in practice we define it as a starting point for the

matrix. The third term shows the fact that the activity belonging to any state could attain $r^k = i$ as long as it considers valid information.

After calculating the probability matrix $q^k(i, r^k)$ one could easily estimate the value of $\bar{r}_i^k$, by computing the weighted average of the possible values of r^k using function (7) as described below.

$$\bar{r}_i^k = \sum_{s=0}^{i} q^k(i,s) \times s \quad 0 < i \leq n \tag{7}$$

The calculation of the matrix and the corresponding values of $\bar{r}_i^k$, assumes that when upstream activity (*k)* receives valid information, information sent to downstream activity (*k+1*) would be affected simultaneously. As such one could observe values for $\bar{r}_i^k = i$ meaning that for sure the activity is going to perform rework at this time *i.*

Using the above formulations one could get the $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}$ from the preceding activity's (*k-1*) decision matrix and using $p^{k-1}(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-2}})$. Finally we conclude this first step by calculating $p^k(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}})$ using the function described in Chapter 3 with the state estimation matrix $\bar{\bar{r}}_i^{k-1}$.

An example showing all the calculations done in step 1, could be found in Appendix E.

Step 2: *Use the estimated* $p^k(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}})$ *in Step 1 to calculate the decision matrix of activity k, utilizing the two-activity format of the multi-activity model.*

This step is by far less complex than the first one. We just use the estimate of the quality of information $p^k(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}})$ calculated in Step 1 as the quality of exchanged information. Solving the recursive DP would result in a decision policy for the activity *k* depending only on r^k.

5.2 Assessing Heuristic Performance by Simulation.

In an effort to assess the performance of the heuristic, an experiment of 2,112 simulation scenarios was conducted. The iterations were identical to the ones described in Section 4.1. The simulation would calculate the decision matrix for every activity using the proposed heuristic, as described in Section 5.1. The simulated scenarios were limited to the four-activity (A-B-C-D) model. These decision matrices would be stored in memory and then used in the simulation to decide whether to consider information at every time stage *i*. While calculating the decision matrices, the vector $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}$ would be used to estimate the quality of information. However, while running the simulation we would generate values for $p^k\left(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}\right)$ using the dynamic values of the state vector $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}$ to mimic real life situations.

We compared the total amount of rework for the activities when using the multi - activity policy and the heuristic's policy. The results show that the amount of total rework when using the heuristic were very close to the optimal multi-activity policy, found as discussed in chapter 3. In some cases the heuristic policy would perform the same as the multi-activity policy, this was observed in Case 4 where both policies would advise to consider information at all stages and states. The worst result recorded was for Case 5 in the s-shaped evolution with λ=1.5, □^{k}=0% and ν=1; where the heuristic led to 1.87% more rework. On average the heuristic's policy performed worse than the multi-activity policy by 0.04% more rework. Statistically both policies are the same as their means are not very far from each other. For instance, the average rework for worst result, reported above, was 2.67 with a standard deviation of 0.17 when the optimal policy is used. On the other hand, using the heuristic's policy the average rework was 2.72 with a standard deviation of 0.12, thus showing that these numbers are statistically

the same. Figure 28 shows the percent difference between these two policies. The horizontal axis shows an index for the simulated cases, we did not indicate the parameters for the specific case because it is not needed in this presentation. Figure 28 is only used to show the variations and the set of input that this Figure corresponds to is not relevant to the study. A positive percentage shows that the heuristic led to more rework. We observe that in no case the heuristic performed better than the multi-activity, thus giving a sense of validation to our work.

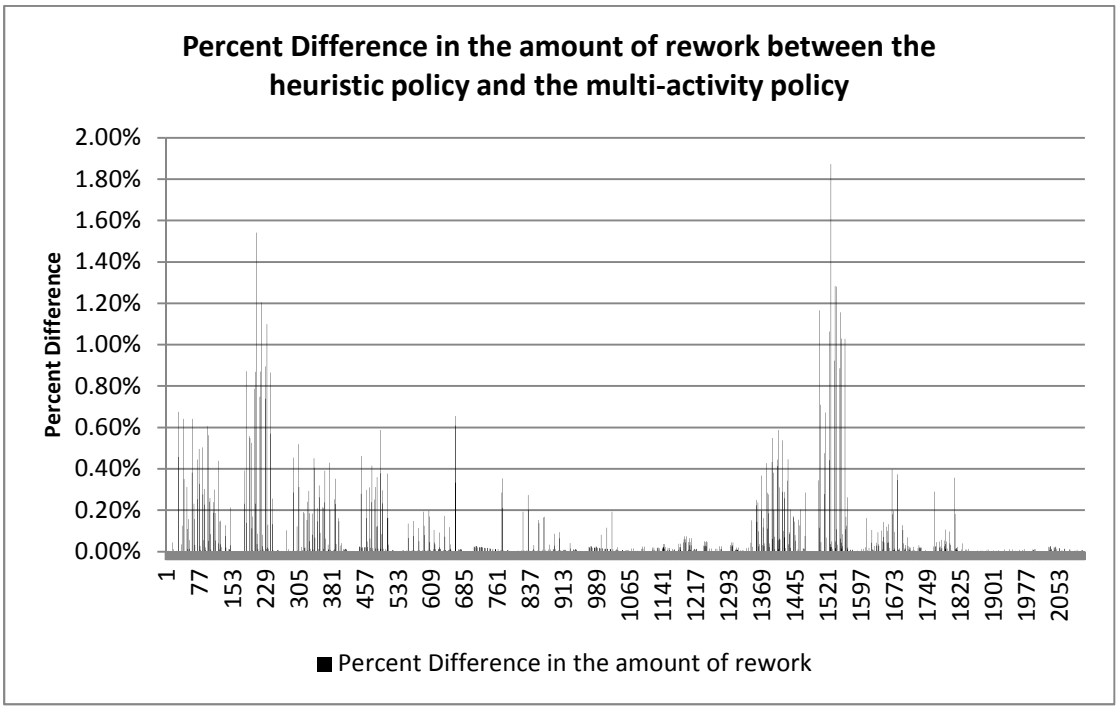

Figure 28: Percent Difference in the amount of rework between the heuristic policy and the multi-activity policy.

We also tried a naive policy, where at every time any participant in the IPD process would consider the information they get from upstream activities. We simulated the four (A-B-C-D) activity setting across all the 2112 scenarios described in section 4.1. The results, shown in Figure 29, show that this naive policy does not perform as well as heuristic. In fact it performs the same as the multi-activity model in cases where this model would advise to always consider information, like case 4. The biggest difference

in the amount of rework is 80% more rework and on average the naive heuristic performs worse by 30% more rework than the optimal multi-activity model. Considering the small size of the problem, $n = 10$, a 30% more rework is significant for such a small project. It is expected that this amount would increase when the project becomes bigger, $n \geq 20$.

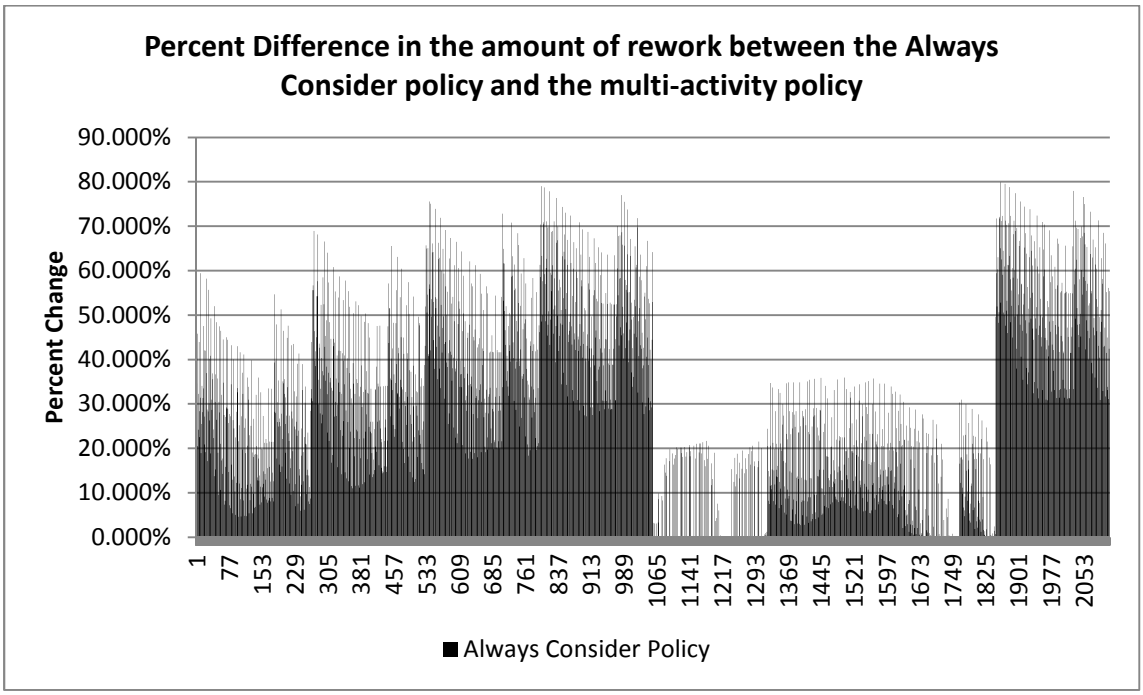

Figure 29: Percent Difference in the amount of rework between the always-consider policy and the multi-activity policy.

The most notable contribution that this heuristic presents is the reduction in the size of the decision matrix thus reducing the complexity of the decision for the participants. The size of the matrix for the multi-activity matrix would be $\sum_{i=1}^{n} n^m$ whereas for the matrix size of the heuristic is only $\sum_{i=1}^{n} mn$. For a 50-stage, 20-activity IPD environment, the multi-activity policy would have 2,763,020,625 decision values for the set of states. On the other hand, for the same example and while using the heuristic's policy only 25,500 decisions states need be stored for the whole system. This 99.999% reduction in the size of the decision matrix is very important in terms of storage and

calculation of the matrix. This will help companies in applying this study the need of PIM/PDM system as it is easy to compute and use. The participants would not have to worry about the statuses of the preceding and succeeding activities, thus making this model more applicable in any IPD environment. Moreover, a setup meeting is only required at the beginning of the product development processes where all the decision policies are computed and no further coordination would be required, to adjust this policy or update the remaining participants to the current status of the remaining activities. Nevertheless, the heuristic policy can also be updated during the project-cycle in case the need arises.

The major drawback of this heuristic is that upstream activities act selfishly and hence do not cater for downstream concerns. Thus downstream activities will have to work with any information available from upstream activities. The optimal information exchange policy reduces the risks of working with this incomplete data by adjusting the behavior of the downstream activity to the expected changes in upstream information, this act lessens the effects of the selfishness of the upstream activities.

A good extension to the study of this heuristic could be to consider a 3 activity chain heuristic, where all upstream activities are collapsed into one source of information and all downstream activities are collapsed into one downstream activity. The concerned activity k would then shape its policy by considering the quality of information sent by upstream activities and by catering to downstream activity's concerns, thus achieving near optimal solution to this problem.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, we have developed a multi-activity policy to manage the exchange of partial information in an integrated development policy. We based the model on upstream information evolution and downstream sensitivity. The proposed model is first of its kind to propose a quantitative, precise, method to calculate a decision policy for a multi-activity IPD environment. This the major contribution that this thesis offers to the literature of this type of study.

After building the model, we resorted to simulation it to study the dynamics of the model and draw out some observations and insights. The insights are considered as general guidelines for any participant in an IPD process. The simulation covered a vast array of possible IPD cases, covering most of the variations of the model's parameters. We utilized two experiments, the first experiment was tailored to study the effects of downstream changes on upstream' behavior. The second experiment was used to study effects of upstream variations on downstream activities' behavior. The simulations showed that upstream activities would consider more information in integration than in isolation, they would do so at early stages. The most important of these insights is the reverse bullwhip effect in the IPD environment: The effect of changes in upstream evolution on downstream rework is dampened as information is sent further downstream.

In chapter 5 we proposed our two-activity chain heuristic. This decomposition heuristic eases the use of this study by sacrificing a small amount of potential rework savings. A simulation study compared the performance of the heuristic to the multi-

activity model, showing very close results. The heuristic would be a less costly solution in terms of policy calculation, decision matrix storage and coordination amongst the participants.

One limitation of the proposed multi-activity model is that it does not take into consideration the effect of the exchange of information on the evolution of the activities. When information is changed upstream the activity receiving the information would have to incorporate this change, taking time to do so and thus affecting the evolution of the activity. We considered that these costs are incurred concurrently, however in practice this the changes would take time to finish and hence would be incurred over several time stages. The integration of such a scheme could be a dense subject for future work.

Another important limitation is the fixed length of the activity duration. The assumption that all activities should complete their work before a defined deadline in time is a binding in real applications. A good extension to the model would be to feature different sizes for the activities' duration.

Finally, another interesting extension to our proposed model is to include feedback between these activities. This requires more fundamental changes to our model and is worthy of further investigation.

REFERENCES

Clark, K., Fujimoto, T., 1991. Product development Performance: Strategy, Organization, and Management in the World Auto Industry. Harvard Business School Press, Boston.

Gerwin, D., Barrowman, N., 2002. An Evaluation of Research on Integrated Product Development. Management Science 48(7), 938-953.

Ha, A. Y., E. L. Porteus. 1995. Optimal timing of reviews in concurrent design for manufacturability. Management Science 41(9) 1431–1447.

Hauptman, O., Hirji, KK., 1999. Managing Integration and Coordination in Cross Functional Teams: An International Study of Concurrent Engineering Product Development. R&D Management, 29(2), 179-192.

Lee, HL, Padmanabhan, V., and Wang, S., 1997. Information distortion in a supply chain: the bullwhip effect. Management Science, 43,(4): 546-558

Joglekar, N., Yassine, A., Eppinger, S., Whitney, D., 2001. Performance of Coupled Product Development Activities with a Deadline. Management Science 47 (12), 1605–1620.

Krishnan V and Ulrich KT. 2001. Product development decisions: A review of the literature. Management Science 47(1),1-21.

Krishnan V, Eppinger SD, Whitney DE. 1997. A model-based framework to overlap product development activities. Management Science 43(4):437-51.

Lin J, Qian Y, Cui W, Miao Z., 2010. Overlapping and communication policies in product development. Eur J Oper Res 201(3),737-50.

Loch CH and Terwiesch C. 1998. Communication and uncertainty in concurrent engineering. Management Science 44(8):1032-48.

Nambisan, S., 2003. Information systems as a reference discipline for new product development. MIS Quarterly. 27(1), 1-18.

Yassine, A., Joglekar, N., Braha, D., Eppinger, S., Whitney, D., 2003. Information Hiding in Product Development: Design Churn Effect. Research Engineering Design 14 (3), 145–161.

Yassine A., Sreenivas RS, Zhu J. 2008. Managing the Exchange of Information in Product Development. European Journal of Operational Research 184 (1), 311 – 326.

Yassine, A., Maddah, B., Nehme, N., 2012. “Optimal Information Exchange Policies in Integrated Product Development,” Submitted.

Appendix A

PARAMETERS FOR THE MULTI-ACTIVITY MODEL

- $\square_i^k$: The evolution function of the activity k.
- f^k: Fixed cost of considering information for activity k.
- β^k: Fixed cost for performing rework for activity k.
- r^k: The last time activity k did rework.
- $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}$: State vector containing all the states of the activities preceding *k+1* except the first activity *(r_k, r_{k-1}, …… r_2)*
- $\square^k$: Proportion endogenous information out of required information available for activity k at time 0.
- $\alpha^k(i, r^k)$: The potential rework at time *i* for activity *k*.
- $p^k\left(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}\right)$: Is the probability that the information considered is valid information and as such would induce rework.

Appendix B

ALGORITHM TO CALCULATE THE MULTI-ACTIVITY DECISION POLICY AS PER THE PROPOSED MODEL

It can be shown that the algorithm below is equivalent to the formulation of the DP in (1).

For $r_k = 0$ to $n-1$,
For $r_{k-1} = 0$ to $n-1$,
.
.
.
For $r_1 = 0$ to $n-1$,
For K=1 to K=m
Set $R_n(r_1, r_2, \ldots, r_{m-1}, r_m) = R_n(r_1, r_2, \ldots, r_{m-1}, r_m) + \alpha^k(i, r_k) + \beta^K + f^K$

For i = n-1 to 1
For $r_m = 0$ to $n-1$,
For $r_{m-1} = 0$ to $n-1$,
.
.
.
For $r_1 = 0$ to $n-1$,
{ Set $R_i(r_1, r_2, \ldots, r_{m-1}, r_m) = R_{i+1}(r_1, r_2, \ldots, r_{m-1}, r_m)$
For $J_m = 0$ to 1,
For $J_{m-1} = 0$ to 1,
.
.
.
For $J_1 = 0$ to 1,
{
set rework=0
For $Z_m = 0$ to J_m,
For $Z_{m-1} = 0$ to J_{m-1},
.
.
For $Z_1 = 0$ to J_1,
{
Set probability = 1
Set work=0;
for k=1 to m
{
probability=probability*(J_k+J_k*(Z_k+$p^k(i, r1, r2, \ldots\ldots, rm-1, rm) - 2\,Z_k\,p^k(i, r1, r2, \ldots\ldots, rm-1, rm)$))
work=work+ J_k*(Z_k *($\alpha^K(i, r_k) + \beta^k$) + f^k)
}
rework=rework + probability *(work +$R_{i+1}(Z_1 * i + (1 - Z_1) * r_1, \ldots\ldots, Z_m * i + (1 - Z_m) * r_k)$)
}
If rework < $R_i(r_1, r_2, \ldots, r_{m-1}, r_m)$
{
Set $R_i(r_1, r_2, \ldots, r_{m-1}, r_m)$ = rework
Set decision$_i(r_1, r_2, \ldots, r_{m-1}, r_m)$={$Z_1$,$Z_2$,.....$Z_{m-1}$,$Z_m$}
}
}
}

Appendix C

MULTI-ACTIVITY SIMULATION MODEL

In this appendix, we describe the simulation tool that was used to simulate the various variations of the model as depicted in chapter 3.

The model was built in Anylogic Professional simulation environment. Anylogic is a multi method simulation software, used to simulate Discrete Event Simulation, System Dynamics and Agent Based Simulation. The software is developed by XJ technologies headed by Dr. Adrei Borchev. Its client base expands across various industries. Companies such as CCC, NASA, Schlumberger , HSBC, Booz & co and many other Fortune 500 companies rely on Anylogic for their heavy simulation analysis.

The simulation tool is divided into two parts. The first part is the simulation software that is developed in Anylogic. The second part is the results analysis software, developed in SQL server and SQL server reporting services.

C.1 Simulation Software

Before any simulation is launched, the software computes the multi activity policy and stores in memory the hyper-matrix representing the decision policy for the activities at any stage and state the system may be in. The software utilizes the algorithm found in Appendix B to calculate this policy. The chosen policy is stored in a global array decision[n][n]....[n][k][i], having k+2 dimensions. The first k dimensions are of size n(number of time stages), used to refer to the possible states, and a dimension for the activity and time. So for a certain state $\ddot{r}^k=(r_k, r_{k-1}, \ldots\ldots r_2, r_1)$ you would have k decisions(for all the activities) and for every time hence the k+2 dimensions. This array will hold the optimal decision for all possible states the system can attain at any stage.

After setting up the simulation by calculating the corresponding decision matrix an entity is injected into the discrete event system. The process logic is shown in Figure 30 below.

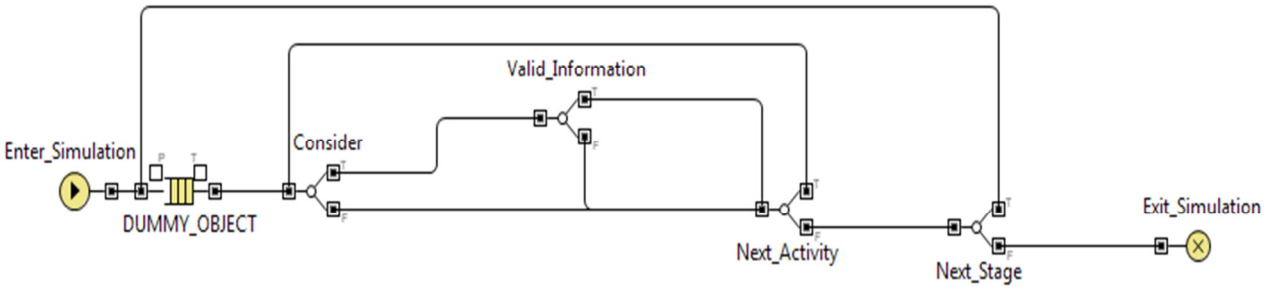

Figure 30: Process logic for the discrete event simulation model.

The process flow of the simulation is made up of an entity source(Enter_Simulation), two nested loops and a sink(Exit_Simulation). Entities enter the system through the Enter_Simulation node. One would think of an entity entering the system as a set of activities belonging to the IPD system starting at time 0. When entering the system the entity will hold a status vector (rk[k]), and an array of number equal to the number of activities. This array called reworkactivity[k] will hold the amount of rework performed by each participant belonging to the IPD system. The index of the participant starting from 0 all the way to m-1 will be the key to all the saved arrays. Along with the rework[k] array the entity holds the Decision[k] array and stage. The Decision[k] array holds the entity's decision for the next stage. The stage variable holds the current stage of the system. Two global arrays, that are not entity specific, are also worth mentioning; consideredact[k][i] and didrework[k][i].These two arrays are used record for every activity k at time i if it considers information or not, and if it does so it records if the activity performs rework.

The entity enters the system with its status vector rk[k] = {0} meaning that all activities haven't done any rework. All other variables are set to zero except stage, it is set to 1 to signify the system has reached the first stage of the decision process. After leaving the Enter_Simulation node, the entity goes to the DUMMY_OBJECT node, this node is used as a dummy node for technical purposes, not related to the actual simulation process. At this node the entity fetches the set of decisions for all its activities from the decision stored in decision[n][n]....[n][k][i]. It then goes to the next node with its variable activities=0, meaning that we are going to simulate the first activity of the IPD system. At the consider node the activity checks it current decision, if it was to consider the info it will leave through the True branch and goes to Valid_Information and increments the global array consideredact[activities][stage] at the correct activity index and stage index; else it will leave through the False branch then goes to Next_Activity. Entities reaching the Valid_Information node are entities representing activities that have considered information. Once the entities enter the node we generate a probability according to $p^k(i, \ddot{r}^k)$ where in this simulation is called proba(stage, activities, rk[k]), the function computes the probability of good information for the current activity. The select node generates this probability and if the information is valid the entity exits the node through the False branch, adding to the amount of reworkactivity[k] and rework the amount of rework done an updates the state of the system by updating rk[k] at k=activities. It also increments the corresponding value for didrework[k][i] at k=activities and i=stage. If the activity does not do rework it will leave through the True branch. Reaching the Next_Activity node, the activities variable is incremented. If activities is less than the number of activities the entity is sent through the True node back to the consider node where we simulate the actions of the next

activity. The entity loops between the Consider node and the Next_Activity node until all the activities are simulated, after that it leaves the Next_Activity node through the False branch to the Next_Stage node where the stage variable is incremented. If the stage variable is less than n the entity is sent back to the DUMMY_OBJECT node where all the activities are simulated for another stage, else the entity leaves through the False branch to the Exit_Simulation node where it is destroyed.

To create a Monte Carlo simulation a set of entities need be injected in the system. That is why we inject n_s entities, each resembling one simulation of the system. The results of all the simulated entities are aggregated and stored in an excel file. Once all the runs are completed the data is read by Anylogic from these files and stored in an SQL database. We resorted to this style of data analysis because we wanted to split the analysis section from the simulation section, this way we would record all the simulation output at a very raw level, then compute all the metrics. Following this policy, we didn't have to run the simulations again every time we wanted to compute a new metric. In the next section we explain how the data is manipulated to get the needed results.

C.2 Data Analysis

At a first, the results of the simulation is dumped into an excel document. In the document are computed, where all the formulas are set before hand. The simulation software dumps the data into the excel document, and then asks excel to update all the results according to the new data, it reads the data and then dumps them back to the SQL server as a final destination to the processed data. The results are stored in a results table. The fields of this table are described in table 2.

Field Name	Data Type	Description
fb	float	Fixed cost of considering information for activity B
bb	float	Fixed cost of performing rework for activity B
fc	float	Fixed cost of considering information for activity C
bc	float	Fixed cost of performing rework for activity C
fd	float	Fixed cost of considering information for activity D
bd	float	Fixed cost of performing rework for activity D
evolution	varchar(255)	Evolution Type
[R-value]	float	λ
[B-2]	float	Number of Times B consider information in 2 activity model
[B-3]	float	Number of Times B consider information in 3 activity model
[B-4]	float	Number of Times B consider information in 4 activity model
[C-3]	float	Number of Times B consider information in 3 activity model
[C-4]	float	Number of Times B consider information in 4 activity model
[D-4]	float	Number of Times B consider information in 4 activity model
[PVIB-3]	float	$PVI_{2,3}^{B}$
[PVIB-4]	float	$PVI_{2,4}^{B}$
[PVIC-4]	float	$PVI_{3,4}^{C}$
[ReworkB-2]	float	Amount of rework done by B in 2-activity model
[ReworkB-3]	float	Amount of rework done by B in 3-activity model
[ReworkB-4]	float	Amount of rework done by B in 4-activity model

[ReworkC-3]	float	Amount of rework done by C in 3-activity model
[ReworkC-4]	float	Amount of rework done by C in 4-activity model
[ReworkD-4]	float	Amount of rework done by D in 4-activity model
[EarlyB-2]	float	Number of Times B consider information in 2 activity model between time [1-3]
[EarlyB-3]	float	Number of Times B consider information in 3 activity model between time [1-3]
[EarlyB-4]	float	Number of Times B consider information in 4 activity model between time [1-3]
[MediumB-2]	float	Number of Times B consider information in 2 activity model between time [4-6]
[MediumB-3]	float	Number of Times B consider information in 3 activity model between time [4-6]
[MediumB-4]	float	Number of Times B consider information in 4 activity model between time [4-6]
[LateB-2]	float	Number of Times B consider information in 2 activity model between time [7-9]
[LateB-3]	float	Number of Times B consider information in 3 activity model between time [7-9]
[LateB-4]	float	Number of Times B consider information in 4 activity model between time [7-9]
[EarlyC-3]	float	Number of Times C consider information in 3 activity model between time [1-3]
[EarlyC-4]	float	Number of Times C consider information in 4 activity model between time [1-3]
[MediumC-3]	float	Number of Times C consider information in 3 activity model between time [4-6]
[MediumC-4]	float	Number of Times C consider information in 4 activity model between time [4-6]
[LateC-3]	float	Number of Times C consider information in 3 activity model between time [7-9]

[LateC-4]	float	Number of Times C consider information in 4 activity model between time [7-9]
[EarlyD-4]	float	Number of Times D consider information in 4 activity model between time [1-3]
[MediumD-4]	float	Number of Times D consider information in 4 activity model between time [4-6]
[LateD-4]	float	Number of Times D consider information in 4 activity model between time [7-9]
[DurationB-2]	float	$\overline{T}^{B}$ for 2 activities model
[DurationB-3]	float	$\overline{T}^{B}$ for 3 activities model
[DurationB-4]	float	$\overline{T}^{B}$ for 4 activities model
[DurationC-3]	float	$\overline{T}^{C}$ for 2 activities model
[DurationC-4]	float	$\overline{T}^{C}$ for 3 activities model
[DurationD-4]	float	$\overline{T}^{C}$ for 4 activities model
[Totalrework-2]	float	Amount of rework for all activities in 2 activity model
[Totalrework-3]	float	Amount of rework for all activities in 3 activity model
[Totalrework-4]	float	Amount of rework for all activities in 4 activity model
[Sensitivity-Type]	varchar(255)	Type of sensitivity function
Slope	float	Slope of sensitivity function
initialinfo	float	Proportion of endogenous information out of total information (%)
casenum	varchar(255)	Case Description

Table 2: Description of the fields that make up the SQL table's structure.

After the simulations are run with all the possible iterations, the data is stored in the table above. A set of queries and views are run on top of that data to mine the results and insights. It is through these aggregated reports that we drew out our observations that led to the listed insights.

APPENDIX D

MULTI-ACTIVITY MODEL APPLIED FOR A THREE-ACTIVITY EXAMPLE

In this appendix we show how the multi-activity compact model can be expanded to the regular formulation for the three-activity model (A-B-C). The purpose of this exercise is to illustrate the method of calculating the decision policy using the model.

As shown in Figure 11, there are four policies to compare at every time stage: No activity considers information, B considers information alone, C considers information alone, both B and C consider information. This is shown when the DP formulation for the three activity model as shown below:

$$R_i(\boldsymbol{r^m}) = \min_{\substack{J^k=0,1,\\ k=2,\dots,m}} \sum_{k=2}^{m} J^k f^k + \mathbf{E}^2\left[\mathbf{E}^3\left[\dots \mathbf{E}^m\left[\sum_{k=2}^{m} J^k Z^k\{\alpha^k(i,r^k) \quad \boldsymbol{p_i^B} \quad + R_{i+1}(\boldsymbol{r^k} + \mathbf{J} \wedge \mathbf{Z} \wedge (\mathbf{i} - \boldsymbol{r^k}))\right]\right] \dots\right]$$

$$R_n(\boldsymbol{r^k}) = \sum_{k=2}^{m} (\alpha^k(n,r^k) + \beta^k + f^k)$$

So for the four cases we get the following:

J^B=0, J^C =0 : $\mathrm{R}_{i+1}(\boldsymbol{r^C})$

J^B =1, J^C =0: $f^B + \mathrm{R}_{i+1}(\boldsymbol{r^C})p^B(i) + (1 - p^B(i))\,[\alpha^B(i,r^B)+\beta^B + \mathrm{R}_{i+1}(\boldsymbol{r^C}=\{i,r^C\})]$

J^B =0, J^C =1: $f^C + \mathrm{R}_{i+1}(\boldsymbol{r^C})\,p^C(i,r^B) + (1 - p^C(i,r^B))\,[\alpha^C(i,r^C)+\beta^C + \mathrm{R}_{i+1}(\boldsymbol{r^C}=\{r^B,i\})]$

J^B =1, J^C=1: $f^B + f^C + p^B(i)\;[\mathrm{R}_{i+1}(\boldsymbol{r^C})\,p^C(i,r^B) + (1 - p^C(i,r^B))\{\alpha^C(i,r^C) + \beta^C + \mathrm{R}_{i+1}(r^B,i)\}] + (1-p^B(i))\,[p^C(i,r^B)\{\alpha^B(i,r^B)+\beta^B + \mathrm{R}_{i+1}(\boldsymbol{r^C}=\{i,r^C\})\} + (1 - p^C(i,r^B))\{\alpha^B(i,r^B)+\beta^B + \alpha^C(i,r^C) + \beta^C + \mathrm{R}_{i+1}(\boldsymbol{r^C}=\{i,i\})]$

Thus the resulting 3-activitiy DP would be:

$$R_i(\boldsymbol{r^C}) = \min\{\mathrm{R}_{i+1}(r_B,r_C),$$
$$f^B + \mathrm{R}_{i+1}(\boldsymbol{r^C})p^B(i) + (1 - p^B(i))\,[\alpha^B(i,r^B)+\beta^B + \mathrm{R}_{i+1}(\boldsymbol{r^C}=\{i,r^C\})],$$
$$f^C + \mathrm{R}_{i+1}(\boldsymbol{r^C})\,p^C(i,r^B) + (1 - p^C(i,r^B))\,[\alpha^C(i,r^C)+\beta^C + \mathrm{R}_{i+1}(\boldsymbol{r^C}=\{r^B,i\})],$$
$$f^B + f^C + p^B(i)\;[\mathrm{R}_{i+1}(\boldsymbol{r^C})\,p^C(i,r^B) + (1 - p^C(i,r^B))\{\alpha^C(i,r^C) + \beta^C + \mathrm{R}_{i+1}(r^B,i)\}] + (1-p^B(i))\,[p^C(i,r^B)\{\alpha^B(i,r^B)+\beta^B + \mathrm{R}_{i+1}(\boldsymbol{r^C}=\{i,r^C\})\} + (1 - p^C(i,r^B))\{\alpha^B(i,r^B)+\beta^B + \alpha^C(i,r^C) + \beta^C + \mathrm{R}_{i+1}(\boldsymbol{r^C}=\{i,i\})]\}$$

APPENDIX E

CALCULATING THE EXPECTED QUALITY OF INFORMATION USING THE TW0-ACTIVITY CHAIN HEURISTIC

In this appendix we present an example of the calculation done to estimate the quality of information being sent from upstream activities for an activity k. We shall assume that the decision policy for activity $k-1$ is given. Also the quality of information being used by $k-1$ is given, and depends only on time i. Assume the decision policy for the activity $k-1$ is as follows:

r^{k-1}/i	1	2	3	4	5	6	7	8	9	10
0	0	0	1	1	1	1	1	1	1	1
1		0	0	1	1	1	1	1	1	1
2			0	0	1	1	1	1	1	1
3				0	1	1	1	1	1	1
4					0	0	1	1	1	1
5						0	1	1	1	1
6							0	1	0	1
7								0	0	1
8									0	1
9										1

Table 3: The decision policy of activity $k-1$: $D^{k-1}(i, r^{k-1})$

In the above table the first vertical column represents the possible values for r^{k-1}, these values represent the state of the activity $k-1$. The first horizontal column represents the time stages i. Thus the table represents a matrix of two dimensions, r^{k-1} and i. The activity considers information when $D^{k-1}(i, r^{k-1})$ =1 and disregards the information otherwise. Also assume that $p^{k-1}\left(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-2}}\right)$ is linear and has the values presented in table 4:

i	1	2	3	4	5	6	7	8	9	10
$\boldsymbol{p^{k-1}\left(i, \bar{r}_i^{k-2}\right)}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1

Table 4: Values of the function $p^{k-1}\left(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-2}}\right)$ as a function of time.

Using formula 6 in Chapter 5 we can compute the probability matrix. The results are shown in table 5.

r^{k-1} / i	1	2	3	4	5	6	7	8	9	10
0	1	1	0.7	0.42	0.21	0.084	0.0252	0.0050	0.000504	0
1	0	0	0	0	0	0	0	0	0	0
2		0	0	0	0	0	0	0	0	0
3			0.3	0.3	0.15	0.06	0.018	0.0036	0.00036	0
4				0.28	0.28	0.28	0.084	0.0168	0.00168	0
5					0.36	0.36	0.108	0.0216	0.00216	0
6						0.216	0.216	0.0432	0.0432	0
7							0.548	0.5488	0.5488	0
8								0.3609	0.36096	0
9									0.04233	0
10										1

Table 5: Probability matrix for activity $k-1$ defining the probability that activity $k-1$ would be in state r^{k-1} at time *i*.

Now that we have all the probabilities for the stages and states, we can calculate the expected state of activity $k-1$ at every time *i*. We do so by using formula 7 of Chapter 5. The expected $\bar{r}_i^{k-1}$ is presented in table 6.

time	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}$	0	0	0.9	2.02	3.37	4.396	6.0676	7.17448	7.388104	10

Table 6: Calculated values for $\bar{r}_i^{k-1}$ as a function of time *i*.

After calculating $\bar{r}_i^{k-1}$ we append it to the vector $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-2}}$ to get the state vector $\bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}$. Thus getting $p^k\left(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}\right)$ that we can calculate using equation (1). Thus we get the quality of information used by activity $k-1$: $p^k\left(i, \bar{\boldsymbol{r}}_{\boldsymbol{i}}^{\boldsymbol{k-1}}\right)$. We use this function to calculate the policy using the two-activity version of the multi-activity model.