Dynamic Modeling of Exchanging Preliminary Information in Concurrent Development Processes

BY LARS NYGREN

Submitted to the Faculty of Engineering in
Partial Fulfillment of the Requirements for the Degree of
Sivilingeniør in Information and Communication Technology

Abstract

Without doubt the ability to develop products fast, cheap and with high quality is critical to survive in many markets. This necessity has lead to an increasing number of manufacturers switching from sequential, functional development to a team-based concurrent development. To be able to perform overlapping of processes successfully, the exchange of preliminary information is a vital factor. Up to this point there has not been a lot of literature covering this concept, and how different strategies, in regard to information exchange, affect project performance. This research investigates this concept.

To be able to draw some conclusions with regard to the research questions, a dynamic simulation model was built, to simulate the dynamics of concurrent development processes. This model was based on Ford and Sterman's (1998) dynamic model of product development processes. To this model I added structures to model the characteristics of the information being in exchanged, in terms of precision and stability, and the characteristics of the processes that are overlapped. For the latter, literature studies suggested that the upstream process evolution and the downstream phase sensitivity are important factors. The user can in my model manipulate these factors, and the behaviour of projects can be observed. I decided to measure project performance in terms of quality, cycle time and cost.

In my studies I examined how projects should be handled, depending on their sensitivity and evolution attributes. To test I used the two polar strategies, set-based and point-based strategy. Set-based strategy implies focus on stability in information exchanged, sacrificing precision. Point-based strategy is the opposite, i.e. precision is favoured in front of stability. Simulations showed that the latter strategy was most appropriate when the sensitivity is low in the downstream phases. Set based strategy

was most effective in projects where the sensitivity was high. Simulations finally showed that the concept of finalising early, i.e. committing to a design before all tasks are finished in the upstream phase should be facilitated when the upstream phase is fast evolving. That way cycle time could be reduced without sacrificing quality notably.

1 Introduction

1.1 Research Motivation

Without doubt the ability to develop products fast, cheap and with high quality is critical to survive in many markets. Therefore it is a necessity to improve the performance of product development projects. An example of this is the change of focus from sequential, functional development to team-based, concurrent development. Concurrent development arguably has at first sight many advantages; it is easy to understand that doing several tasks at a time instead of one after another is an effective way to save time. But it is also a fact that this way of working makes for much more complex projects, especially from a manager's viewpoint. Unfortunately many managers have not been able to keep up with the progress, causing projects to suffer from their inability to evaluate and estimate dynamic influence on performance. Therefore, there is an obvious need for research that can help project managers, etc. understand the dynamics of concurrent development processes.

To be able to run concurrent development processes, the exchange of preliminary information is a vital factor. Despite its obvious importance not much research has been done in this field, neither from a theoretical nor managerial perspective. There are not many studies showing the pros and cons of giving downstream phases a head start by passing down preliminary information from upstream phases.

1.2 Research Question

In this report I will present some of the written material that does exist on this specific phenomenon. Using a system dynamic model I intend to examine the concept of information exchange between phases. I will model the means project managers have to affect the outcome of overlapped projects. **The main object will therefore be to examine how these means should be applied to achieve an optimum result when overlapping development processes.** To be able to do this, I'm going to modify a model Ford and Sterman have made (Ford and Sterman, 1998). With the modifications, I will be able to simulate a wide array of different projects. To these possible projects I will apply different types of strategies for overlap, and decide from

the simulation output which is best suited under the given circumstances. The criteria for the success of the project I intend to use are project duration, quality and cost.

I will inspect some of the research that has already been done in this field, with focus on the two papers ”A Framework For Exchanging Preliminary Information In Concurrent Development Processes ” (Terwiesch et al. 1998), and ”A Model Based Framework to Overlap Product Development Activities” (Krishnan et al., 1997). Terwiesch et al. have qualitatively discussed the trade-offs involved with overlap of processes, but the concept has not been quantitatively formulated in an analytical model. By combining the theories presented in these papers in one model, I hope I will be able to shed some additional light to the concept of information exchange.

1.3 Research approach

I have chosen to use computer aided dynamic modelling, i.e. system dynamics, as the tool for my studies. Simulating instead of experimenting in a real life environment obviously has many advantages. First of all, trying out different possibilities can be done quite quickly on a computer, something that would be both time and cost consuming in real life. Another advantage is that the model can be used for educational purposes later on, i.e. as a tool to make managers and project leaders more aware of the effect their decisions will have on resources, time and work-cost, etc. The graphical interface provided by today’s dynamic modelling software contributes to that. One might of course argue that a computer model never would be as realistic as real life. Which, to some extent is true, but when concentrating on small environment, it is possible to create dynamic models that act very realistic. As dynamic models are made up entirely by mathematical equations, simulations obviously require computation of thousands of results per second when run, the exploding increase in computer power the latest decade obviously has made more and more realistic dynamic models possible. The fact that assumptions are presented as formal equations also make them unambiguous and explicit, which is strength when conclusions are drawn on the grounds of simulation outputs.

2 Concurrent Engineering

2.1 Concurrent Engineering and Preliminary Information

One of the most effective methods for improving project development processes in terms of time cost, is more and more extensive use of overlap of tasks. Obviously, doing several things at a time instead of one after the other (provided the necessary resources are present) is an effective means for increased competitiveness in a market, which demands faster and faster development times. Unfortunately it is not quite that simple. Usually it is not a coincidence that processes have been done in sequence. They were done that way because one of the processes requires the final outcome of the other to be finished, or even started. To exemplify this, imagine a project that involves the development of an electronic device. You can not make the casing before you know the final dimensions of the component board. So if it is desired to do these two tasks in parallel you are presented with basically two options. You can choose to lock the upstream phase at an early, in this case the development of the component board, to a product specification, thereby giving the downstream phase (design of the casing), concrete information to work on. This option has disadvantages, as it leaves no room for improvisations or changes of any kind in the upstream phase, which ultimately may lead to problems if anything unexpected were to occur. Maybe the actual construction of the board reveals that two of the components interfere with each other. As a result, the placing of the components has to be changed and the specified dimensions of the board altered. Also market changes can force changes in the specification. A situation where there is a customer demand for smaller and smaller electronic devices would for example require flexibility. So this solution is arguably not advisable for most projects. That leaves us with the second option. Instead of locking the upstream phase to the specification, you accept the fact that changes will happen, and that rework will be required in the downstream phase. When choosing this option, some assumption on the final result of the upstream phase has to be done, in order for the downstream phase to start early. To return to our example, the team developing the component board can’t say the dimensions for sure before they have actually completed the work. So assumptions made are likely to be subject to change. To achieve an end result where the component board and casing actually fit together, information on changes in the board development phase has to be passed.

That way casing designers can continuously adjust their work, to the new information they receive from the upstream phase.

This may seem like a simple solution, but in practice there is a whole art linked to the concept of exchanging preliminary information. The reason is the vast array of different project types, which unfortunately lead to a situation where there is no single method for running concurrent processes that work for all. There are numerous decisions that must be done when running concurrent processes, all of them which probably will affect the final outcome of the project in some way. Put simply there are three important decisions that has to be done when overlapping development processes:

- When to start the downstream phase by providing preliminary information from the upstream phase.
- When and how often to passed information once the downstream phase has started.
- When to finalise work in the upstream phase.

Firstly, project managers have to decide how early the downstream phase should be provided with preliminary information (in my example the design of the casing). Obviously, the earlier information is passed the earlier the downstream phase can start. But there is a trade off involved here. Consider for example a project where it takes quite a lot of work before the upstream phase gets a clear idea about the work at hand, and how it should be solved. In such a case there is likely to be huge changes in the early stages of the upstream phase. Which also will lead to large amounts of rework in the downstream phase to accommodate changes made in the upstream phase. As a consequence, it might be wiser to actually withhold information, until certainty is increased, and thus unnecessary iteration in the downstream phase is avoided. On the other hand, if estimations on the final result can be made at an early point without sacrificing certainty too much, there is a good chance that development time will be saved.

The size intervals at which information is passed once the downstream phase will also affect the outcome of the project. As I have said, we are dealing with a situation where the downstream phase is dependent on results from the upstream phase to be

able to continue with their own work. When intervals are small, all work completed in the upstream phase will more or less immediately be available to the downstream phase. That means the downstream will get the results they need to proceed with their own work as soon as possible. The trade-off is that immediate release of information allows the upstream phase no time to check the correctness. If the upstream phase later find the work they did to be wrong, both phases would have to do rework. Larger intervals will on the other hand give the upstream phase more time to review their work, before results are passed to the downstream phase. That will lead to more certain information and thus less rework in the downstream phase. The downside is that large intervals may lead to work standing still in the downstream phase because they simply do not have enough information to proceed. This situation where the downstream is waiting for additional information from the upstream phase is called *starvation*.

The third mean project managers can use to affect the outcome of concurrent development processes is *finalisation* in the upstream phase. Finalisation implies freezing the work to a set solution. Once work is finalised in the upstream phase, the workers can not make changes that stray from this specific design, but are forced complete the remaining tasks according to the specification that is agreed upon.
The sooner upstream workers commit to a specific design, the sooner the downstream phase will have certain information to work on, and thus the sooner they are able to finish their work. Premature finalisation is unfortunately not without disadvantages. Obviously, when committing to a set specification at an early time, leaves the upstream phase little room to experiment and pursue different possible solutions. Also the upstream phase will be very inflexible, which can lead to daring consequences if anything unexpected were occur. In other words, early finalisation in the upstream will lead to quality loss.

These are some of the questions that need to be considered. One should also note that not all processes in development projects are advisable to run concurrently. For example, to start development of the product, when the design and concept is still in the planning/specification stage would probably not be a wise move. The planning/specification stage is often a very tumultuous phase where big changes can happen quick and often, something that would lead to huge amounts of rework in downstream phases, possibly to the degree where nothing is gained from starting

early. In other words, it varies how much impact a change in an upstream phase has on downstream phases. For some projects, changes in upstream phases are relatively easy accommodated by the downstream phases. For other projects the opposite is the case, like in my example above.

2.2 Precision, stability, evolution and sensitivity

Above I presented some key decisions that have to be made when running concurrent development processes. I also argued that the impact these decisions will have on the outcome of a project would differ based on the characteristics of the specific project at hand. To be able to model overlap of processes effectively, it is therefore obvious that I need to find some relevant way of describing the characteristics of different projects. That way I can accurately simulate the impact the release of information has on downstream phases, and be able to draw some conclusions on how different projects should be handled by managers. In the literature that already exists on this subject, I have found some key terms when describing the characteristics of projects and the properties of the information being exchanged. In this chapter I will explain these terms, i.e. precision, sensitivity, stability and evolution, and at the same time try to give the reader an impression of how these terms relate to decisions I presented above.

I have previously mentioned Terwiesch et al. (1998). In their paper, they examine the managerial decisions that a development process faces in the exchange of preliminary information. They introduce a framework that operationalizes the concept in the form of two dimensions, precision and stability. Precision the way Terwiesch et al. see it is a measure of how accurate the information passed to the downstream from the upstream phase is. To get back to the example with the development of the electronic device, precise information would be the exact measurements of the component board. In this case, maximum precision would mean exact measurements, for example w*h*d=25 cm*23 cm*6 cm. Of course passing exact information early in the design of the component board makes it likely that it will be changed later on. This is where stability comes in, which is a measure for how prone the information is to change later on. In this case precision is high, whereas stability is considered to be low.

Let us say that there is some uncertainty linked to how deep the circuit board will be. Instead of the upstream phase telling downstream that the depth is going to be 6 cm, they say that it's probably going to be somewhere between 4 and 9 cm. This is of course less precise, but on the other hand this information is less likely to change. In other words, this information has higher stability. As the project progresses the upstream will get closer and closer to the final result, and thus this interval will decrease until it reaches the final value. For every task the upstream workers do, they gather some knowledge of what they are doing causing them to get closer and closer to the final result. This gained knowledge isn't necessarily increasing proportionally with tasks done. Sometimes knowledge is gained quicker at the earlier stages of the phase, i.e. they get close to the final result early, but then use a lot of time to finally narrow it down. In other cases it takes a lot of work before the phase seems to get anywhere near the final result, until they suddenly break through and quickly move toward the final result. The figure below shows typical examples of fast and slow evolution:

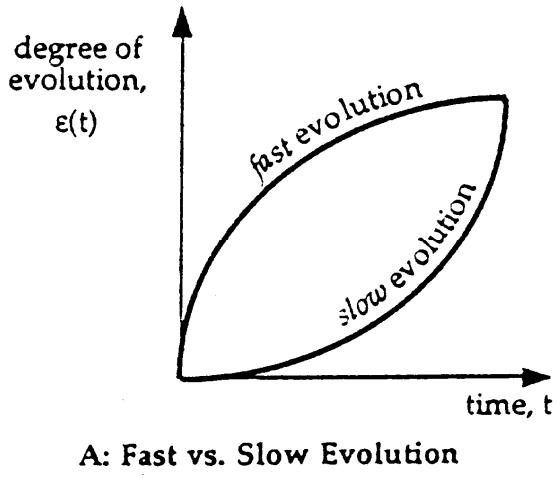


Figure 1: Fast and slow evolution

This is something Krishnan, et al. (1997) also mention in their paper. They use the term "evolution" for this phenomenon, and I have chosen to do the same. It is perhaps easy to mix the two terms evolution and precision. To explain the difference, the precision Terwiesch et al. (1998) talk about is the just a measure of how precise the information being passed is, i.e. to which degree it has the same form as a final result. Evolution on the other hand is a measure of how much knowledge the upstream phase has actually gained about the work at hand. Precision and evolution are therefore not necessarily the same. For example can 100 % accurate estimates and assumptions on

the final result be made, even though little work has been actually been done. This can be seen in relation to what I said earlier about there being a difference, in how much the upstream phases know, and how much of their knowledge they actually make accessible to downstream phases.

To continue the example above, let's say the upstream phase, after having done 10% of the work, says to the downstream phase that they think the board measurements are going to be 25 cm*12 cm*13 cm. These are pretty much 100% precise data the downstream can use as input for the work they're doing. Obviously after having only done 10% of the work, this information is likely to change. So in other words, releasing highly precise information (i.e. estimates on the final result) while the evolution in the upstream phase is still low leads to instability. The last term that needs to be explained is sensitivity. Sensitivity is a measure of the impact an error created in the upstream has on the downstream. This is highly related to what I said earlier about different projects accommodating changes in the upstream phase easier than others. Krishnan et al. (1997) define sensitivity as the "relationship between the duration of the downstream iteration and the magnitude of change in the upstream phase". High sensitivity would imply time-consuming rework in downstream for every error inherited, whereas low sensitivity would mean that changes could be quickly absorbed. In some cases, the sensitivity changes over the duration of the phase. For example downstream processes are often more sensitive to changes in upstream at the end of the phase cycle. That would be the case where the downstream phase more or less has to start over every time changes are made in the upstream phase.

2.3 Strategies for running concurrent development processes

I have established some terms that I will use to define the quality of information. At the start of this chapter I also suggested some different ways of running development strategies, and what trade-offs they involved. These different types of strategies are already documented in literature. Terwiesch et al. talk of two extreme types of strategies, set-based and point-based. As I will be referring to these terms later I will do a small introduction to them here. Simply put, point-based strategy involves exchanging accurate information for the duration of the overlap time between the phases. As I have argued earlier, passing precise information in the early stages of the

upstream phase implies uncertainty, i.e. low stability. In other words this strategy necessarily leads to rework in the downstream phase.

On the other hand, set-based strategy implies passing stable, in other words more or less 100 % certain information, for the entire overlap time. The obvious gain of such a strategy is that less rework will be necessary in the downstream phase. The disadvantage is that once the upstream phase passes certain information to the downstream, they close the door for a set of solutions, because that part of the work can not be changed later on. The longer the upstream phase hold on to the work, the more possible improvements they are likely to find. But holding on to the work in the upstream phase will also lead to less available information for the downstream phase. Obviously, it is a waste of resources to do nothing in the downstream phase while waiting for additional information. In some cases it may therefore be useful to compensate lack of data by developing alternatives. In other words, the downstream phase start making possible solutions that might fit the end result of the upstream phase. This method is most useful when there are a limited set of possible outcomes of the upstream phase, and/or the downstream are skilled at choosing alternatives that are likely to include the final outcome of the upstream phase.

To sum up I have decided to plot these strategies in a precision vs. stability chart (figure 1). This figure is the same that can be found in Terwiesch et al. (1998). To explain the chart in short, to be able to reach the final result, both precision and stability must be 100 %. ‘Beginning’ in the lower left corner implies the start of the project where both stability and precision are zero. ‘Final’ is the end of the project where both are maximal. For the point based strategy, precision is maximal from the very start of the project, and the stability is slowly gained as the project evolves. For set-based the opposite is the case. Stability is maximal from the beginning, and then precision increases as more and more knowledge is gained in the upstream phase. This should be consistent with what I have said above. Of course there are intermediate ways to reach the final result, but I have chosen to show these extreme cases for the sake of clarity.

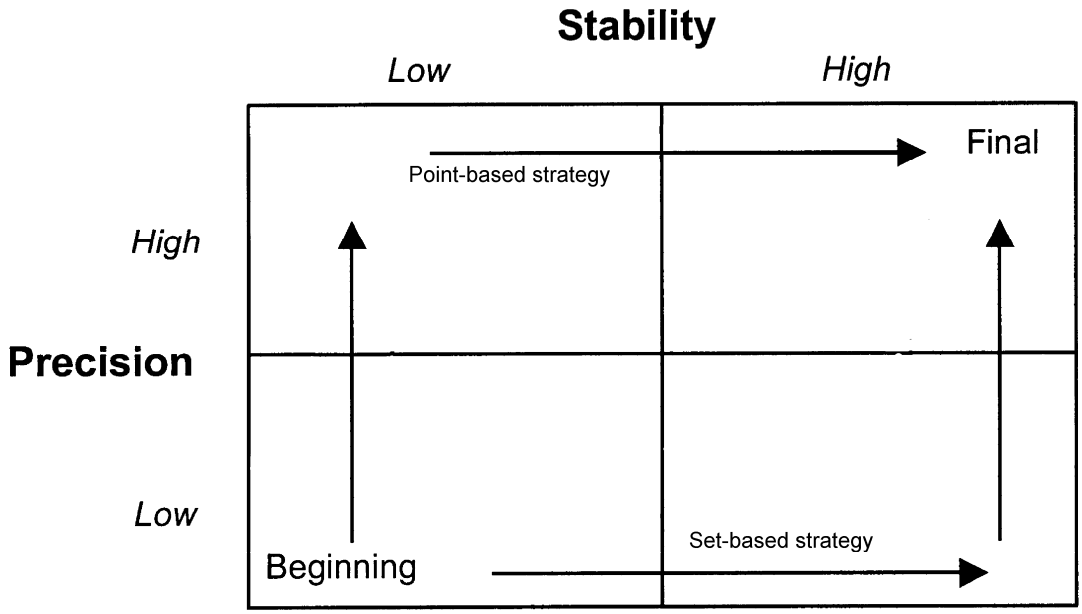


Figure 2: Strategy Plot

2.4 Criteria for the degree of success

As I have pointed out in this chapter there are numerous factors that affect whether overlap of phases can be considered successful. To be able to come to any conclusions on whether a certain strategy for exchange has been effective given a set of circumstances, I will need to have some criteria that decide the success of the project. Obviously, as the main object of facilitating a concurrent approach is to save product development time, one of these criteria has to be time cost. Second, I have pointed out the danger of quality suffering as a result of overlapping processes. So that has to be another criteria. Another disadvantage of concurrent engineering is that rework and redundant work will happen in downstream phases. The amount of rework will be determined by which strategy the project management chose when overlapping. Obviously if the cost of rework is very expensive then overlapping is not advisable. Therefore I think the cost of rework and basework needs to be modelled and used as the third criteria.

Of these criteria, managers have to decide which ones that are most important, and from that perspective choose a strategy that makes for an optimum result.

3 The Model

3.1 Introduction to the model

The model I have used in my studies is a modified version of the model the authors present in their work (Ford and Sterman, 1998). I am not going to prove the validity of that model in any way in my report, as Ford and Sterman already have done that thoroughly in their work. In this chapter I will give a basic explanation of how the model works. In addition, I will examine to which degree Ford & Sterman's model makes it possible for me to simulate and find answers to the questions mentioned in the first chapter. Eventually, I will point out some enhancements that need to be done before I can start my studies. My version of Ford & Sterman's model consists of two phases, named upstream and downstream. The upstream phase is the information passing process whereas the downstream is the information receiving process. In other words, it is dependent on data from the upstream phase to be able to do its own work. Having only two phases is of course a simplification of a project development process, but it also enables me to fully concentrate on the specific phenomena I wish to examine. Each of the phases has two co-flows, in addition to the main structure. To give the reader not familiar with system dynamics a basic idea of what I mean by structures and co-structures, I have provided a figure of the upstream main structure.

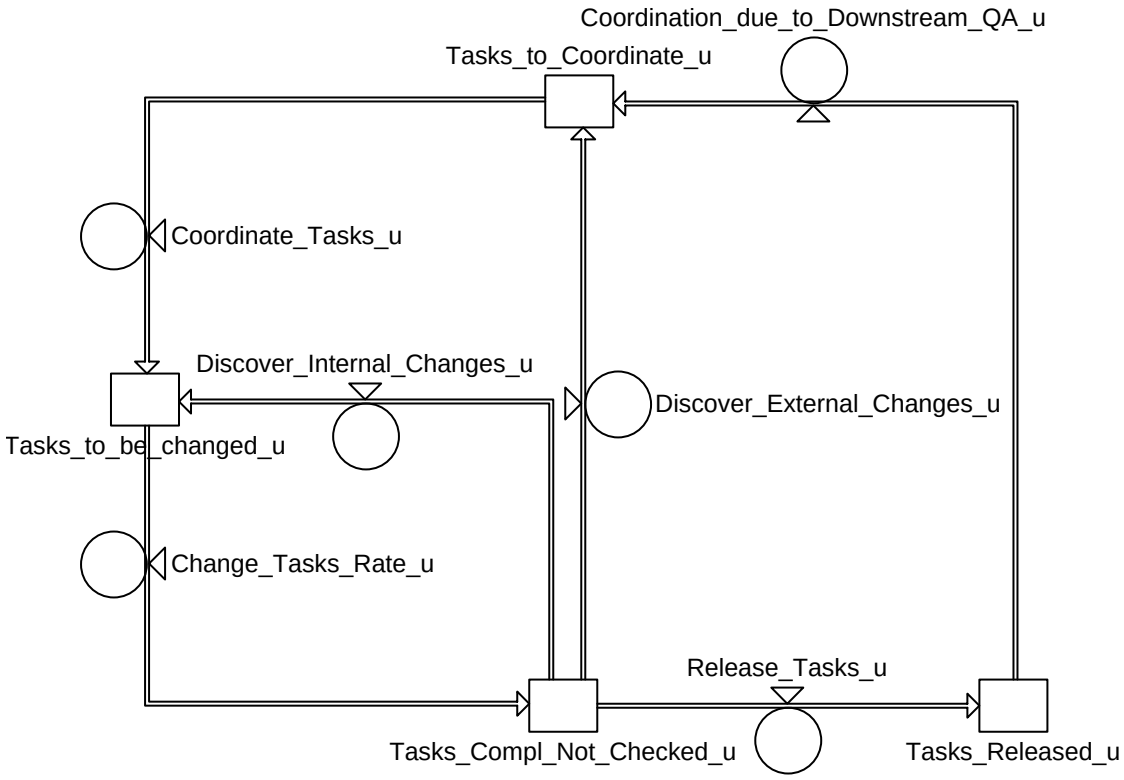


Figure 3: Main Structure Outline ('u' depicts upstream phase)

As one can see, the structure is made up of squared boxes, called *stocks*, connected to each other with pipes, called *flows*. These flows decide at what rate the contents in the stocks are moved from one to the other. In this structure, the contents in the stocks are tasks. I will use the same definition of a task as Ford and Sterman do, i.e. that it is an atomic unit of development work, such as installing a steel beam or selecting a pump. I will assume all tasks to be uniform in same size, and to be small enough to either be correct or erroneous. In other words, a task can not be partially flawed. To explain the concept of stocks and flows a bit further, let us take a closer look at a smaller piece of the model.

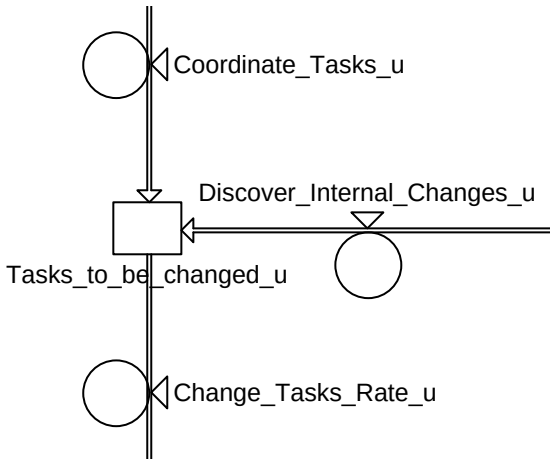


Figure 4: 'Tasks to be Changed'-stock

For this particular stock, the number of *Tasks to be changed*, there are two flows going in to the stock and one going out. The two rates "*Discover_Internal _ Changes_u" and "Coordinate_Tasks_u"* put together represents the rate at which tasks that need to be changed are discovered (a more specific explanation of "intra-phase" and "co-ordinate" will be provided later). In other words, all erroneous tasks found are moved to the tasks to be changed stock. Obviously, corrected errors no longer need change, so they should be moved out of the tasks to be changed stock. The outgoing flow, "*Change_Tasks_Rate_u"* decides at what rate the quality assurance workers in the project manage to correct errors, and therefore also how quick they are moved out of the stock.

Both the main structure and the external and internal error co-flows are basically identical to the ones Ford and Sterman have described in their work, so I will only do a short explanation of these here. I will get back to the difference between external and internal errors later, so do not worry about those terms for now. The main flow follows the tasks as they are being initially completed, approved and finally released. This flow is made out of five stocks, tasks not completed, tasks completed not checked, tasks to be changed, tasks approved, tasks to be co-ordinated and tasks

released (see figure 3 above). There are basically three ways a task can move through the main flow. Tasks in the tasks not completed stock, which initially is the phase scope, are moved out at the rate of completing tasks. The number of people assigned to basework and the time completing one task takes decides this rate. They end up in the tasks completed not checked stock. Now the tasks take different routes. A correct task will be approved by the quality assurance and placed in the approved tasks stock, and eventually released. Tasks are released when a sufficient amount, determined by the user, of tasks are approved. An erroneous task can take three paths. One possibility is that the error is discovered by quality assurance and therefore is sent to the tasks to be changed stock, where it's waiting to be redone. (The rate of rework is similarly defined as the basework rate). When the task is redone it's moved back to the tasks to be checked stock. From there it will follow the path of a correct task (provided the rework people didn't made a mistake when trying to correct the error. In that case it will be moved back to the tasks to be changed stock). The second possibility for faulty tasks is that the quality assurance workers fail to discover the error. The task will then be considered as correct and move along with those tasks to the released tasks stock. Eventually, errors will be discovered by a downstream phase. The tasks will then be removed from the tasks released stock and moved to the tasks to be co-ordinated stock, at the rate quality assurance workers in downstream phases are able to find errors inherited from upstream. From the tasks to be co-ordinated stock, these tasks are eventually moved to the tasks to be changed stock. In practice, what happens in between is that project managers had a meeting where they have made each other aware of mistakes they have discovered other phases have done. In the meeting, phases involved will have to come to some conclusion on how the tasks should be, before all can change work accordingly. The last path an erroneous task can take is from task to be changed to tasks to be co-ordinated via the external error flow. As the name implies, these are tasks that are found to contain errors inherited from upstream phases. In practice, this means that this task depended on results from an upstream phase. When this information later turns out to be wrong, obviously both phases have to do corrections. This is what happens when the task is sent to the tasks to be co-ordinated stock, and then eventually corrected via the tasks to be changed stock.

As the names imply, the co-structures keep track of internal and external errors made. Internal errors are errors made as a result of mistakes within the phase, whereas external errors are mistakes made due to false information from the upstream phase. The difference between the main structures and the co-structures is best illustrated by an example: Imagine 100 tasks are completed in the upstream phase, and that 10 of these are erroneous. Observing the simulation you will see that 100 units, in this case tasks, are moved from the tasks not completed stock to the tasks completed not checked stock. In the internal changes co-structure you will see that 10 units, meaning changes, are moved from the potential changes stock to the undiscovered changes stock. In other words, the co-structures enable us to observe the life cycles of the changes the same way the main structures let us follow tasks.

3.2 Limitations and Assumptions

I have used the equations for the model presented by Ford and Sterman (1998) to construct a base model I could as basis when trying to simulate the concept of exchange of preliminary information. I will get back to how well Ford and Sterman's model is suited for simulating process overlap as it is, and eventually what needs to be added or modified. For now I will discuss some of the limitations I have made compared to the model by Ford and Sterman.

First of all I have decided to only have two phases, one called upstream and a second called downstream. Like I explained in the last chapter, the downstream phase is dependent on results from the upstream phase to be able to complete its own work. In practice, there would most likely be information passing in two directions. For simplicity's sake I have decided to have information flow in just one direction, i.e. from the upstream to the downstream phase. In a real-life situation, one could think of the two phases as taking turns being the upstream and the downstream phase for each information exchange, i.e. that the phases are interdependent. Furthermore, Ford and Sterman talk of external and internal changes in phases, where external changes imply inherited changes from the upstream phases. Since there are no upstream phases for the upstream phase, no external co-structure will be needed for the upstream phase. Changes made in the upstream phase will have an impact on the amount of rework in the downstream phase, and thus plays an important part when modelling overlap of processes. For that reason I kept the internal changes co-structure for the upstream phase. The changes made by the upstream phase and released downstream, will be

regarded as inherited changes in the downstream phase, In Ford and Sterman's model, inherited changes are tracked in the external co-structure, so this was kept in the downstream phase. Like I said above, I will regard the exchange of preliminary information as being one-directional. Therefore I it seems logical to disregard internal changes made in the downstream phase, as they will have effect on the upstream phase. In other words, the internal changes co-structure was also left out. Note that I am therefore also assuming that probability of generating a change when correcting a change in the downstream phase. Correcting a change in the downstream phase implies accommodating a design change made in the upstream phase. I am assuming that once the downstream phase are made aware of these changes, the will do this perfectly according to the new information.

I will also assume that the work available in the downstream phase is only decided by the amount of information given by the upstream phase. In other words, the internal concurrence relationship factor in Ford and Sterman's model was disabled for this phase. For the upstream phase both the external and internal concurrence relationship factors was disabled. In other words, all work will be available from the start of the phase, as the upstream is neither dependent of information from upstream phases, or of work being done in a certain order within the same phase. As for the workforce allocation, I have made the assumption that the rates of tasks being changed, co-ordinated, approved etc. are only constrained by the amount of time it takes to perform these activities, and not how many workers that at any one time is assigned to the various activities. I am in other words assuming an unlimited workforce. The duration for all the activities was set to 20, except for basework, which was set to 30 days. I find it eligible that completing a task for the first time by average is more time consuming than for example changing it at a later time.

3.3 The model and exchange of preliminary information

Before I can proceed, I will need to know how this model can be used to simulate the concept of preliminary information, and eventually what needs to be added or modified. In chapter 2, I explained some central aspects in conjunction with overlap of concurrent processes and the exchange of preliminary information. Now I will

examine how well these are covered in Ford and Sterman's model and eventually what needs to be changed or modified.

The first question that needs to be answered is how information is passed on in Ford & Sterman's model. Taking a closer look at the model reveals that the amount of work available in the downstream phase is determined by the external concurrence relationship factor. This variable is a function, with tasks released in the upstream phase as input. With this function the user can define the amount of overlap between the upstream and downstream phases. Depending on the setting they can be everything from completely parallel to completely sequential. What the external concurrence relationship variable does is to compute a number for the percentage of tasks available in the downstream phase, depending on the percentage of tasks released by upstream. Note that it's not information that is passed on per se, but rather results of upstream progress are made accessible to the downstream through release of tasks. This is of course not a problem, as the moment upstream release their work, it is also possible for downstream phases to extract the data they need to be able to proceed with their own tasks. So we can establish release of tasks as the information passing process in our model.

The concept of design changes is an important factor when dealing with preliminary information. Until now I have talked about the changes in Ford and Sterman's model as errors. When doing my studies, I think it is more important to look at changes due to concept changes in the upstream phase, as opposed to changes due to tasks being done incorrectly. The difference might not be obvious, so I will do a quick example to clarify. A design change could for example be that the upstream phase decides that the component board can be made smaller by changing the placement of the components. In this case upstream workers did not do a mistake the first time designing the board, they just came up with a way to improve the design. An error on the other hand would for example be if the upstream workers simply forgot one of the components in the design, something that would lead to the electronic device not working at all. So the question is if Ford and Sterman's model can be used to simulate design changes as it is, or if modifications will be needed. To answer this I will first do a slight recapitulation of what happens in Ford and Sterman's model when upstream phase releases changes. First changes are transferred to the external changes co-flow in the downstream phase, via the probability of external change in the

downstream phase. This probability is the density of error in the tasks released by the upstream phase. Then they are moved into the undiscovered stock as tasks are completed. Now comes the interesting part. From here tasks are discovered and moved to the tasks to be co-ordinated stock. The way Ford and Sterman' document this variable, it is the rate at which downstream workers are able to find errors made by the upstream phase. As I have decided to focus on design changes, this calls for some interpretation. W should see changes as the number of possible improvements in the upstream phase. Then the undiscovered changes stock in the downstream phase should contain the tasks will must be changed in the downstream phase, if these improvements were implemented in the upstream phase. That would make the discover external error rate rather the rate at which these changes are discovered. The question is who would make these discoveries. It is certainly not the downstream quality assurance workers. It seems more logical to assume that the upstream quality assurance workers find these possible improvements when reviewing the work. In Ford and Sterman's model, external changes are first discovered in the downstream phase and then reported to the upstream phase, who then correct them. This does not make perfect sense in light of what I have just said, so this is something I will need to address. In the next chapter I will discuss some possible solutions to this problem.

Another factor in the model that is affected by the switch of focus from errors to design changes is the probability of internal error in the upstream phase ("Probability_Intraphase_ Change_u"). In Ford and Sterman's model this is a constant, which does not make sense when talking of design changes. It seems logical that design changes are most likely to happen in the early stages of the project when the workers still have no clear idea of what the end result is going to turn out like. It is also reasonable that the probability of a major design change towards the very end of the phase, when workers pretty much have decided on the final result, is unlikely. These two assumptions suggest that the probability should be a factor that changes over time i.e. start at the maximal value and then decrease towards 0. I will suggest a concrete solution to this in "Model Enhancement" chapter.

To get back to the way Ford & Sterman's model handles overlap of phases; I feel that having only a single function determining the amount of overlap is a simplification, at best. Basing amount of overlap on tasks released is not accurate. As I have mentioned, projects evolve differently, so I think this part of the model would be more realistic if

I model evolution explicitly and base the overlap on that. In the last chapter I also pointed out the concept of withholding information in upstream, so that they can review their work and consequently release less errors downstream. This is not possible in Ford & Sterman's model. The problem is that the "release of tasks" rate is only determined by the number of tasks approved. That leaves no way to control number of errors released, and thus not stability. I will explain this further in the next chapter, Model Enhancement, and suggest a solution for this as well. Sensitivity to changes is not implemented in any way in Ford & Sterman's model, so that's another thing I need to add.

As for the success criteria, they are pretty much covered in Ford & Sterman's model. The time cost is easily extracted from a model such as this. The quality can be measured in terms of number of errors released, which can be found as stocks in the co-flows, named Released_Changes_i and Released_Changes_e respectively for internal and external errors. That leaves the project cost. Ford & Sterman have not modelled this per se. I have decided to see work cost as the amount of rework, as the basework will be the phase scope (1000 tasks), regardless of which strategy for overlap project managers choose.

4 Model Enhancement and Validation

In this chapter I look closer at the enhancements I suggested in the last chapter, and explain how I applied them to my model.

4.1 Stability

As I pointed out in the last chapter Ford & Sterman's model has some weaknesses when it comes to modelling exchange of preliminary information accurately. For example, I have argued that if the upstream phase waits with releasing work to the downstream phase, the upstream phase quality assurance workers have more time to review the work. In other words they can make sure that the information passed to the downstream phases is more stable, and consequently less rework in the downstream phase will be needed. In Ford & Sterman's model tasks completed and checked are moved to the approved tasks level waiting to be released. In Ford and Sterman's model there is a variable called "Release_Package_Size". This is a number in percent that decides how much work that has to approved before tasks are released. Zero means tasks are immediately released as soon as they are approved, whereas 100% means no tasks are released before all work in the upstream phase is approved. In my simulation earlier I used 20% as the setting for the "Release_Package_Size". In other words approved tasks are released as soon as 20% of the remaining work since last time tasks got released are approved.

Unfortunately, release package size has no affect on the final result when it comes to changes passed down to downstream phase. That is due to the fact that no quality assurance is performed on the tasks that are approved. Changes that slipped through quality control and wrongfully approved, will therefore never be found no matter how long the reside in the "Tasks_Approved_u"-stock. In other words we have no means for controlling the stability of the information passed to the downstream phase. To correct this I had basically two options. One possibility was to add a quality assurance flow from the changes approved level to the tasks to be changed level, i.e. make sure that the quality assurance workers also check tasks approved for possible changes. The equations for this flow would be similar to the quality assurance flows Ford and Sterman already have included in his model.

The second possibility was to completely remove approving of tasks from the model. There are several reasons for choosing this latter option. First, there is no mention of the concept of approving tasks in neither Terwiesch et al. nor Krishnan et al.'s work. In addition, adding the quality assurance flows to the approved levels like I suggested above, really is just a way to remove the effect the approve tasks concept has on the behaviour of the model in the first place. So I decided to remove all the stocks and flows in conjunction with approve of tasks from the model. Instead I connected the "Tasks_Completed_Not_Checked" levels directly to the "Released_Tasks"-stocks, facilitating a slightly modified version of the equations Ford and Sterman used for the flow connecting approved and released tasks. Experimenting with the "Release_Package_Size" variable, showed that a higher setting made for fewer errors passed to the downstream phase. As I have mentioned above, withholding release of tasks, gives the quality assurance more time to review their work, so this result was consistent with what I wanted to accomplish.

Unfortunately, the new structure had a major weakness. When setting the release package size quite small, an abundant amount of changes was released to the downstream phase. Inspecting the model closely revealed the reason for this model behaviour. The "Average_Release_Duration" is 1, leading to a situation where tasks are immediately released as soon as a sufficient amount of tasks has built up in the "Tasks_to_be_Changed"-stock. The setting for "Average_QA_Duration" was 20, i.e. the quality assurance workers plain and simple only had the time to review small amounts of the work before it was released. There are a number of ways to address this problem. One is setting quality assurance duration lower so tasks are checked at a higher rate. Also completion and change rate can be reduced to prevent build-up of tasks in the "Tasks_Completed_Not_Checked" stock. But these are just means to work around the problem. I therefore decided to modify the criterion for release of tasks. The fact that we are dealing with the stability of information being passed to the downstream phase suggested that it would be reasonable to shift the focus from tasks released to number of known changes for the release time switch.

Instead of releasing whenever a certain amount of tasks were finished, I decided tasks should be released as soon as the upstream phase has reworked their work to the point where it has reached a certain degree of certainty. In practice that would mean release

of tasks is stopped as long as the number of known changes is bigger than a user-defined percentage of the number of tasks not yet released. This would be like saying, that the upstream phase are not going to release anything before they have made sure there is at least less than (for example) 5 % errors in the work they have done so far (or since last time work was released)". The equation for the release time criteria looked like this after the modification:

(1) *IF(Release_Error_Density_Criteria_u>=(Known_Error_Density), Nominal_Release_Time_u,1E99)*

The release error density criteria is a user-defined number in percent, determining how small the known error density must be before the upstream phase is allowed to release any tasks. The known error density is the number of known errors divided by the number of tasks not yet released. In my model, known errors are represented as the sum of known internal changes and changes co-ordinated from downstream phases. In other words, that means the sum of errors we have done that we have discovered ourselves, in addition to the number of errors we have done that downstream phases has made us aware of and errors made because of false information from upstream phases. The amount of tasks completed not released is given by tasks to be co-ordinated, tasks to be changed and tasks completed but not checked, put together, which means all tasks we have completed but not released yet. The equation for the known error density was therefore generically defined as:

(2) *(Changes_to_be_Coordinated_i+Known_Changes_i+Known_Changes_e +Changes_to_be_Coordinated_e)/(Tasks_Completed_Not_Checked+Tasks_to _be_changed+Tasks_to_Coordinate)*

(Note: The i and e represents internal and external respectively.)

As there are no external changes in the upstream phase, both Changes_to_ Coordinated_e and Known_Changes_e equals zero. With this structure I found myself able to control the stability of the information released.

4.1.1 Validation of stability structure

When simulating the results should show that by setting the known error density release criteria lower the number of errors will decrease. In the table below you will see the number of errors released, as a function of the criteria. The probability of change is set to 0.5 for this test. Other values are set according to the basic initialisation settings that can be found in appendix.

Release_Error_Density_ Criteria_d	*1*	*0.5*	*0.2*	*0.1*	*0.05*	*0.01*
Changes_released_d	262	252	167	96	54	13

Table 1: Changes released as function of release density criteria

As we can see the number of errors released decreases, as the release criteria ("Release_Error_Density_Criteria_d") is set lower. This is consistent with what I wanted to accomplish with this structure.

4.2 Evolution

In the last chapter I argued for the importance of modelling evolution in order to simulate the impact of information exchange between phases accurately. Evolution is, as I argued in the first chapter, a measure of progress. To make the term a little bit more precise, evolution can be seen as the rate at which the upstream phase, in this case, is getting closer to the final result. An example: Imagine there are 10000 different ways to complete the work in the upstream phase, and that one of these ways is the most optimal. For every small piece of the work that is completed, the upstream gain a unit of knowledge, which take them a step closer to the 'correct' solution. Terwiesch et al. (1997) define increase of evolution as the rate that the phase discards possible solutions. To exemplify this, let us say that at some point the upstream phase has managed to discard 3000 of the 10000 initial possible ways. That would imply an evolution of 30 %. (This is of course a very theoretical example, but should nonetheless give the reader the basic essence of the evolution term).

To be able to model evolution, I needed to figure out what actions in the upstream phase would cause the above-mentioned units of knowledge to be gained. The logical

answer seemed to be that evolution should increase when tasks is initially completed (basework) and when tasks are changed (rework). I made a co-structure consisting of basically the same stocks and flows as in the main structure. The object was to be able to track evolution in the various stocks, in the same way I can track changes in the changes co-structure. In other words the units that is moved around in this co-structure should be evolution. At the beginning of the project evolution is zero. So all stocks should also initially be zero. The goal is to end up with 100% evolution. At the end of the phase all the units of evolution should be in the tasks released stock. That implies that the final solution is reached, and released to the downstream phase. Note that evolution may reach 100 % before all the tasks in the upstream phase are completed. The reason for this is that even though the upstream phase *knows* what the end result should be like, they will still have to complete the remaining tasks before the phase is finished.

I argued that evolution only should be increased when completing or reworking tasks. That means that all other rates in the co-flow should merely move evolution between stocks. We should think of each task as the carrier of a small amount of knowledge that lead the staff in the upstream phase closer to the final result. When tasks is moved between stocks, so are the units of knowledge they carry. In other words the rates in the evolution co-flow have to be closely related to the task rates from the main flow. Generically I defined the equations as the task rate from the main flow multiplied with the average amount of knowledge carried by the tasks in the stock driving the flow. The average amount of knowledge is the units in the evolution co-flow stock divided by the amount in the task stock. That gave me the following equations.

(3) *Release_Tasks_u*(Evolution_in_tasks_not_checked/Tasks_Completed_ Not_Checked_u))*

(4) *Coordination_due_to_Downstream_Quality_Assurance_u*(Evolution_in_ released_tasks/Tasks_Released_u)*

(5) *Coordinate_Tasks_u*(Evolution_in_Coordinated_tasks/Tasks_to_Coordinate _ u)*

(6) Discover_Interphase_Changes_u(Evolution_in_tasks_not_checked/ Tasks_ Completed_Not_Checked_u)*

(7) Discover_Intraphase_Changes_u(Evolution_in_tasks_not_checked/Tasks_ Completed_Not_Checked_u)*

That leaves the two activities where knowledge actually is gained, namely change and completion of tasks. Let us look at how change tasks rate affects the evolution co-structure. Obviously, when you change tasks you move the amount of knowledge carried by these tasks from the "Tasks_to _be_Changed_u"- stock to the "Tasks_completed_ not_checked_u"-level. That part is covered by this equation:

(8) Change_Tasks_Rate_u(Evolution_in_Tasks_to_be_Changed/Tasks_to_be_ Changed_u)*

In addition, the change of task process makes the workers gain an extra piece of information. This can be modelled by adding a second rate going in to the evolution in tasks to be completed not checked stock. A small piece of knowledge is gained from each and every task changed so the equation for these rates is also highly related to the change tasks rate. I defined the equation as:

*(9) Change_Tasks_Rate_u*Increase_of_knowledge_per_tasks_changed*

I will get back to the magnitude of knowledge that should added for each task later. The change in evolution caused by completion of tasks is similar. The knowledge carried by task not completed is zero at the start of the phase, and stays that way. The equation could be expressed as Completion_Rate_u*(Evolution_in_Tasks_not_ Completed/ Tasks_ Not_ Completed_u). But evolution in tasks not completed is always zero, so that part of it can be ignored. That leaves the equation for the increase of knowledge, which is similarly formulated as the one for change of tasks:

*(10) Complete_Tasks_Rate_u*Increase_of_knowledge_per_tasks_completed*

The question is then how much knowledge should be gained for each task done. I have mentioned earlier how different projects evolve differently. Let us take a look at

the two extremes, fast and slow evolving processes. We should think of fast evolution as being processes where the workers acquire relatively large amounts of knowledge doing the tasks in the early stages of the project, and then progressively smaller amounts towards the end of the phase. In practice this would mean that the phase quickly gets a rough cut of what the final result should be like, and then spend a lot of time doing the final polishing. To model this, I decided to use the amount of possible knowledge that is left to gain, as basis for the increase of knowledge. Of the potential knowledge I always add a constant percentage. Potential evolution gain will be 100% at the start of the project and 0% at the end, so adding a constant percentage makes for large increase in the early stages and practically zero in the end. The equation for the increase was defined as:

(11) IF(Evolution_in_released_tasks>999,0,1000-Total_knowledge_gained)

The increase of knowledge for slow evolving phases was defined in a similar fashion as the one for fast, but instead of basing the increase on potential knowledge that can be gained; I based it on the knowledge already gained. That will be small in the beginning, and large at the end of the project. Obviously, this will rise to infinity if no cut-off is applied. I therefore added a condition check that makes sure evolution does not increase beyond the maximum value 1000.

(12) IF(Evolution_in_released_tasks>999,0,Total_knowledge_gained)

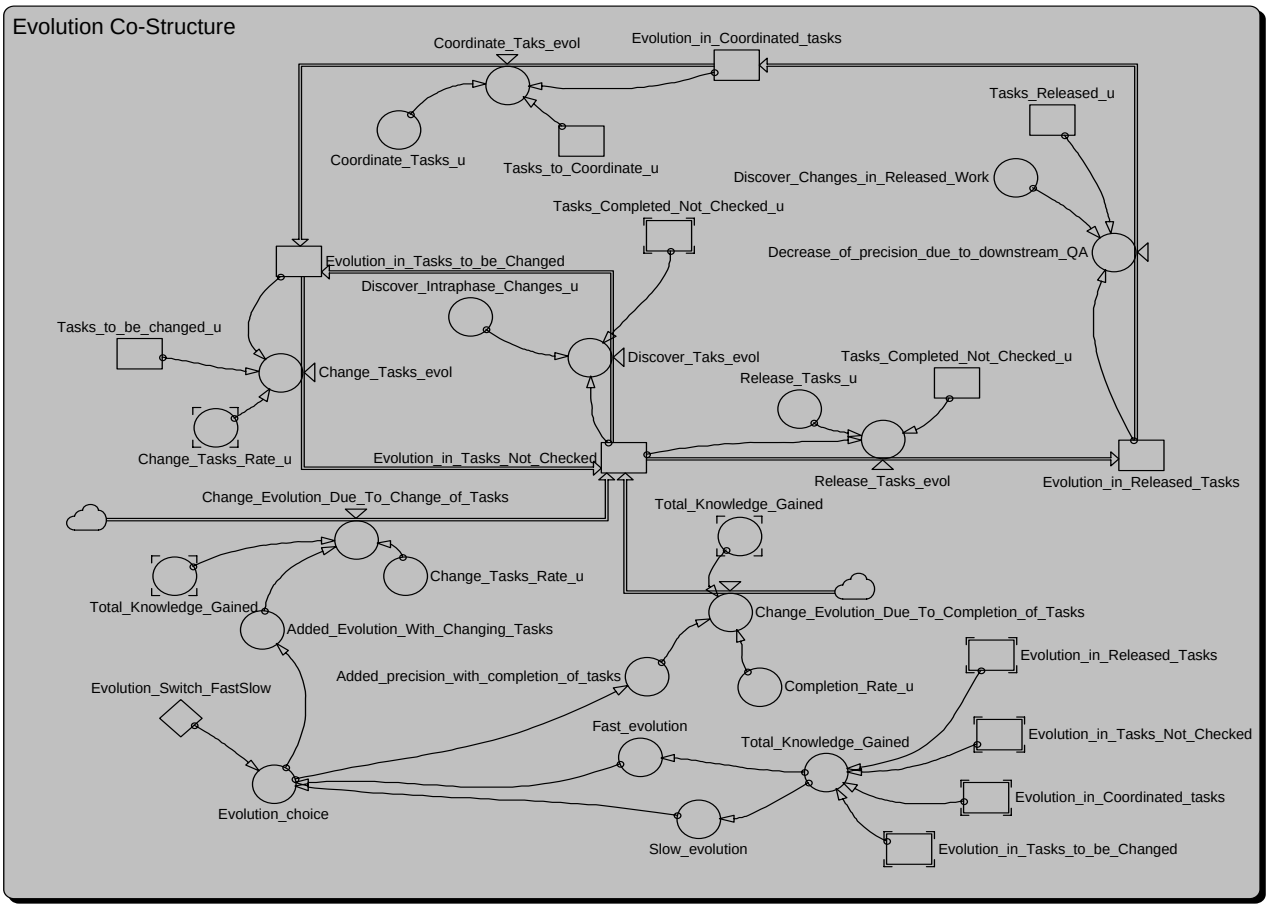


Figure 5: Evolution Co-Structure

4.2.1 Validation of evolution structure

As it is hard to estimate the percentage that should be added for each task done, I had to do some simulations. The goal was to find some settings that made the evolution plots in my model resemble the ones shown in the evolution figure in chapter 2 (Figure 1). Applying the trial and error method, I fine-tuned these values. For the fast evolution I am too a large degree free to choose values, but for slow evolution I had to experiment to make sure the fast rise was timed right so that it peaked on maximal value at the end of the project. If I set the percent increase of knowledge too low or too high, evolution will never reach 100% or peak at 100% too early, respectively. I came to the conclusion that satisfactory results were reached when the setting was set to 0,4% for the increase of evolution due to change of tasks, and 0,35 % for the increase in evolution due to completion of tasks. For the completion of tasks I also added a small constant, 0.03. The reason for this is that the slow evolution is based on the knowledge already gained, which will be zero at the start of the phase. The constant makes sure knowledge is gained even though knowledge already gained is zero, or else increase in evolution would never happen.

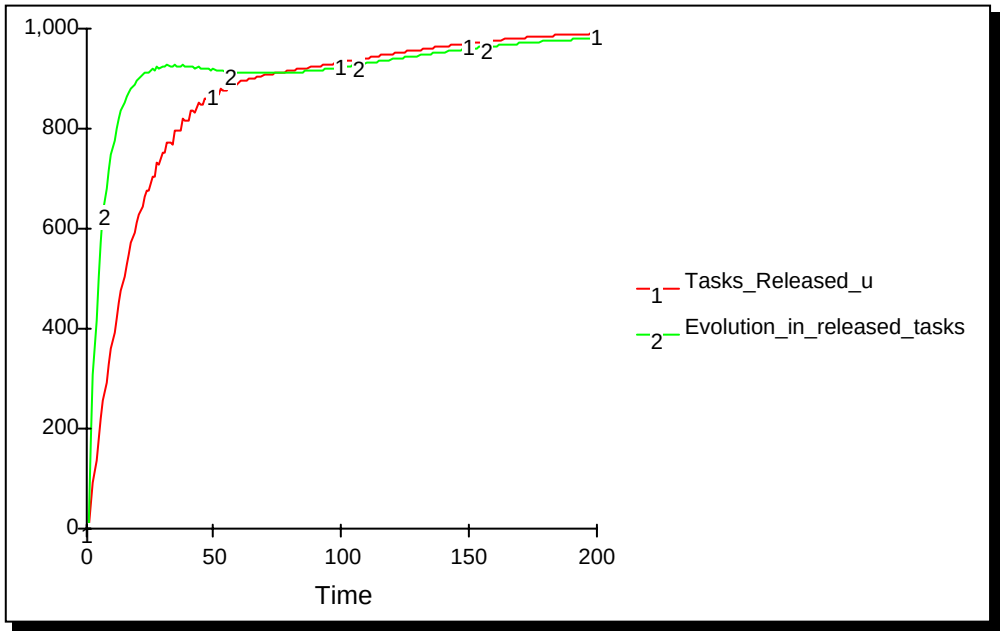


Figure 6: Fast Evolution

The graph shows the evolution in tasks released in the upstream phase ("Evolution_in_released_tasks") as a function of time, when the choice of evolution setting was set to *fast*. The increase of tasks released ("Tasks_Released_u") is shown as comparison. As we can see the evolution increases quickly in the early stages of the phase, and then slows down towards the end.

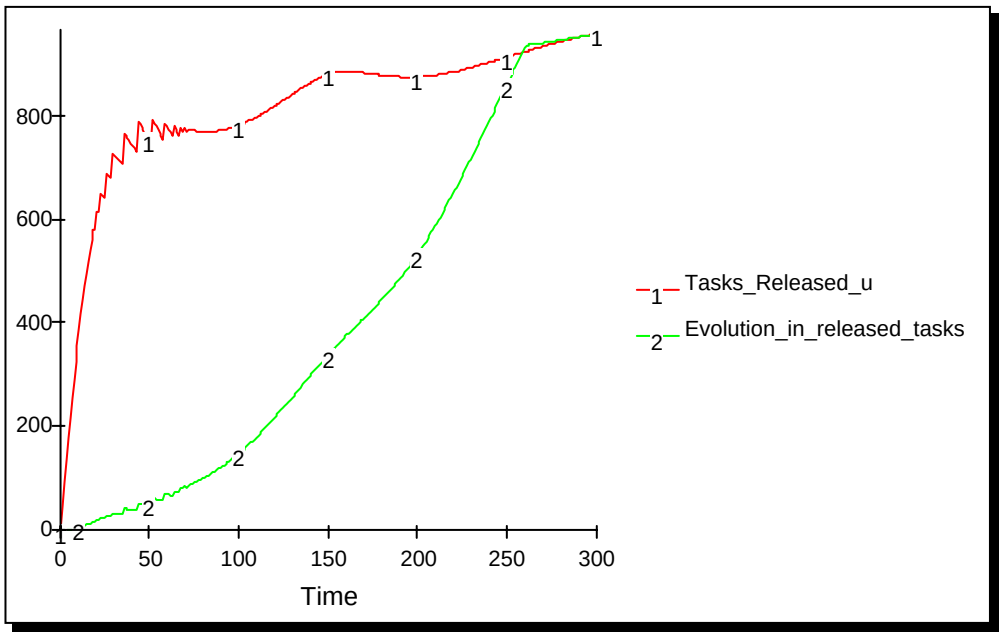


Figure 7: Slow Evolution

Figure 7 shows slow evolution as a function of time. As we can see the actual evolution is much lower than the number of tasks released for the most of the duration of the upstream phase, then starts to catch up towards the end. This is consistent with my description of the slow evolution above.

4.3 Sensitivity

The way sensitivity is defined, by both Krishnan et al. (1997) and Terwiesch et al. (1998), is that should be the duration of the rework as a function of the size of change in between the new information and the previously released information. In other words small changes create little rework, while large changes create large iterations. In Ford and Sterman's model there is a constant called "Avg_Change_ Duration_s" that decides the amount of time that it takes to complete one change in the downstream phase. It therefore occurred to me that I could model sensitivity by modifying this constant. Changes in the downstream phase are introduced through the "Discover_External_Changes" activity, which is the rate at which possible improvement in the upstream phase design specification is discovered. After co-ordination with the upstream phase, where it is decided whether the possible improvement should be implemented or not, potential improvements that are chosen to be included are moved to the "Tasks_to_be_Changed"-stocks in the respective phases. I therefore think it is safe to assume that the magnitude of "information change" in my model can be measured with the number of tasks residing in the "Tasks_to_be_ Changed"-stock in the downstream phase. In Krishnan et al. (1997) on page 442, the authors present a figure of their definition of low and high sensitivity.

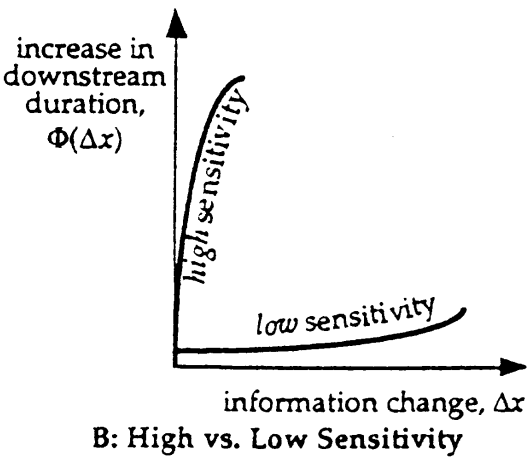


Figure 8: High and Low sensitivity

I have decided to use this figure as inspiration when modelling sensitivity. The graphs have two dimensions, information change on the x-axis and increase in downstream iteration duration on the y-axis. I have established the 'information change' as the number of tasks waiting to be changed in the downstream phase ("Tasks_to_be_Changed_d") and downstream iteration duration as the "Avg_Change _Duration_d"-constant. The solution for modelling sensitivity therefore would seem to be changing the latter constant into a function with "Tasks_to_be_Changed_d" as the input. Because Terwiesch et al. (1997) do not provide a specific definition of what 'small' and 'substantial' changes is, in terms of number of changes, I had to do some interpretation when defining the functions for high and low sensitivity. I decided to focus on the essence of high and low sensitivity and since I am dealing with extreme cases in the first place, the exact definition is not that important. Below you will see the functions I defined for high and low sensitivity:

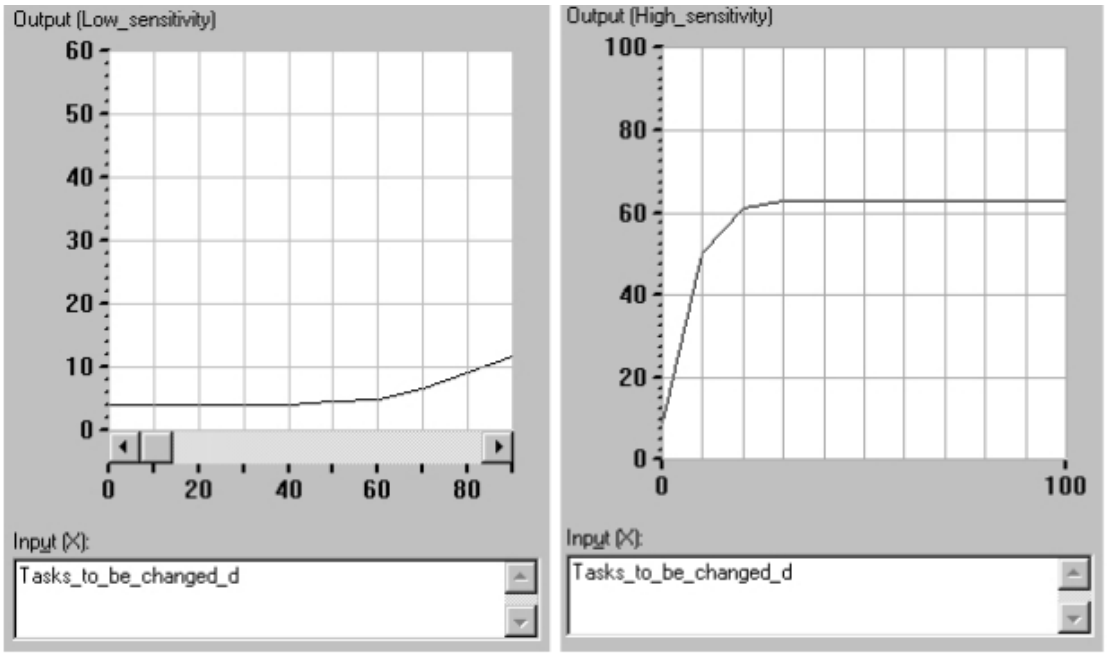


Figure 9: Low and High Sensitivity (Change Duration vs. Number of Tasks To Be Changed)

4.3.1 Validation of sensitivity structure

After applying the modification to the change duration factor, I needed to verify that the model behaved in accordance with the description of sensitivity above. I therefore did some basic tests. Obviously, when working correctly, the simulation should finish earlier when sensitivity is set to low than when set to high, provided all other conditions are the same. I set the "Release_Density_Criteria_u" to a quite high value

(0.7) and the probability of change in the upstream phase equally high, to make sure that substantial amount of changes were passed to the downstream phase. The "Release_Density_Criteria_d" was set low (0,03) to make sure that the downstream phase would make changes, in other words, not release tasks before making possible improvements. The table below shows the project duration, for high low and neutral sensitivity (Neutral is a constant iteration duration of.20).

Sensitivity setting	*Low*	*Neutral*	*High*
Project Duration	437	466	523

Table 2: Project Duration as function of sensitivity setting

The table shows that the model behaves as predicted. When the sensitivity is set low, the project duration is significantly lower than for high sensitivity. The neutral setting lands in the middle, which also makes sense.

4.4 Design Changes

In the previous chapter I explained my decision to focus on design changes rather than mistakes like Ford and Sterman's do in their model. I also pointed out that the biggest difference lies in the way these design changes are discovered. In the model as it is now, changes are discovered by the quality assurance staff in the downstream phase. As the design changes are made in the upstream phase it is more logical that the upstream phase them selves find the possible design changes in their own work. I have decided to stick with quality assurance discovering changes like in the Ford and Sterman model I started out with. But instead of errors being discovered by downstream quality assurance staff, I think it is more precise to make the upstream quality assurance workers discover their own potential design improvements. Consider the point-based strategy: At its most extreme tasks are released to downstream phases as soon as they are finished. In Ford's model there is no quality assurance performed by the upstream on the tasks that are released, eventual changes are discovered by the downstream phase which report them back to the upstream phase. I have decided to change this so that the upstream quality assurance people constantly review the work, even the tasks that are released to the downstream phase.

If a possible design improvement is discovered in the work that is already released to the downstream phase, the changes will either have to be accommodated in both phases, or not implemented at all. This decision will have to be taken by the project managers. If an optimal result is wanted, more or less all design improvements will have to be implemented. On the other hand, ignoring a possible change will lead to loss of quality, may be lead to reduced development times. What decision is most suitable depends on numerous factors. In some cases the design change is essential if the final product is going to be successful. Consider an error were made in the initial design, for example that two electronic components that interferes with each other were placed too close to each other causing the design to not work at all in practice. Other cases where the project managers are forced to choose redesign may be if competing development teams in the market launch a better design, or if increased customer requirements make the old design unsellable. The option to ignore possible design improvements would be most appropriate when the change is not essential for the product to be successful and the downstream phase is very sensitive to changes in the upstream phase, or when finishing early is more important than achieving maximal quality. I will discuss these trade-offs and how they affect the behaviour of a project in the next chapter: *Studies*. For now I will concentrate on implementing what I have discussed in the model. First of all I will need to add a structure to the model so that the upstream quality assurance workers also check the tasks released to the downstream phase for possible design changes. The figure above shows how I decided to model upstream quality assurance on tasks released.

(13) *IF(Tasks_Released_u=0,0,QA*Probability_Discover_Change_u*(Changes_ Released_u/Tasks_Released_u))*

"QA" is the quality assurance rate, in other words the number of tasks the upstream quality assurance workers can check every time unit. The quality assurance people is already checking the tasks completed not checked, but because the rate of quality assurance only is limited by the average duration of checking one task, no modification is needed on the "Quality_Assurance_d"-equation. "Probability_ Discover_Change u" is the measure of how skilled the quality assurance workers are at finding possible design improvements. The last parenthesis is the density of changes in the tasks released to the downstream phase. As we can see changes discovered in the released work in the upstream phase is moved to the "Tasks_to_be_

Coordinated_u"-stock. As I have said these changes have to be implemented in both phases or not at all. This requires co-ordination, i.e. both phases are made aware of what design changes should be accommodated and which should not. For now I will assume that the co-ordination equation for the upstream phase does not need to be changed. I will instead turn my attention to what changes need to be done to the downstream structure to reflect the shift from modelling errors to design changes.

The number of changes in the downstream phase is decided by the "Generating_ Change _During_Completion_Activity_de". This is defined as the rate at which the upstream phase completes tasks, multiplied by the density of changes in the tasks released. This makes sense in conjunction with exchange of preliminary information. The downstream phase will at anyone time use the last information passed from the upstream phase as basis when completing tasks. Therefore, if the information passed to the downstream phase at for example time t_1 contains 30 % potential design changes, then the work completed in the downstream phase at time t_1 necessarily have to the same 30 % changes. So this part did not need to be changed either. Changes introduced to the downstream phase goes to the "Undiscovered_ Changes_de"-stock in the downstream phase. As I have mentioned, design changes have to be discovered by the upstream phase workers. Because the degree of overlap between phases may differ from project to project, the equation for the discover design changes rate in the downstream can not simply be defined as the rate design changes are discovered in the upstream phase. I therefore ended up with this equation:

*(14) Quality_Assurance_ud*Probability_Discover_Change_u*(Undiscovered_ Changes_de/Tasks_Complete_Not_Checked_d))*

Where Quality_Assurance_ud is defined as:

(15) Tasks_Complete_Not_Checked_d/Avg_Quality_Assurance_Duration_u

To the reflect the fact that it is the upstream phase that discovers design changes the probability of discovering change and average quality assurance is the same as for the upstream phase.

As I have mentioned, project managers can choose to implement possible design changes or not. Ignoring potential improvements can be simulated in my model by manipulating the "Release_Density_Criteria_d" in the downstream phase. Setting this to a high value will cause a certain amount of changes to be released before they are implemented. Problem is that changes should not be corrected in one phase and not the other. In this case, if nothing is changed in the upstream phase to reflect the concept of the downstream phase ignoring changes, the upstream phase will go ahead and implement changes ignored by the downstream phase. The percentage of tasks that managers decide should be implemented in the downstream phase is given by the number of tasks in the co-ordination-stock divided by the number of tasks residing in the changes released and co-ordination-stocks. I called this percentage the "Fraction_of_changes_ decided_to_implement". Below you will find the equation:

(16) Changes_to_be_Coordinated_de/(Changes_Released_de+Changes_to_be_ Coordinated_de

The same fraction of the changes should be implemented in the upstream phase, giving this new version of the "Coordinate_Tasks_u" rate:

(17) Coordinate_Tasks_u(Changes_to_be_Coordinated_ui/Tasks_to_ Coordinate_u)*Fraction_of_changes_decided_to_implement*

4.4.1 Validation of design change structure

To validate the modifications to the model, I did some test under extreme conditions to check that the model behaves like it should. The modifications will have little impact on the model if small amounts of changes are released to the downstream phase. Therefore I only did two tests, one where the changes released to the downstream phase are all implemented in both the phases, and one where a part of the changes are ignored, i.e. early finalisation. I used the basic settings found in the appendix, in addition to setting the "Release_Density_Criteria_u" quite high to make sure a substantial amount of changes were released to the downstream phase.

	Changes_ Released_u	*Corrected_ Changes_u*	*Corrected_ Changes_d*	*Changes_ Released_d*
Normal	0	120	175	0.25
Early	98	22	21	194

Table 3: Effect of finalising early

The table above shows the results for the two tests. 'Normal' depicts the test where all changes released to the downstream phase were implemented (in both phases). We can see that this test lead to very few changes released by both the phases, which is what should happen. All changes are implemented in both phases. One might be surprised that the amount of changes corrected in the two phases is different. This is due to two factors. Firstly, there is a chance that changes that are corrected actually can be improved again, meaning the task will be released to the downstream phase twice. The downstream phase will then correct the change once, then correct the same task for the second time when an additional improvement is discovered in the upstream phase. The second reason for the difference in number of corrected changes in the two phases is the way changes are 'transferred' from the upstream to the downstream phase in my model. The probability of generating possible change in the downstream phase is decided by the density of changes in the upstream phase. The density of changes in the upstream phase is given by the number of tasks released by the upstream phase that contain changes. A simple example to clarify: Let us assume 100 tasks are released by the upstream phase and 10 of these are changes, in other words a change density of 10 %. If the downstream phase completes 200 tasks with that information, those 10 changes will lead to 20 changes in the downstream phase. This behaviour is not unrealistic, so I will ignore the difference in corrected changes.

Turning my attention to the second test ("Early Finalisation"), shows that this also works as it should. In this test the work in the upstream phase was finalised very early. We see that very few of the possible changes was implemented in either of the phases.
(For the same reasons as I have explained above, the number of released changes are different.)

4.5 Probability of change in the upstream phase

I have pointed out that having a constant for probability of change in the upstream phase is not very realistic when modelling design changes. Before I can decide on how this can be modelled I need a clear definition of what I 'probability of change' really means in conjunction with design changes. I think it is right to see this factor as the probability that a way to improve the task will be discovered at a later stage. In other words, if 100 tasks are completed and the probability of design change at that time is 60 %, the workers will at some point come up with one or more design changes that will improve 60 of these tasks. By improvement, I mean a change that in some way increases the quality of the final product. To return to the example I used in the first chapter, where the goal were to make an electronic device, an improvement would for example be if the upstream phase found a way to make the component board smaller. In the previous chapter, I suggested that instead of a constant for the probability of design change I should have a time varying factor, that should be maximal at the start of the project and zero when information is finalised in the upstream phase. It also seems reasonable that this factor is related to the level of evolution in the upstream phase. At the start of a project, the workers usually have little knowledge of the work at hand. They are therefore less likely to do the right choices when it comes to picking the best solutions to the problems that occur. That means that the probability of design change is at its maximum for tasks done at an early stage. As the upstream workers gather knowledge it will decrease until reaching zero when work is finalised. I have chosen to see the decrease as directly proportional with the knowledge gained. That assumption suggests the following equation for the probability of change in the upstream phase ("Probability_IntraPhase_Change_u"):

(18) IF((Total_knowledge_gained/Phase_Scope_u)>1,0,Initial_Probability ((1-(Total_knowledge_gained/Phase_Scope_u))))*

Note that the "Initial_Probability" is a user-defined value, which decides at what level the probability of design change will be at the start of the project. In projects where the workers already are experienced in solving similar problems, it is likely that they will be able to do more or less the right decisions from the start. In such cases the "Initial_Probability" setting should be lower, than for example projects where the workers have little experience. Another factor that affects this constant, is the

difficulty or complexity of the work in the upstream phase. In some cases the work may be fairly straightforward while in other cases the work is so complicated that even experienced workers need to correct their work several times before the optimum result is reached. High probability of design changes could also be a result of constantly changing user demands. Due to the number of factors affecting the choice of this constant it is a hard to give a specific estimate of this value. In my experiments later I will set this variable to different values

4.5.1 Validation of probability of change

To test the validity of the new probability of change factor I decided to do some extreme conditions tests. I have argued above that the probability of change should decrease as the upstream workers gain more and more knowledge of the work at hand. This suggests that we should see notable differences in the decrease of the probability of change for slow and fast evolution. The plots below show the change in this probability as a function of time, for fast and slow evolution respectively. For this test I used all the standard settings for the upstream phase that can be found in the appendix.

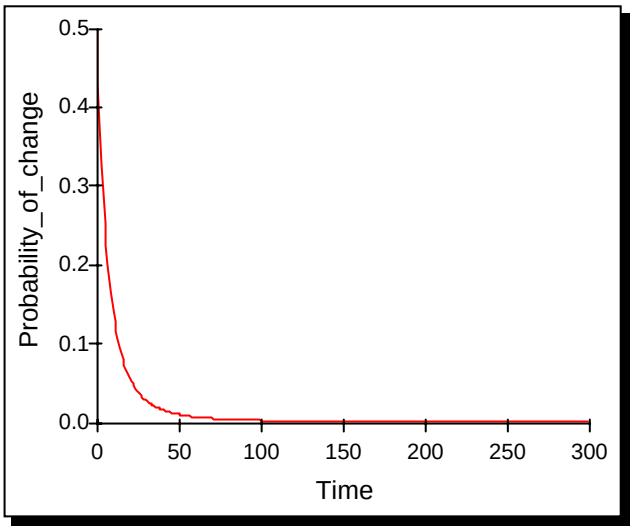


Figure 10: Probability of Change, Fast Evolution

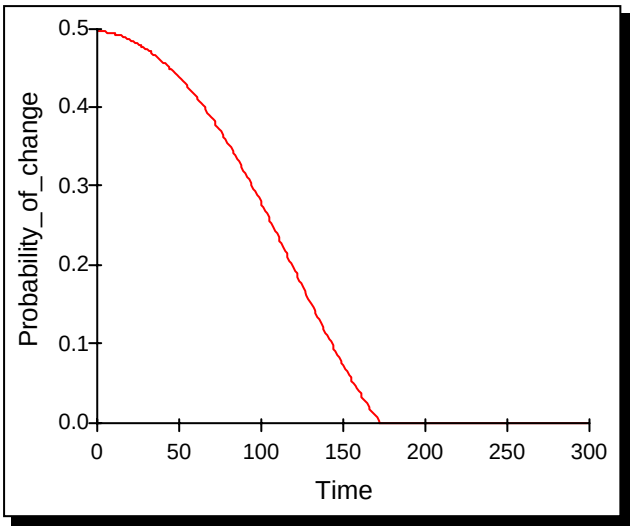


Figure 11: Probability of Change, Slow Evolution

We see that when the evolution is fast, the probability of change quickly decreases levels out and then finally reach zero at around day 100. This is reasonable; as fast evolution would imply that the upstream phase at an early stage get a clear idea about the optimal solution to the work at hand. Therefore the chance that the tasks that are completed in the upstream phase contain a potential improvement, should also quickly decrease. The opposite is the case when evolution is slow, which also makes sense. The probability of change decreases slowly at the early stages of the project, then start to decrease faster and faster.

5 Studies

In the last chapter I argued for how the model should be enhanced to enable me to simulate the theories Terwiesch et al. and Krishnan et al. talk about in their papers. In this chapter I will first of all, validate that my model works as it should. I have decided to do this by simulating certain extreme scenarios, and examine the simulation outputs these produce. Extreme scenarios are easy to predict and are therefore useful when trying to validate the behaviour of the model. I will start by examining how my model can be used to simulate the terms Terwiesch et al. (1998) presents, then move on to Krishnan et al.'s theories. I will do these examinations separately, to make sure they work reasonably one by one. Later I will combine the theories, and see how projects work under those conditions.

5.1 Modelling Strategies

As I have said earlier, Terwiesch et al. have used the terms stability and precision, to create a framework for exchange of preliminary information in concurrent development processes. They talk of two extreme opposites of strategies that can be applied to overlapped processes, namely point-based and set-based strategy. As they define them, the first focuses on a high level of precision in the exchanged information, whereas the latter focuses, more or less, entirely on stability. In my model I am able to control these two parameters, so it should be possible to reproduce the plot I presented for these two strategies in Chapter 2 (figure 2) with my model. A slight problem occurs when trying to plot stability vs. precision explicitly like Terwiesch et al. do. Although I am able to control stability by controlling the number of changes released, I have no explicit measure of stability in terms of percent. Therefore I had to make a function in the model that 'translates' number of released changes to percent stability. In their paper Terwiesch et al. talk of 'low' and 'high' stability which to some degree is pretty vague. I therefore had to interpret this in terms of percent and make it concrete. (The translation function can be found in appendix). Also, although I have a measure of the accuracy actually accomplished in the upstream phase, i.e. evolution, I have no measure of the accuracy of the information passed to the downstream phase. As I have explained earlier, these evolution and precision is not necessarily the same. I have therefore decided to use the external

concurrence relationship as the measure for precision in information exchanged. I will get back to the reason why below.

5.1.1 Simulating point based strategy

To be able to simulate the point-based strategy in my model, there are two variables that need to be set. Those are the “Release_Density_Criteria_u” and the “External_Concurrence_Relationships_d”. I have explained the function of these variables earlier, so I will only do a quick recapitulation here. The “Release_Density_Criteria_u” makes sure that no tasks are released as long as the error density is above this value. For point-based strategy this value should be high. This strategy implies that precise information is passed for the duration of the overlap between the upstream and the downstream phase, regardless of the fact that the information passed may turn out to be wrong at a later stage. In other words, the upstream phase is not withholding any information in fear that it might be wrong, thus the clamp on release should be disabled. As for the “External_Concurrence_Relationships_d”, this variable decides how much work is available to the downstream phase depending on the percentage of work released by the upstream phase. For point-based strategy this should be 100 % for any number of tasks released by the upstream phase, in other words complete overlap. I said above that 100 % precise information is passed downstream for the duration of the upstream phase. That means that the downstream phase will have accurate preliminary information, for example in the form of a product specification or a prototype, even before the upstream phase has started the actual work. These data will enable the downstream phase to start and complete their work. For this reason it is therefore logical that I use the external concurrence relationship as the measure for precision in the information passed. The precision of the information passed decides the amount of work that is available in downstream phases, which the “External_Concurrence_Relationship_d” variable controls in my model.

For the point-based strategy, the only effect the upstream phase has on the work downstream is when information is changed. Changes are communicated in my model via the “Changes_Released_i” variable, which decides the amount of rework in the downstream phase. With these settings I got the following plot in my stability vs. precision chart:

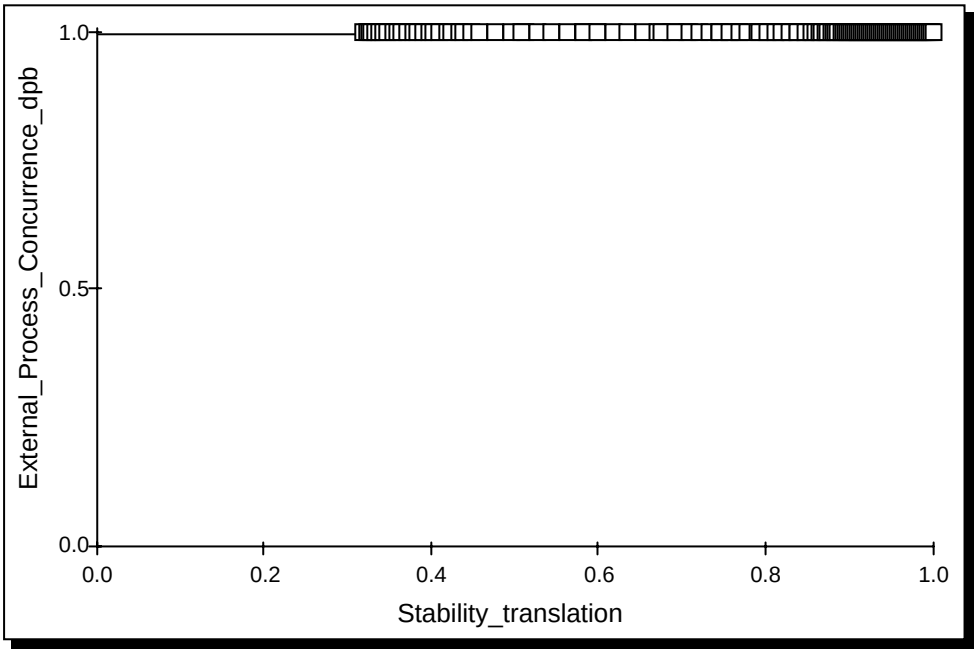


Figure 12: Point-Based strategy

As we can see the precision ("External_Concurrence_relationship_dpb), is at maximum from the start, while stability is gained as the project progresses. The boxes represents time samples, so that means the project start with precision equals 100 % and stability approximately 0.35 %. This graph looks similar to the stability vs. precision plot I presented in Chapter 2.

5.1.2 Simulating set-based strategy

When simulating the set-based strategy I have to initialise the same two variables in order to get a realistic result. I have argued earlier how set-based strategy implies focus on stability, and that stability is highly related to the number of errors released by the upstream phase. In other words, the "Release_Density_Criteria_u" should be quite low, to simulate high stability. This way the upstream get a chance to review their work before it is released, thus making sure fewer errors are passed to the downstream phase. The "External_Concurrence_Relationships_d" also needs to be set differently than for the point-based strategy. Extreme application of the set-based strategy implies only 100% certain information is passed to downstream phases. So the downstream phase is to a much higher degree dependent on the progress in the upstream phase, than for the point-based strategy. I assume that phases are related to

the extent that 10 % precise information released by the upstream phase makes for 10 % work available in the downstream, 20% precise information makes 20% work available and etc. (The same assumption was made for the point-based strategy above but it does not affect that strategy). On these grounds the "External_Concurrence_ Relationships_d" should be set as the figure below shows.

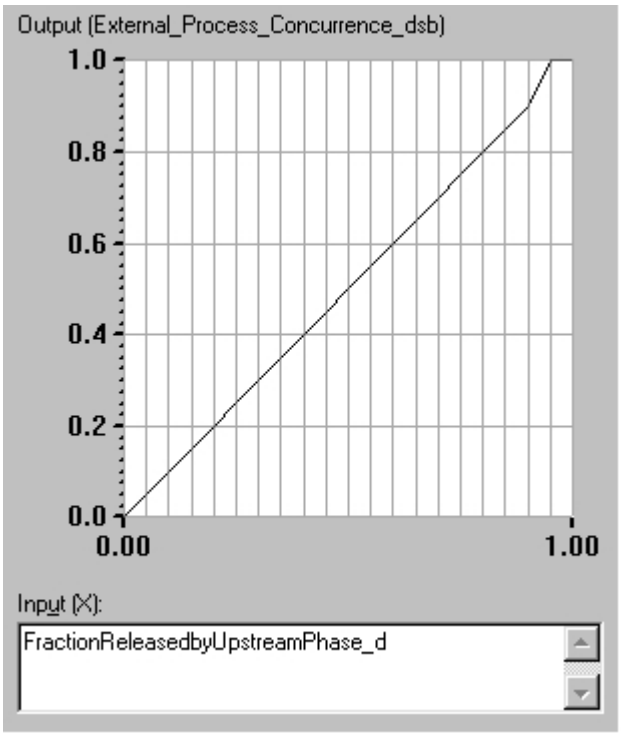


Figure 13: Percentage of Tasks Available (downstream) vs. Evolution In Tasks Released (upstream)

The "FractionReleasedByUpstreamPhase_d" is the evolution in tasks released to the downstream phase (Percentage of evolution based on phase scope). The increase in percentage of work available in the downstream is almost completely proportional to the increase in percentage of released tasks by the upstream phase. We see that the availability of tasks is set to 1 also when the "FractionReleasedByUpstreamPhase_d" is 95 %, i.e. not 0.95 as one would assume. My simulations showed that the evolution never quite reach 1000, and thus the downstream could never be finished if I do not make this modification.

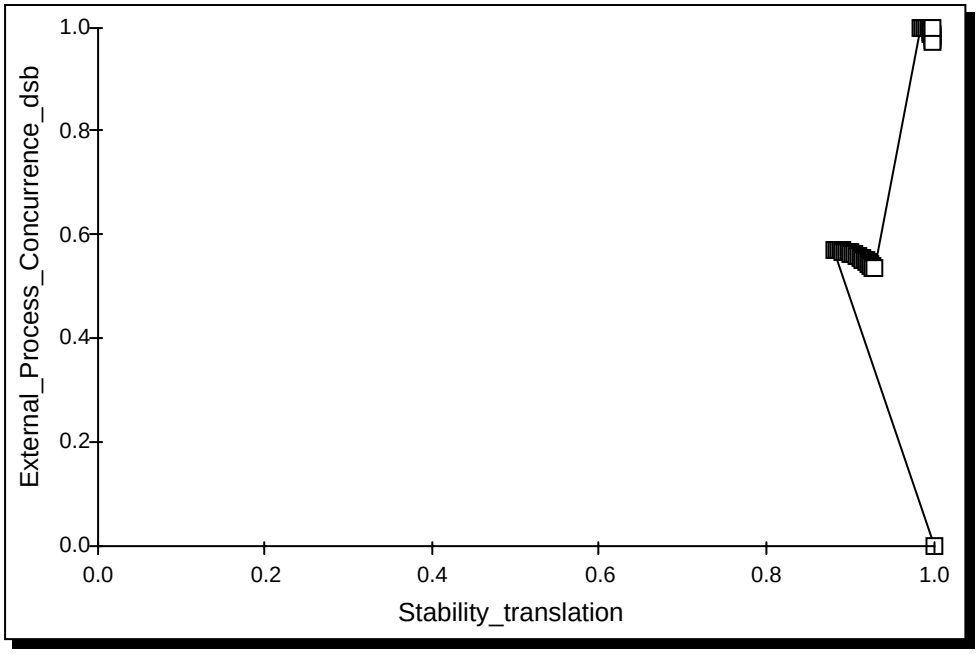


Figure 14: Stability vs. Precision: set-based strategy

The figure above shows the stability vs. precision plot my model produced for the set-based strategy. As we can see the first time samples (square boxes) start at about 85 % in stability and 55 % in precision, i.e. high stability and fairly high precision. The reason that precision is relatively high from the start is that release of tasks is withheld in the upstream phase until the stability is high, giving precision time to increase.

Note that I have not tried to model the concept of developing alternatives that might fit the outcome of the upstream phase at times of starvation in the downstream phase. This would be complicated to model accurately, and I have therefore chosen to ignore this aspect of set-based development. I will therefore assume that the work in the project I'm modelling is sufficiently complex to make duplication in the downstream phase inappropriate. In practice this could for example mean that the number of possible solutions to the upstream phase and the cost of developing alternatives are both too high for such an approach to be effective.

5.2 Simulating Research Questions

Having validated that I can model the two strategies Terwiesch et al. (1998) talk of, I now turn my attention to the theories presented by Krishnan et al. (1998). In their

paper, they introduced evolution and sensitivity. I have described these terms in chapter 2. Furthermore, with evolution and sensitivity as criteria, they have set up a guideline for how overlapped processes should be handled. I intend model the same conditions and see if Krishnan et al.'s conclusions can be grounded in the results of my simulations. I have shown in the previous chapter that I can simulate fast and evolving projects as well as low and high sensitivity. In other words I'm able to reproduce the conditions shown in the figure on page 448 in Krishnan et al. (1998).

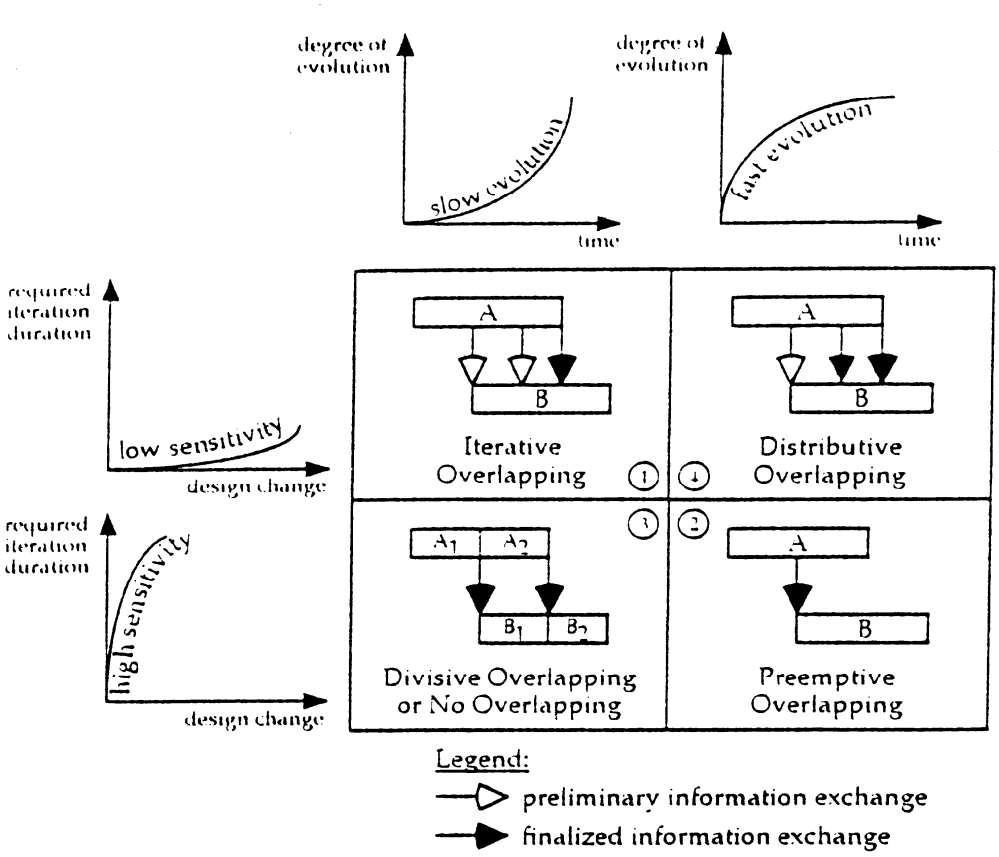


Figure 15: Guideline

As we can see, the chart has slow and fast evolution on the x-axis, and low and high sensitivity on the y-axis. That makes four different combinations. These four combinations should be seen as the extreme cases. Krishnan et al. has examined these cases and suggested how they should be handled by project managers to reach an optimum result based on the criteria lead-time and quality. I will model the same four extreme cases and see if my model suggests the same actions.

5.2.1 Case 1: Low sensitivity, low evolution

The first case I will look into is a project where the upstream phase is slow evolving and the downstream phase is not very sensitive to changes in the upstream phase. Slow evolution implies, as I have explained earlier, that major changes in the design can and will happen towards the very end of the phase. In such projects one would think that releasing very accurate preliminary information, would make for great uncertainty for the most part of the duration of the phase. It is only when the upstream workers begin to get a clear idea about the final result at the end of the phase, that highly precise and highly stable preliminary information can be passed to downstream phases. So applying the point-based strategy to such a project will probably lead to large amounts of iteration. On the other hand, using the set-based strategy can also prove difficult. I have argued that this strategy implies highly stable information being passed on for the duration of the overlap between the up- and downstream phase. In a project where the evolution is slow, it arguably takes quite some time before certain information can be passed to the downstream phase. While the upstream phase works and reworks tasks to find the optimum solution, work will stand still due to the lack of information, i.e. starvation in the downstream phase. In slow evolving projects there seem to be a choice between huge amounts of rework or starvation in the downstream phase. What choice is best will be decided by the sensitivity factor, which for this case is low. That implies that large changes in the upstream phase can be relatively quickly be accommodated in the downstream phase. So even though the slow evolution in the upstream phase undoubtedly will lead to a lot of changes in the downstream phase, the low sensitivity will probably make sure that these changes will not delay the completion of the downstream phase abundantly. In other words one would think that the point-based strategy would be most suited under these circumstances. This is also what Krishnan et al. conclude in their paper (Terwiesch et al., 1998).

To simulate the conditions of case 1, I set the “Change_Duration_d” factor to the pre-set for low sensitivity. Furthermore, I set the evolution switch for the upstream phase to 0, meaning slow evolution. The “Probability_of_Intraphase _Change_u” variable was initialised to 0.5, in other words there is a 50% chance that the first tasks completed can be improved by a design change that will be discovered later on (This number will of course decrease as the knowledge gained, i.e. evolution increases).

The choice of a relatively high setting for probability of change factor was made to make the differences in the way the two strategies affect the result of the project more evident. All the other settings that can be found in the 'Basic settings' appendix were not changed.

To decide the degree of success of the project, I decided to watch lead time ("Duration") and the amount of rework done in the downstream phase. With the basic settings for the "Release_Density_Criteria" constants in the two phases, there will be no quality loss in either of the phases. For the set-based strategy the default value is 0,001 for this constant, meaning almost no tasks are released in the downstream phase, before potential changes inherited from the upstream phase are discovered and implemented.

	Project_ Duration	*Changes_ Released_u*	*Corrected_ Changes_u*	*Corrected_ Changes_d*	*Changes_ Released_d*
Point-Based	359	0	366	472	0.25
Set-Based	451	0	8.4	463	1.04

Table 4: Simulation Outputs for Case 1

As we can see from the table, the lead time for the point-based strategy is noticeably smaller than for set-based strategy. So the first is obviously more time effective. With lead time as criterion, I can therefore conclude the same as Krishnan et al. (1998). We also see that the applying the point-based strategy involved a lot more rework in the downstream phase than for the set-based strategy. The reason that the project still was completed at a much earlier time is undoubtedly that the low sensitivity lead to changes being quite quickly accommodated in the downstream phase. I have earlier argued one of the decisions that have to be made by project managers is when to provide the downstream with preliminary information. The way I have chosen to model point-based strategy in my model, precise information is passed to the downstream phase from the start, i.e. the downstream phase will be able to start their work immediately. This can be observed on the plot below.

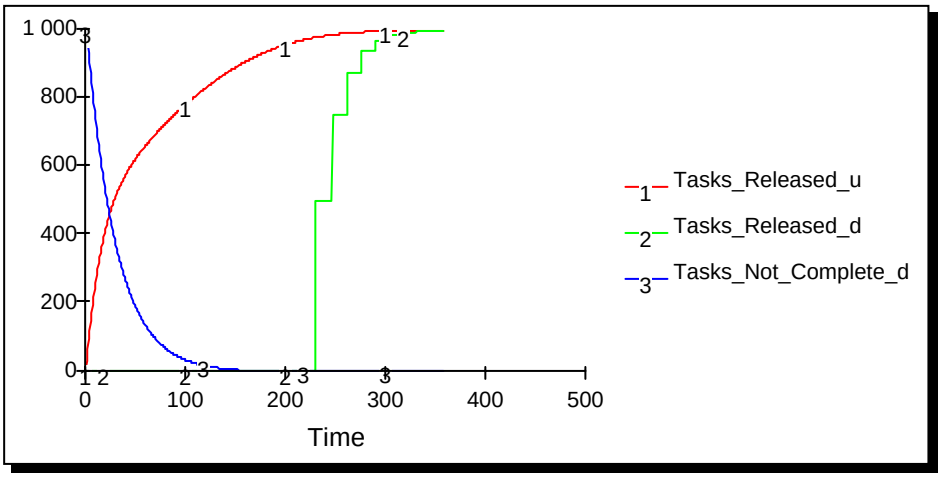


Figure 16: Case 1: Point-based strategy

The "Tasks_not_Complete_d" starts to decrease from day 0, and the downstream phase has in fact completed all the work by day 150. After a period of iterations in the downstream phase, all changes are implemented, and so the tasks can be released.

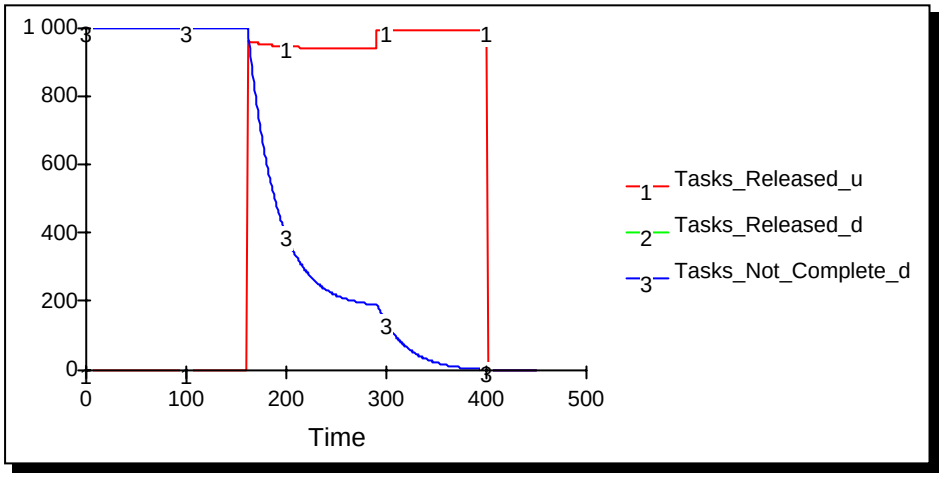


Figure 17: Case 1: Set-based strategy

The plot above shows the model behaviour when the set-based strategy was applied. The plot shows that no tasks are released from the upstream phase until day 150. Comparing the two plots tells us that the application of the point based strategy lead to the downstream getting a 150 day head start on the project where the a set-based approach was facilitated. Due to the low sensitivity the downstream project only lost 50 days of this head start because of iterations (Final completion dates were 359 and 451 for the point-based and set-based project respectively).

We saw that the point-based strategy was most appropriate when maximal quality is wanted. The graphs for this case, showed that when the set-based approach was facilitated for this project, the degree of overlap between the two phases was very limited. The “Release_Density_Criteria_u” was set very strict, which means that the downstream phase was not allowed to release information unless they were almost 100 % sure all possible improvements in the upstream work was discovered and implemented. In a project where the evolution is slow, this will happen at a quite late time, meaning almost no information was passed to the downstream phase before the upstream work was almost finished. In other words my model shows that when evolution is slow, a strict set-based policy will lead to more or less sequential development.

I have mentioned the concept of finalisation, and how early finalisation can lead to development time being saved. Unfortunately, finalising results early in the upstream phase in a process where the final result is very much in the blue until the very end is not advisable. Every time some part of the work is finalised, project managers close the door for a set of possible outcomes of the process. At the same time they risk making a decision they might regret later, when they know more about the work at hand. In slow evolving projects one would assume that this risk was extra high. I will test this in my model.

Let us assume a project where quality is not an requirement to quite the extent as in the above tests, i.e. that finishing the project early is more important than making a product that is optimum quality-wise. Let us imagine that the project managers are content as long as the product works as it should and that the overall quality is fair. In my model there is no differentiation on the nature of possible design improvements, but let us imagine that 10 % of the changes that can be observed in my simulation are not essential for the final product to succeed in the market. In practice that will mean finalisation of parts of the work as soon as a certain level of quality is reached within that part. In my model this can be simulated by setting the “Release_Density_ Criteria_u” slightly higher, so that the known improvements density criteria for release is not quite as strict (I choose the setting 0.05 which lead to around 100 changes being released downstream). One would think that this would lead to more information being released to the downstream phase at an earlier time. I said, that the upstream phase would be content when a certain degree of quality is reached, that

means that they would have no intention of implementing the possible design improvements released to the downstream phase. So the downstream should not either. In other words the information passed can be considered as 100 % certain. In my model this can be simulated by setting the "Release_density_criteria_d" in the downstream phase to 1, in other words tasks are released immediately regardless of numbers of changes. Also I needed to turn of quality assurance of tasks released in the upstream phase. Obviously upstream phase will not perform any quality assurance on tasks that they are content with. With these settings I got the following results from the simulation:

Release_Density_ Criterium_d	*Project_ Duration*	*Changes_ Released_u*	*Changes_ Released_d*
0.05	382	42	43

Table 5: Case 1: Early finalisation

Even though we set the quality requirement lower the project still finishes later when applying the set-based strategy instead of the point-based strategy. The reason for this is obviously that the slow evolution makes finalisation infeasible until late in the project, if large quality loss is to be avoided.

5.2.2 Case 2: High sensitivity, low evolution

In the second case, evolution is still slow, but this time sensitivity is high. In other words, even small changes in the upstream phase will require very time consuming iterations in the downstream phase. Therefore it seems logical that to save large amounts of rework in the downstream phase, as little changes as possible should be released to downstream phases. In the previous case we saw that the point-based strategy involved quite large amounts of rework in the downstream phase. It is therefore fair to assume that a shift from point-based to set-based paradigm is the logical thing to do. This is what Krishnan et al. also suggest in their publication (1997). Choosing a set-based strategy will make sure the stability on the released is high, thus ensuring minimum of rework in the downstream phase. As I have pointed out in the discussion of case 1, low evolution implies that stability can not be accomplished before late in the process, when evolution has reached a sufficiently high level, i.e. close to 100 %. Krishnan et al. (1998) suggest that the work should be

split into independent parts. The object of this method is to find out if some of the components evolve faster than others, and thus can be finalised at an earlier time. The other solution that Krishnan et al. (1998) presents is that the phases simply should not be overlapped at all. Krishnan et al. (1998) argues this strategy should be used when no disaggregated parts of the work evolve quicker, or if the information extracted from the part evolving quicker is useless/does not affect the downstream phase. As I do not have disaggregation implemented in my model per se, what will happen when I apply the set-based approach, is probably that there will be little or no overlap between phases. This is also what I observed out in the case 1. Note that the behaviour of the upstream phase is not affected by the sensitivity setting in the downstream phase. The only activity in the downstream phase that affects the upstream is the rate at which it is decided what possible improvements should be implemented. After the phases the have decided what parts of the work should be reworked, the rate the respective phases actually performs the iteration is indifferent for the other part. In other words the downstream should start at the same time in case 2 as for case 1, i.e. practically sequentially.

Even though the concept of disaggreagation of work into smaller pieces is not featured in my model, this could for be modelled by creating two or more copies of the upstream phase model structure. Assume the phase scope of 1000 tasks is divided into two smaller, independent pieces of 700 and 300 tasks. In that case, we would have two copies of the upstream phase structure, one initialised with 300 tasks as phase scope and the other with 700 tasks. Both structures will feed the downstream phase, i.e. the sum of changes and tasks released by these structures put together will decide the amount of rework and work available, respectively, in that phase. The settings for the two upstream structures would have to be the same, except for some settings deciding the characteristics of the work, which could be set independently. This could for example include rate of evolution, probability of design change and so fort. It is likely that if some parts of the work evolve quicker than others, that lead time will be saved. I have not followed this any further though.

To simulate case 2 in my model I kept the same settings as in case 1, except this time I set the sensitivity to the high setting for the duration of the overlap. The criteria of success are also the same as above. The table shows the simulation outputs for high sensitivity/low evolution.

	Project_ Duration	*Corrected_ Changes_d*	*Changes_ Released_u*	*Changes_ Released_d*	*Corrected_ Changes_u*
Point-Based	486	366	0	0	472
Set-Based	453	8.5	0	0.9	465

Table 6: Simulation Outputs for Case 2

The results show that the set-based strategy is best suited to obtain minimum lead time under this given set of circumstances. The high sensitivity has lead to massive amounts of rework in the downstream phase, to the point where nothing is gained by starting the work downstream early by using preliminary information. As we can see the set-based approach lead to little difference in lead time for case 1 and 2. This is reasonable, because few changes are released in both cases when applying this strategy, so the sensitivity should have little effect.

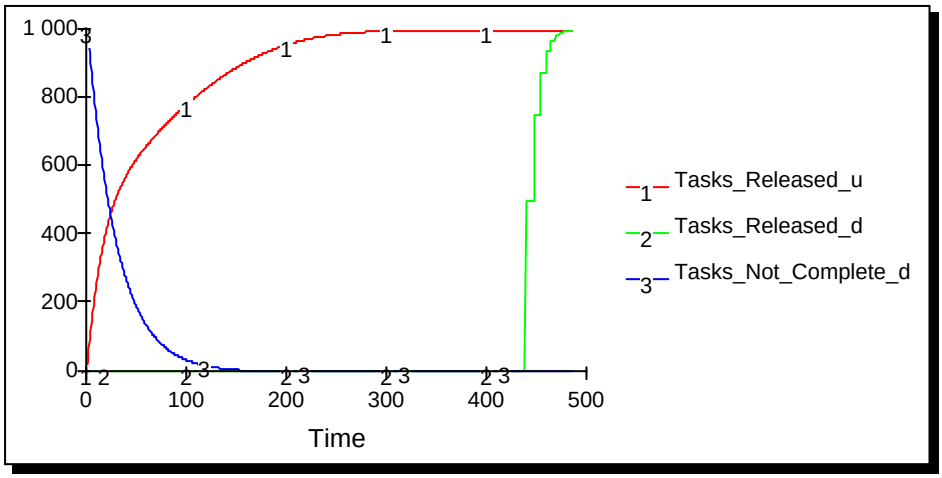


Figure 18: Case 2: Point-based strategy

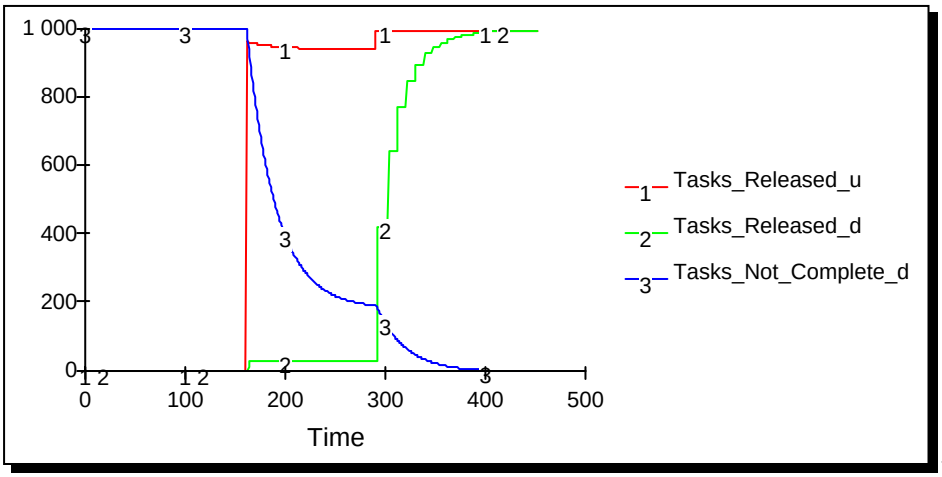


Figure 12: Figure 19: Case 2: Set-based strategy

Comparing these plots with the similar plots for case 1 strengthen my conclusion. The set-based plots are basically identical for the two cases. The point-based strategy plots on the other hand show that the sensitivity has lead to a large period of iterations in the downstream phase. The head start given by facilitating the point-based approach has in other words not compensated for time required for rework in the downstream phase. I can therefore conclude that the set-based strategy is most appropriate when the evolution is slow and the sensitivity is high.

Once again I trid to finalise information in the upstream phase, at varying times to see how that affected project outcome.

	Project_ Duration	*Corrected_ Changes_d*	*Changes_ Released_u*	*Changes_ Released_d*	*Corrected_ Changes_u*
80	464	315	166	52	295
100	471	331	129	35	332

Table 7: Case 2: Early finalisation

We see that the finalsing early, lead to great quality loss, compared to the number of corrected changes shows a deviation of our 150 changes. We see that not much time is saved when it comes to project duration. In other words the few days that are saved by finalising earlier comes at a great cost in regard to quality.

5.2.3 Case 3: Low sensitivity, fast evolution

The third extreme case is when sensitivity is low and evolution is fast. As I have explained earlier fast evolution implies that the upstream phase at a quite early time gain enough knowledge of the work at hand, to be able to stipulate relatively accurate the final result. In other words, highly stable and precise information can be communicated to the downstream phase at a quite early time. So one would think that passing preliminary information from the very start of the project could be done without risk of massive amounts of rework in the downstream phase. Therefore a point-based strategy would seem to be appropriate. The fact that the downstream is not very sensitive to changes made in the upstream phase strengthens the choice of this option.

A point-based strategy is also more or less what Krishnan et al. suggests in their conclusion. But they add an another dimension by saying that the information also should be finalised at a quite early time in the upstream phase. The fast evolution in the upstream phase implies that the exchanged information can be finalised at a quite early time, without risking much quality loss. In other words the downstream phase will have information that will not change, long before all the tasks are actually completed in the upstream phase. Giving the downstream phase finalised information early has many advantages. The downstream phase will in most in cases have some deadline that has to be met. Therefore the sooner they have finalised information to work on; the more time they will have to pursue different solutions to their own work. Ultimately that will lead to higher quality in the downstream phase, because they will have more time to experiment, instead of constantly readjusting to new changes in the upstream phase. In other words, finalising early in the upstream phase when evolution is high will lead to a situation where some quality is sacrificed in the upstream phase, so that the downstream phase will have more time to improve quality in their work. Krishnan et al. (1997) therefore calls this *distributive* overlapping, because the impact of overlapping is divided more equally between the two phases.

I have done the same tests for this case as for the two first; first I applied the set-based and point-based strategies and then examined how these affect the project differently. The sensitivity setting is set to low sensitivity with the “Sensitivity_Switch”, whereas the evolution was set to the pre-made fast evolution. In light of what I have said above it was also obvious that I should try initially set the strategy to point set, then at some

point when the evolution is sufficiently high, finalise information, and see how that affects the quality of the project. To check the quality it is natural to look at the number of changes released by the respective phases.

	Project_ Duration	*Changes_ Released_u*	*Changes_ Released_d*
Point-Based	333	0	0.25
Set-Based	329	0	1.3

Table 8: Simulation Outputs for Case 3

As we can see the results for the two strategies was basically the same. Looking at the graph for the tasks released in the respective phases for the set-based and pint-based strategies, shows why this is the case:

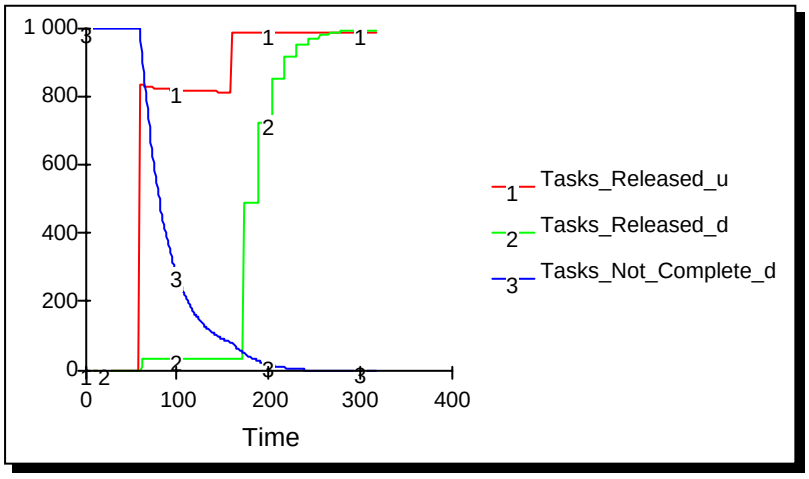


Figure 20: Case 3: Set-based strategy

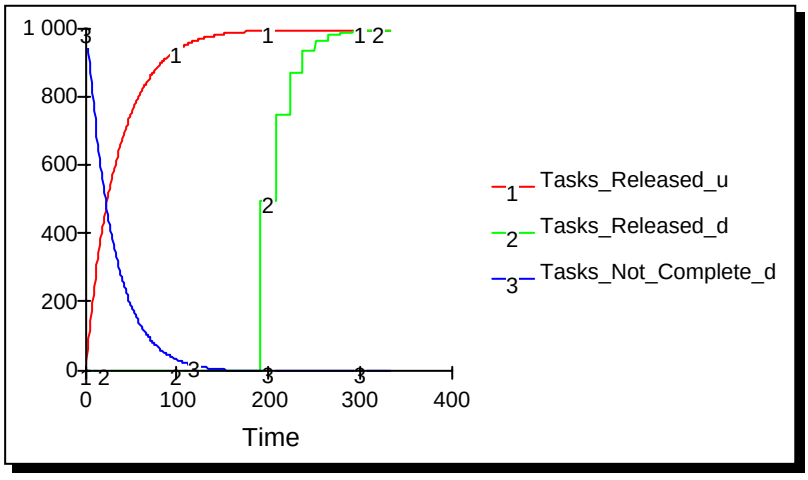


Figure 21: Case 3: Point-based strategy

As we can see the evolution is so fast that already at day 50 of the project the upstream phase has gained enough knowledge to able to release almost 80% of the work. As the settings in my model make sure that the stability is very high before any tasks are released when applying the set-based strategy, that means both stable and quite precise information is passed on very early. So the 50-day head start the point base strategy gave the downstream phase has actually been eaten up by iterations, to the point where nothing is really gained. In this case the sensitivity was low in the downstream phase, so this tendency will certainly be even more notable when the sensitivity is high. I will examine that in the next case.

Last it will be interesting to examine how distributive overlapping, i.e. early finalisation in the upstream phase will affect the behaviour of the project. Early finalisation would imply that the upstream phase at some point feel that they have reached a satisfactory design and so further effort to find possible improvements are stopped. At that point the upstream phase will only finish the remaining tasks according to the frozen specification. To simulate this I have chosen to put in a switch that turns of the quality assurance in the upstream phase. Beyond the point the switch is set, no more quality assurance will therefore be performed, i.e. no new changes are discovered. I feel that this is consistent with the definition I gave above. The strategy was set to point-based, then I experimented with finalise switch to get observe the relationship between time of finalisation and quality loss.

Time of finalisation	*Project_ Duration*	*Changes_ Released_u*	*Changes_ Released_d*
50	259	30	55
60	267	20	39
70	274	16	27
80	280	11	19

Table 9: Early Finalisation in Case 3

As we can see the earlier the finalisation the higher the quality loss and the less lead time. We see that the longer we wait by finalisation the less is the effect. For example the difference in saved lead time is bigger when finalising at day 50 and 60 (12 days) than for day 70 and 80 (6 days). The tendency for reduction of quality loss is the

same. This leads us to believe that the small 'polishing' of the upstream phase design, which is time-consuming but only accomplishes minor improvements, starts about day 60. My simulation shows that a lot of time can be saved by simply skipping this last polishing phase of the work in the upstream phase. The results also show that quality will not suffer greatly if doing so.

5.2.4 Case 4: High sensitivity, fast evolution

The final case I will examine is a project where the evolution is fast, whereas the sensitivity is high. We saw in case 2 how the point-based approach lead to a lot of time spent on rework in the downstream phase, because of the high sensitivity to changes made in the upstream phase. In that case the advised strategy was to pass very high stability information, in order to minimise the amount of rework. The difference between this case and case 2 is the evolution. In case 2 the evolution was slow, which with a strategy involving only stable information being passed, lead to almost no overlap between the upstream and downstream phase. The reason was that upstream phase only towards the very end knew enough to provide the downstream phase with certain data. In this case, where the fast evolution we would assume that the amount of overlap between the phases would be much greater. Because of the high sensitivity one would still assume that the set-based strategy should be favoured in front of the point-based strategy. We saw in case 3 that the head start given by facilitating a point-based approach was in fact entirely eaten up, even though changes was relatively quickly accommodated by the downstream phase. This tendency will certainly not change direction when the sensitivity is high. So the simulation should show that a strategy implying that stable information is passed for the duration of the upstream phase, would be more time cost effective than an iterative approach.

In the previous case I also argued that in projects where evolution is fast, the upstream phase are able to freeze the design at a quite early time without quality suffering too much. Krishnan et al. (1997) suggests that the same approach should be facilitated for this case as well. But instead of providing the downstream phase with advance information like in case 3, Krishnan et al. advice that the downstream phase should not get any information until work is finalised upstream. The reason for this that rework should be avoided in the downstream phase because of the high sensitivity.

I will simulate this approach in my model and see if this conclusion can be grounded in my simulation results as well.

	Project_ Duration	*Corrected_ Changes_d*	*Changes_ Released_u*	*Changes_ Released_d*	*Corrected_ Changes_u*
Point-Based	386	120	175	0	0
Set-Based	333	115	11	0	1,2

Table 10: Simulation Outputs Case 4

As I expected the point-based approach makes the project finish notable later than for the project where the set-based strategy was facilitated. This is obviously due to the relatively high amount of rework in the downstream phase. The project duration for the set-based strategy is almost the same for case 3 and 4, which makes sense. The low amount of downstream rework makes sure the level of sensitivity has little effect.

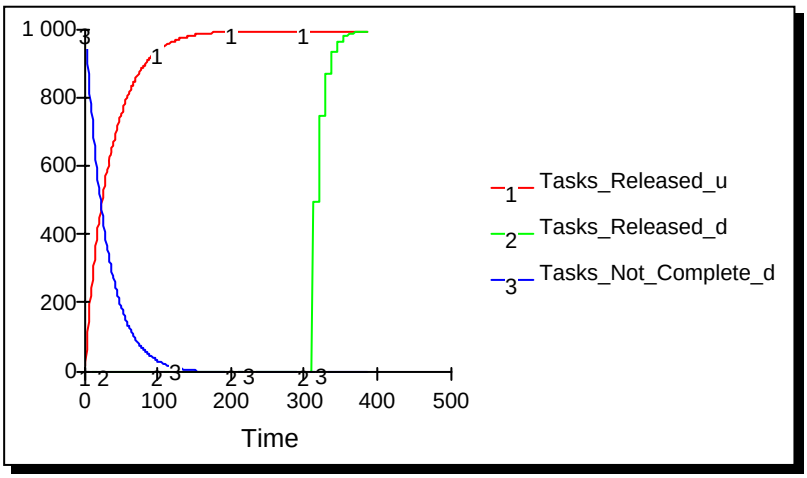


Figure 22: Case 4: Point-based strategy

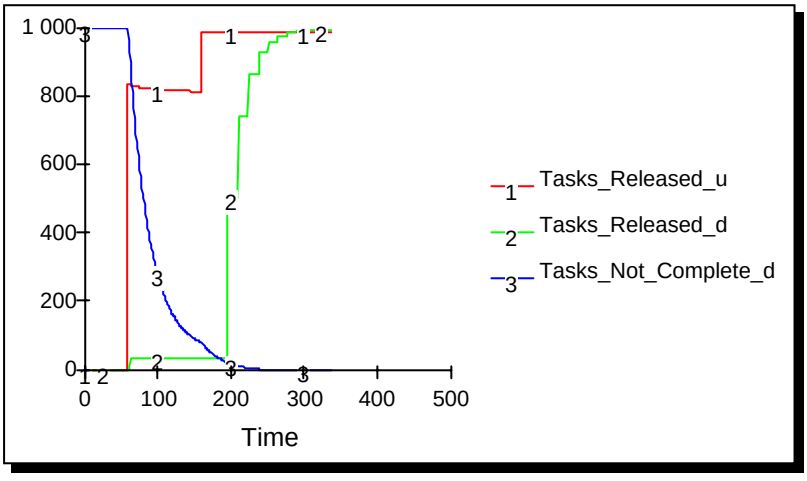


Figure 23: Case 4: Set-based strategy

By comparing the plots for the point-based strategy, for case 3 and 4, shows we see that they are similar, except for the release of tasks in the downstream phase is delayed noticeable in case 4. This is due to undoubtedly due to the differences in rework duration in the downstream phase. The plots for the set-based strategy is basically identical, which is what we could predict, based on the assumption that sensitivity has little effect when applying this approach.

To examine the effect of finalising early I facilitated much of the same approach as when doing the similar test in the previous case, except that this time the strategy was set to set-based. The table below shows my findings.

Time of finalisation	*Project_ Duration*	*Changes_ Released_u*	*Changes_ Released_d*
50	266	22	25
60	266	22	25
70	266	15	18
80	271	10	13
90	272	7	9

Table 11: Early Finalisation in Case 4

This table shows much of the same results as for the similar test for case 3. Finalising can be done, leading to notable reduction in lead time, without much quality loss. The longer we wait with finalising the less quality loss. We see here that the difference in lead time is not tremendous, so the upstream can wait with finalisation until the density of changes is quite low.

6 Conclusion

With Ford and Sterman's model (Ford and Sterman, 1998) as basis I have constructed a model that can be used simulate the overlap of concurrent development processes. My goal was to more accurately be able to describe different projects and the characteristics of the information that is exchanged between the involved phases. I have therefore added structures that allow me to describe the information in terms of its precision and stability, and the projects in terms of the upstream evolution and the downstream sensitivity to change. I have also done some modifications to reflect the fact that I wanted to model design changes in the upstream phase, as opposed to errors like Ford and Sterman do in their model. To verify that the model behaves reasonably realistic, I performed extreme condition tests for each of the structures I have added to the model. Extreme conditions makes for behaviour that is fairly easy to predict, thus allowing me to judge the validity of the model. From the simulation outputs of these tests I could conclude that the model works like it should. It would therefore be appropriate as a learning tool, where project managers can manipulate conditions, and check how different actions affect vital factors as quality, work cost and project duration.

In Chapter 2 I presented three means project managers have to influence the outcome when overlapping development processes. I argued that the decisions project managers make in conjunction with how to apply these means will ultimately decide how effective the overlap will be. In this report I have examined how these means affect the behaviour of projects, and come to some conclusion on how they should be applied to obtain a optimum result. Projects were described in terms of upstream evolution and downstream sensitivity to changes in the information passed from the upstream phase.

In my model two different polar strategies can be applied, namely set and point-based strategy. The set-based strategy implies that the downstream phase is not provided with preliminary information before the upstream is certain of its validity. That means late start of the downstream phase and large intervals between additional updates from the upstream phase. Point based strategy implies the opposite, upstream phase is started right away and intervals between information exchanges are small. My

simulations showed that the set-based strategy was most advisable in projects where the upstream evolution was slow and the downstream sensitivity to changes was high. In that case, major amounts of rework was required in the downstream phase, which due to the high sensitivity lead to vast delay of completion in the downstream phase. I found that by withholding information until the upstream workers were able to pass more certain information, in fact lead to shorter project duration even though the downstream phase was started at a much later time. In the other case with slow evolution, where the sensitivity was low, the fact that rework was quickly accommodated, downplayed the effect of withholding information. In other words, I could conclude that under those conditions the point-based strategy was most appropriate. These conclusions were consistent with the conclusions Krishnan et al. (1998) have made.

Experimenting with finalisation in the two first cases showed that this mean had little effect when the evolution was slow. The effect of freezing the design will decrease the fewer tasks that remains to be completed. In slow evolving projects it is necessary to be able to do design alterations to the very end for optimum quality to be reached. Simulations proved that the finalisation would have to be done very late in the upstream phase, if large quality loss were to be avoided. In fact so late, that there was not much to gain in terms of saved project development time.

The last two cases I examined both dealed with projects where the evolution in the upstream phase was high. For these projects Krishnan et al. (1998) have suggested the mean of finalising early, i.e. before all tasks are completed in the upstream phase, should be facilitated to obtain the optimum result. The fact that the upstream phase at an quite early time have gained enough knowledge of the work at hand, make the risk of quality loss when finalising early minimal. My simulations showed that when sensitivity was low, the combination of a point-based strategy as well as early finalisation made for an optimum result. Even though the point based strategy lead to some rework in the downstream phase at the early stages, this was quite quickly accommodated and thus led to an early finish of the project. Additional project time was saved by finalising around the point where new iterations only produced minor improvements.

For the case where the sensitivity was high, I found that it was not appropriate to start the downstream phase with uncertain preliminary information, as this lead to time consuming rework. Simulation results suggested that the upstream phase should be finalised early, and then the downstream phase should be started with that information. I found that this technique lead to quite early start of the downstream phase, without quality suffering too much in the upstream phase. Because finalised data are 100 % certain information that will not change, no rework was required in the downstream phase, leading to an early completion time. My result for the cases where the evolution was high was also in coherence with Krishnan et al's. suggestion on how these projects should be handled (1998).

7 References

Ford., D. N., J.D. Sterman, 1998 .“Dynamic Modelling of Product Development Processes”, System dynamics Review Vol. 14, No. 1, 31-68

Krishnan, V., S. D. Eppinger, and D.E. Whitney, 1997.“A Model-Based Framework to Overlap Product Development Activities”, Management Science 43, 1997, 437-451

Terwiesch, Ch., Loch, Ch. H., and A. De Meyer, 1998. “A Framework for Exchanging Preliminary Information in Concurrent Development Processes”

Richardson, G.P, A.L. Pugh III, 1981, “Introduction To System Dynamics Modelling”

Ward, A., Liker, J.K., Cristiano, J.J. and D. K. Sobek II, 1995. “The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster”, Sloan Management Review, 1995, 43-61

Appendix

Appendix A: Model Equations

init Changes_Released_de = 0
flow Changes_Released_de = +dt*Release_Changes_de
doc Changes_Released_de = External changes released. The number of changes released by the downstream phase.

init Changes_Released_ui = 0
flow Changes_Released_ui = -dt*Discover_Changes_in_Released_Work
+dt*Release_Changes_ui
doc Changes_Released_ui = Internal changes released. The number of changes that at anyone time is released to the downstream phase.

init Changes_to_be_Coordinated_de = 0
flow Changes_to_be_Coordinated_de = +dt*Discover_External_Changes_d
-dt*Coordinate_Tasks_d
doc Changes_to_be_Coordinated_de = Possible improvements discovered in the upstream affecting both phases.

init Changes_to_be_Coordinated_ui = 0
flow Changes_to_be_Coordinated_ui = +dt*Discover_Changes_in_Released_Work
-dt*Coordinate_Taks
doc Changes_to_be_Coordinated_ui = Changes to be coordinated. Changes discovered by the upstream phase awaiting coordination with the downstream phase.

init Corrected_Changes_de = 0
flow Corrected_Changes_de = +dt*Correct_Changes_de
doc Corrected_Changes_de = Corrected changes in the upstream phase. The number of external errors that are corrected in the downstream phase.

init Corrected_Changes_ui = 0
flow Corrected_Changes_ui = +dt*Correct_Changes_ui
doc Corrected_Changes_ui = Corrected changes in the upstream phase. The number of changes that are corrected in the upstream phase.

init Evolution_in_Coordinated_tasks = 0*100
flow Evolution_in_Coordinated_tasks =
+dt*Decrease_of_precision_due_to_downstream_QA
-dt*Coordinate_Taks_evol
doc Evolution_in_Coordinated_tasks = Evolution in tasks to be coordinated. Knowledge gained carried by tasks waiting to be coordinated.

init Evolution_in_Released_Tasks = 0
flow Evolution_in_Released_Tasks = +dt*Release_Tasks_evol
-dt*Decrease_of_precision_due_to_downstream_QA
doc Evolution_in_Released_Tasks = Evolution in tasks released. Knowledge gained carried by tasks that are released to the downstream phase.

init Evolution_in_Tasks_Not_Checked = 0
flow Evolution_in_Tasks_Not_Checked =
+dt*Change_Evolution_Due_To_Change_of_Tasks
+dt*Change_Evolution_Due_To_Completion_of_Tasks

-dt*Release_Tasks_evol
-dt*Discover_Taks_evol
+dt*Change_Tasks_evol
doc Evolution_in_Tasks_Not_Checked = Evolution in tasks completed not checked. Knowledge gained carried by tasks waiting to be checked.

init Evolution_in_Tasks_to_be_Changed = 0
flow Evolution_in_Tasks_to_be_Changed = +dt*Discover_Taks_evol
-dt*Change_Tasks_evol
+dt*Coordinate_Taks_evol
doc Evolution_in_Tasks_to_be_Changed = Evolution in tasks to be changed. Knowledge gained carried by tasks waiting to be changed.

init Known_Changes_de = 0
flow Known_Changes_de = -dt*Correct_Changes_de
+dt*Coordinate_Tasks_d
doc Known_Changes_de = Known changes in the downstream phase.

init Known_Changes_ui = 0
flow Known_Changes_ui = -dt*Generate_Changes_During_Change_Activity_ui
-dt*Correct_Changes_ui
+dt*Discover_Intraphase_Changes_u
+dt*Coordinate_Taks
doc Known_Changes_ui = Known internal changes in upstream phase. The number of task at any one time that are waiting to be corrected, according to the present design spesification.

init Possible_Changes_de = 0
flow Possible_Changes_de =
+dt*Generating_Changes_During_Completion_Activity_de
-dt*Release_Changes_de
-dt*Discover_External_Changes_d
doc Possible_Changes_de = Changes inherited from the upstream phase.

init Potential_Changes_ui = 1000
flow Potential_Changes_ui = -
dt*Generate_Changes_During_Completion_Activity_ui
doc Potential_Changes_ui = Potential internal changes. The number of internal changes the upstream phase is prone to make. Initially set to the phase scope.

init Rework_Cost = 0
flow Rework_Cost = +dt*Increase_of_rework_cost_d

init Tasks_Complete_Not_Checked_d = 0
flow Tasks_Complete_Not_Checked_d = -dt*Release_Task_Rate_d
-dt*Discover_External_Changes_d
+dt*Completion_Rate_d
+dt*Change_Tasks_Rate_d
doc Tasks_Complete_Not_Checked_d = Tasks completed not checked in downstream phase. Once tasks are completed they are moved to this stock, waiting to be checked by quality assurance workers for changes

init Tasks_Completed_Not_Checked_u = 0
flow Tasks_Completed_Not_Checked_u = -dt*Release_Tasks_u
+dt*Change_Tasks_Rate_u
+dt*Completion_Rate_u
-dt*Discover_Intraphase_Changes_u
doc Tasks_Completed_Not_Checked_u = Tasks completed not checked in upstream phase. Once tasks are completed they are moved to this stock, waiting to be checked by quality assurance workers for possible changes.

init Tasks_Not_Complete_d = 1000
flow Tasks_Not_Complete_d = -dt*Completion_Rate_d
doc Tasks_Not_Complete_d = Tasks not complete. Total tasks of the phase scope that has not yet been done.

init Tasks_Not_Complete_u = 1000
flow Tasks_Not_Complete_u = -dt*Completion_Rate_u
doc Tasks_Not_Complete_u = Tasks not complete. Remaining tasks of the phase scope, waiting for completion.

init Tasks_Released_d = 0
flow Tasks_Released_d = +dt*Release_Task_Rate_d
doc Tasks_Released_d = Tasks released. Tasks that are released is moved to this stock. This stock represents the amount of work in this phase that is available to downstream phases.

init Tasks_Released_u = 0
flow Tasks_Released_u = +dt*Release_Tasks_u
-dt*Discover_Changes_in_Released_Work
doc Tasks_Released_u = Tasks released. Tasks that are released is moved to this stock. This stock represents the amount of work in thsi phase that is available to downstream phases.

init Tasks_to_be_changed_d = 0
flow Tasks_to_be_changed_d = +dt*Coordinate_Tasks_d
-dt*Change_Tasks_Rate_d
doc Tasks_to_be_changed_d = Tasks to be changed. Tasks that are found to contain changes are moved to this stock.

init Tasks_to_be_changed_u = 0
flow Tasks_to_be_changed_u = -dt*Change_Tasks_Rate_u
+dt*Discover_Intraphase_Changes_u
+dt*Coordinate_Tasks_u
doc Tasks_to_be_changed_u = Tasks to be changed. Tasks that are found to be faulty is moved to this level.

init Tasks_to_Coordinate_d = 0
flow Tasks_to_Coordinate_d = +dt*Discover_External_Changes_d
-dt*Coordinate_Tasks_d
doc Tasks_to_Coordinate_d = Tasks in the downstream phase waiting to be coordinated. I.e. possible imprvements in the upstream phase that are discovered. Co-ordination activity will decide which changes to implement and which to ignore.

init Tasks_to_Coordinate_u = 0
flow Tasks_to_Coordinate_u = -dt*Coordinate_Tasks_u
+dt*Discover_Changes_in_Released_Work
doc Tasks_to_Coordinate_u = TtCoord. Tasks to coordinate. Tasks discovered by the upstream phase to contain possible improvements, awaiting coordination with the downstream phase.

init Undiscovered_Changes_ui = 0
flow Undiscovered_Changes_ui = -dt*Release_Changes_ui
+dt*Generate_Changes_During_Change_Activity_ui
+dt*Generate_Changes_During_Completion_Activity_ui
-dt*Discover_Intraphase_Changes_u
doc Undiscovered_Changes_ui = Undiscovered changes in upstream phase. Internally generated mistakes made during task completion, that are not yet discovered.

aux Change_Evolution_Due_To_Change_of_Tasks =
IF(Total_Knowledge_Gained>999,0,Change_Tasks_Rate_u*Added_Evolution_With_Changing_Tasks)

aux Change_Evolution_Due_To_Completion_of_Tasks =
IF(Total_Knowledge_Gained>999,0,Completion_Rate_u*Added_Evolution_with_completion_of_tasks)
aux Change_Tasks_evol =
(Change_Tasks_Rate_u*(Evolution_in_Tasks_to_be_Changed/(Tasks_to_be_changed_u+0.000001)))

aux Change_Tasks_Rate_d = Tasks_to_be_changed_d/Avg_Change_Duration_d
doc Change_Tasks_Rate_d = Change tasks rate in downstream. The rate at which the downstream workers are able to correct changes.

aux Change_Tasks_Rate_u = Tasks_to_be_changed_u/Avg_Change_Duration_u
doc Change_Tasks_Rate_u = Change tasks rate in upstream. The rate at which the upstream workers are able to correct changes.

aux Completion_Rate_d =
TasksAvailable_for_Completion_d/Avg_Completion_Duration_d
doc Completion_Rate_d = Completion rate in upstream phase. The rate at which the upstream workers are able to complete tasks.

aux Completion_Rate_u =
TasksAvailable_for_Completion_u/Avg_Completion_Duration_u
doc Completion_Rate_u = Completion rate in upstream phase. The rate at which the upstream workers are able to complete tasks.

aux Coordinate_Taks =
Coordinate_Tasks_u*Fraction_of_changes_decided_to_implement
doc Coordinate_Taks = Coordinate tasks rate. The rate at which the project managers decide to implement discovered improvements.

aux Coordinate_Taks_evol = Coordinate_Tasks_u*(Evolution_in_Coordinated_tasks/(Tasks_to_Coordinate_u+0.000001))

aux Coordinate_Tasks_d = Tasks_to_Coordinate_d/Average_Coordination_Duration_d
doc Coordinate_Tasks_d = Coord(page 53) The rate at which the downstream phase is able to coordinate tasks that have been discovered to be faulty in downstream phases.

aux Coordinate_Tasks_u = IF(Coordinate_Tasks_d=0,0,Tasks_to_Coordinate_u/Avg_Coord_Duration_u*(Fraction_of_changes_decided_to_implement))
doc Coordinate_Tasks_u = Coord(page 53) The rate at which the upstream phase is able to coordinate tasks that have been discovered to contain possible improvements.

aux Correct_Changes_de = Change_Tasks_Rate_d
doc Correct_Changes_de = The rate at which changes are corrected in the downstream phase.

aux Correct_Changes_ui = (1-Probability_of_change_when_completing_tasks)*Change_Tasks_Rate_u
doc Correct_Changes_ui = Correct internal changes rate in upstream phase. The rate at which internal errors are corrected in the upstream phase.

aux Decrease_of_precision_due_to_downstream_QA = Discover_Changes_in_Released_Work*(Evolution_in_Released_Tasks/(Tasks_Released_u+0.00001))
doc Decrease_of_precision_due_to_downstream_QA = Decrease_of_precision_due_to_downstream_QA. Decrease of precison due to downstream phases discovering errors in tasks released, thereby making the upstream phase aware of that the precison thay thought they had accomplished was too much.

aux Discover_Changes_in_Released_Work = IF(Finalise_Switch=1,0,IF(Tasks_Released_u=0,0,QA*Probability_Discover_Change_u*((Changes_Released_ui)/Tasks_Released_u)))
doc Discover_Changes_in_Released_Work = The rate at which the upstream phase discover possible improvements in the work released to the downstream phase.

aux Discover_External_Changes_d = IF(Finalise_Switch=1,0,IF(Tasks_Complete_Not_Checked_d=0,0,Quality_Assurance_ud*Probability_Discover_Change_u*(Possible_Changes_de/Tasks_Complete_Not_Checked_d)))
doc Discover_External_Changes_d = Discover external errrors rate.The rate at which the possible improvements in the upstream phase affecting the work in the downstream phase are discovered.

aux Discover_Intraphase_Changes_u = IF(Finalise_Switch*Strategy_choice_PointSet=1,0,Quality_Assurance_u*Probability_Discover_Change_u*Probability_of_Change_u)

doc Discover_Intraphase_Changes_u = Discover intraphase changes in upstream. The rate at which the upstream workers are able to discover internally generated errors.

aux Discover_Taks_evol = Discover_Intraphase_Changes_u*(Evolution_in_Tasks_Not_Checked/(Tasks_Completed_Not_Checked_u+0.000001))
doc Discover_Taks_evol = Move_carried_precision_to_tasks to be changed. The rate at which precision is moved from tasks completed not checked to tasks to be coordinated.

aux Generate_Changes_During_Change_Activity_ui = Probability_of_change_when_completing_tasks*Change_Tasks_Rate_u
doc Generate_Changes_During_Change_Activity_ui = Generate changes during internall change activity in upstream phase. The rate at which changes are made when trying to correct a change.

aux Generate_Changes_During_Completion_Activity_ui = Completion_Rate_u*(Probability_of_change_when_completing_tasks)
doc Generate_Changes_During_Completion_Activity_ui = GenChComp. Generate changes during completion activity. The rate at which the upstream make internal mistakes.

aux Generating_Changes_During_Completion_Activity_de = IF(Tasks_Released_u=0,0,Completion_Rate_d*(Changes_Released_ui/Tasks_Released_u))
doc Generating_Changes_During_Completion_Activity_de = GenChComp. Generate changes during completion activity. The rate at which the downstream inherits changes from the upstream phase.

aux Increase_of_rework_cost_d = Change_Tasks_Rate_d*Cost_of_rework_d

aux Release_Changes_de = IF(Tasks_Complete_Not_Checked_d=0,0,Release_Task_Rate_d*(Possible_Changes_de/Tasks_Complete_Not_Checked_d))

aux Release_Changes_ui = IF(Tasks_Completed_Not_Checked_u=0,0,Release_Tasks_u*(Undiscovered_Changes_ui/Tasks_Completed_Not_Checked_u))
doc Release_Changes_ui = Release internal changes rate. The rate at which the upstream phase releases changes, due to inaccurate quality assurance.

aux Release_Task_Rate_d = IF (Release_Time_d>9999999,0,Tasks_Complete_Not_Checked_d/Release_Time_d)
doc Release_Task_Rate_d = The rate at which tasks completed in the downstream phase is released.

aux Release_Tasks_evol = Release_Tasks_u*(Evolution_in_Tasks_Not_Checked/(Tasks_Completed_Not_Checked_u+0.000001))

doc Release_Tasks_evol = Evolution release rate.The rate at which carried knowledge in tasks completed not checked, is released. In essence The rate at which the knowledge gained in the upstream phase is made known to the downstream phase.

aux Release_Tasks_u = IF ((Release_Time_u>9999999),0,(Tasks_Completed_Not_Checked_u/Release_Time_u))
doc Release_Tasks_u = Release tasks rate. The rate at which the upstream workers are able to release tasks. Releasing tasks implies letting downstream phases in on the final results that the workers has reached.

aux Added_Evolution_With_Changing_Tasks = (Evolution_choice*0.3%)
doc Added_Evolution_With_Changing_Tasks = Change in precision. To which degree the precision changes when completing or changing a task. When this is 1 the precision accomplished evolves completly in portion with tasks completed.

aux Added_Evolution_with_completion_of_tasks = 0.017+(Evolution_choice*0.40%)

aux AvailableExternalConcurrence_d = IF(Strategy_choice_PointSet=1,External_Process_Concurrence_dpb,External_Proces s_Concurrence_dsb)
doc AvailableExternalConcurrence_d = The constraint for the amount of work available in the downstream phase. This constraint is decided by the precison of the information released by the upstream phase.

aux Avg_Change_Duration_d = IF(Sensitivity_Switch=1,High_sensitivity,Low_sensitivity)
doc Avg_Change_Duration_d = Average change duration in downstream. The number of days it takes for the downstream workers to correct an error.

aux Evolution_choice = IF(Evolution_Switch_FastSlow=1,Fast_evolution,Slow_evolution)

aux Evolution_precentage = Total_Knowledge_Gained/Phase_Scope_u
doc Evolution_precentage = The precentage of potential knowledge gained in the upstream phase.

aux External_Process_Concurrence_dpb = GRAPH(FractionReleasedbyUpstreamPhase_d,0,0.05,[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1"Min:0;Max:1;Zoom"])
doc External_Process_Concurrence_dpb = The relationship deciding the amount of overlap between the ups and downstream phase when the strategy choice is point-based

aux External_Process_Concurrence_dsb = GRAPH(FractionReleasedbyUpstreamPhase_d,0,0.05,[0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9,1,1"Min:0;Max:1;Zoom"])
doc External_Process_Concurrence_dsb = The relationship deciding the amount of overlap between the ups and downstream phase when the strategy choice is set-based.

aux Fast_evolution = IF(Total_Knowledge_Gained>999,0,1000-Total_Knowledge_Gained)

aux Fraction_of_changes_decided_to_implement = IF((Changes_Released_de+Changes_to_be_Coordinated_de)=0,0,Changes_to_be_Coordinated_de/(Changes_Released_de+Changes_to_be_Coordinated_de))
doc Fraction_of_changes_decided_to_implement = The fraction of the possible improvements that is dedided to be implemented. (Only changes that affect both the upstream and downstream phase)

aux FractionReleasedbyUpstreamPhase_d = Evolution_in_Released_Tasks/Phase_Scope_u
doc FractionReleasedbyUpstreamPhase_d = The precentage of tasks released by the upstream phase.

aux High_sensitivity = GRAPH(Tasks_to_be_changed_d,0,10,[11,54,60,61,61,61,61,61,61,60,59"Min:0;Max:100"])
doc High_sensitivity = Sensitivity setting when the sensitivity is high, i.e. that changes released by the upstream phase are time-consuming to implement in the downstream phase.

aux Known_Error_Density = (Known_Changes_ui+Changes_to_be_Coordinated_ui)/(Tasks_Completed_Not_Checked_u+Tasks_to_be_changed_u+Tasks_to_Coordinate_u)
doc Known_Error_Density = The percentage of known changes in the work not released to the downstream phase.

aux Low_sensitivity = GRAPH(Tasks_to_be_changed_d,0,10,[4,4,4,4,4.2,4.8,5.1,6.8,9.3,11.9,14.4"Min:0;Max:60"])
doc Low_sensitivity = Sensitivity setting when the sensitivity is low, i.e. that changes released by the upstream phase are quickly accomodated in the downstream phase.

aux Probability_of_Change_u = IF(Tasks_Completed_Not_Checked_u=0,0,(Undiscovered_Changes_ui/Tasks_Completed_Not_Checked_u))

aux Probability_of_change_when_completing_tasks = IF(Evolution_precentage>1,0,initial_prob*((1-Evolution_precentage)/1))
doc Probability_of_change_when_completing_tasks = The probability that a task being completed contains a possible improvement that will be discovered at a later time, when the upstream workers have gained more knowledge about the work at hand..

aux Project_Duration = TIME

aux QA = IF(Tasks_Released_u=0,0,(Tasks_Released_u/Avg_Quality_Assurance_Duration_u))
doc QA = the rate at which the upstream workers review tasks released to the downstream phase for possible design improvements.

aux Quality_Assurance_u = Tasks_Completed_Not_Checked_u/Avg_Quality_Assurance_Duration_u
doc Quality_Assurance_u = Qualiy Assurance. The rate at which the upstream workers are able to test tasks for possible errors.

aux Quality_Assurance_ud = Tasks_Complete_Not_Checked_d/Avg_Quality_Assurance_Duration_u

aux Release_Error_Density_Criteria_u = IF(Strategy_choice_PointSet=1, Release_Criterium_for_pointbased,Release_Criterium_for_setbased)

aux Release_Time_Criteria_d = IF((Tasks_to_be_changed_d+Tasks_Complete_Not_Checked_d+Tasks_to_Coordinate_d)=0,0,(Known_Changes_de+Changes_to_be_Coordinated_de)/(Tasks_to_be_changed_d+Tasks_Complete_Not_Checked_d+Tasks_to_Coordinate_d))

aux Release_Time_d = IF(TIME<7,1E99,IF(Release_Error_Density_Criteria_d>=(Release_Time_Criteria_d),Nominal_Release_Time_d,1E99))
doc Release_Time_d = Release time. Release switch, that stops release of tasks as long as the known error density is higher than the user defined release criteria.

aux Release_Time_u = IF(Strategy_choice_PointSet=0 AND TIME<9,1E99,IF(Release_Error_Density_Criteria_u>=(Known_Error_Density),Nominal_Release_Time_u,1E99))
doc Release_Time_u = Release time. Release switch, that stops release of tasks as long as the known error density is higher than the user defined release criteria.

aux Slow_evolution = IF(Total_Knowledge_Gained>995,0,Total_Knowledge_Gained)

aux Stop_Condition = STOPIF(Tasks_Released_d>999)

aux TasksAvailable_for_Completion_d = IF(TIME<2,0,MAX(0, Total_Tasks_Available_d-(Phase_Scope_d-Tasks_Not_Complete_d)))
doc TasksAvailable_for_Completion_d = Tasks available for completion in downstream phase. The amount of work in tasks that at any one time is available for completion. This is contrained by external and internally concurrence relationships.

aux TasksAvailable_for_Completion_u = MAX(0, Total_Tasks_Available_u-(Phase_Scope_u-Tasks_Not_Complete_u))
doc TasksAvailable_for_Completion_u = Tasks available for completion in upstream phase. The amount of work in tasks that at any one time is available for completion.

aux Total_Knowledge_Gained = Evolution_in_Coordinated_tasks+Evolution_in_Released_Tasks+Evolution_in_Tasks_Not_Checked+Evolution_in_Tasks_to_be_Changed
doc Total_Knowledge_Gained = The total amount of knowledge gained in the upstream phase.

aux Total_Tasks_Available_d = Phase_Scope_d*AvailableExternalConcurrence_d
doc Total_Tasks_Available_d = the total number of tasks at any one time available in the upstream phase.

aux Total_Tasks_Available_u = Phase_Scope_u
doc Total_Tasks_Available_u = Total tasks available in upstream phase. I am assuming that all taks are available to the downstream phase, from the very start of the project.

const Average_Coordination_Duration_d = 20
doc Average_Coordination_Duration_d = The average time in days it takes to co-ordinate one task.

const Avg_Change_Duration_u = 20
doc Avg_Change_Duration_u = Average change duration in upstream. The number of days it takes for the upstream workers to correct an error.

const Avg_Completion_Duration_d = 30
doc Avg_Completion_Duration_d = Average change duration in downstream. The number of days it takes for the upstream workersto complete a task.

const Avg_Completion_Duration_u = 30
doc Avg_Completion_Duration_u = Average change duration in upstream. The number of days it takes for the upstream workersto complete a task.

const Avg_Coord_Duration_u = 20
doc Avg_Coord_Duration_u = The average time it takes to co-ordinate one change affecting both the upstream and downstream phase.

const Avg_Quality_Assurance_Duration_u = 20
doc Avg_Quality_Assurance_Duration_u = Average quality assurance duration. The number of days it takes to review a task for possible improvements.

const Cost_of_rework_d = 1

const Evolution_Switch_FastSlow = 0
doc Evolution_Switch_FastSlow = 1 = Fast evolution, 0 = Slow Evoultion

const Finalise_Switch = 0
doc Finalise_Switch = Switch that can be manipulated to force finalisation of the work in the upstream phase.

const initial_prob = 0.5
doc initial_prob = The probability at the start of the project that a task which is completed contains a possible change that will be discovered later on.

const Nominal_Release_Time_d = 2
doc Nominal_Release_Time_d = Nominal release time. The amount of days it takes to release a task.

const Nominal_Release_Time_u = TIMESTEP

doc Nominal_Release_Time_u = Nominal release time. The amount of days it takes to release a task.

const Phase_Scope_d = 1000
doc Phase_Scope_d = Phase scope for upstream. The size of the work that is to be done in the downstream phase in number of tasks.

const Phase_Scope_u = 1000
doc Phase_Scope_u = Phase scope for upstream. The size of the work that is to be done in the upstream phase in number of tasks.

const Probability_Discover_Change_u = 0.7
doc Probability_Discover_Change_u = Probability of discovering change in the upstream phase. The number is in percent.

const Release_Criterium_for_pointbased = 1
doc Release_Criterium_for_pointbased = The relase criteria for the point based strategy. This is high to make sure work is immidiately released to the downstream phase. I.e. the downstream phase is constantly updated on changes in the upstream phase so that they can accomodate them in their own work.

const Release_Criterium_for_setbased = 0.05
doc Release_Criterium_for_setbased = Release criteria for the set-based strategy. Because this strategy implies that tasks should only be released when the upstream is pretty certain that they will not need to be changed later on, this value is very low.

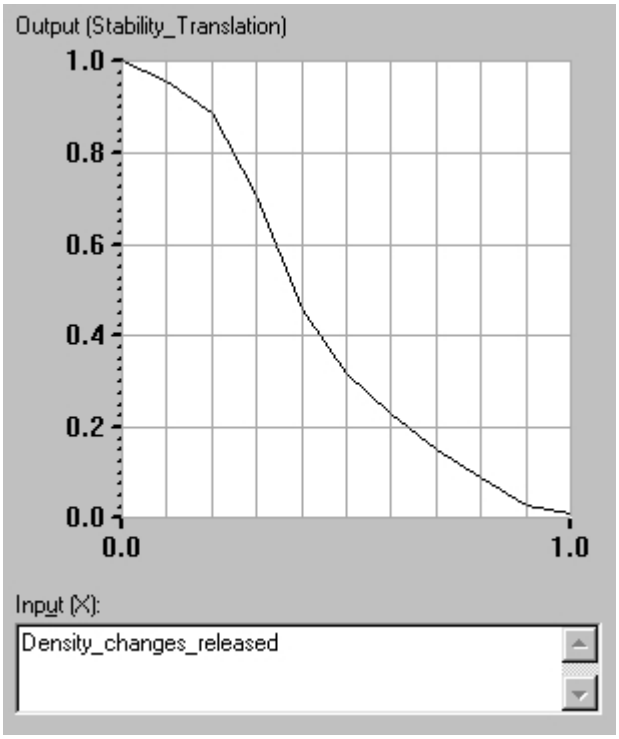
const Release_Error_Density_Criteria_d = 0.001
doc Release criterium for downstream phase. Release of tasks is stopped in the downstream phase, as long as the density of known errors in the unreleased work surpass this criterium.

const Sensitivity_Switch = 1
doc Sensitivity_Switch = Switch deciding the sensitivity in the downstream phase. 1=high, 0=low

const Strategy_choice_PointSet = 0
doc Strategy_choice_PointSet = 1=Point-based, 0=Set-based. switch deciding wich strategy to apply to the project.

Appendix B: Model Diagrams

APPENDIX C: Translation Function



Stability,in percent, as a function of the density of changes in the tasks released by the upstream phase.

Appendix D: Basic Settings

Avg_Completion_Duration_u=30
Avg_Completion_Duration_u=30

Avg_Change_Duration_u=20
Avg_Change_Duration_u=20

Avg_Coord_Duration_u=20
Avg_Coord_Duration_u=20

Avg_Quality_Assurance_Duration_u=20
Avg_Quality_Assurance_u=20

Nominal_Release_Time_u=TIMESTEP
Nominal_Release_Time_d=TIMESTEP

Phase_Scope_u=1000
Phase_Scope_d=1000

Appendix E: Glossary

Concurrent: In conjunction with development processes, concurrent depicts that several processes are ran in parallel/simoultanously.

Evolution: The level of knowledge workers have of the work at hand. In effect a measure of how close the workers are to a final result.

Finalisation: Committing to a spesific design. Beyond the point of finalisation, changes can not be done to the design. Remaining tasks will be completed according to the freesed design.

Iteration: Reworking tasks that are already completed in order to improve the quality of the work

Precision: In conjunction with information exchange, a measure of to which degree the information passed has the same form as a final result.

Sensitivity: The magnitude of impact a change in the design of the information passing process will have on the information receiving receiving process. Large impact implies that the a change will require very time consuming iterations in the downstream phase, whereas low sensitivity means that changes can be accommodated quite quickly.

Stability: in conjunction with information exchange, the probability that the information being passed is likely to change at a later point in time.