

System Dynamics in Project Management: Assessing the Impacts of Client Behaviour on Project Performance

ALEXANDRE G. RODRIGUES¹ and TERRY M. WILLIAMS²

¹Pugh-Roberts Associates, PA Consulting Group, USA, and ²University of Strathclyde, Glasgow, UK

The influence of Client behaviour on a project is complex, including schedule restrictions on milestones, high demand on progress reports, delays in approving documents, and changes to workscope throughout the life-cycle. Quantifying these effects is important to project managers, in particular for effective communication with the Client. Traditional planning and control tools have proved ineffective at providing quick and reliable information. Their tendency to increase in detail has increased their complexity, inhibiting practical strategic analyses. System Dynamics provides an alternative view in which these major influences are considered and quantified explicitly. This approach dispenses with much of the detail required by the traditional tools, but enables modelling of the systemic effects which traditional tools cannot model. The authors have developed a conceptual framework in which System Dynamics models are combined with traditional tools providing complementary support, and validated it in a large software project. This paper describes its use to assess and quantify the impacts of Client behaviour. A practical example is discussed.

Key words: project management, system dynamics

INTRODUCTION

Major projects generally involve essentially two parties. The first specifies the deliverable required, supplies the finance and will own the result or benefits of that project: the “owner” or “client”. The second consumes the finance to provide resources to execute the project, generally ending involvement at the end of the project: the “contractor”¹. This paper is being written from the point of view of a contractor carrying out a project for a client. Typical project life-cycle diagrams shown in project-management textbooks (such as the “waterfall” model for software projects¹) assume that the client's role is confined to specifying the requirement, and then approving the final product. In practice, however, it is rare for a client to place a contract and then passively await the final outcome. Indeed, the client frequently has an essential part in the project (by necessity), needing to approve sub-project milestones or intermediate product documentation in order to progress to the next stage of design. However, as well as this facilitating role, the client can cause effects detrimental to the project. He (or she, throughout) can require changes to the product definition or the project workscope; cause delays in documentation approval; cause delays in supply of essential information (e.g. about the environment within which the product must operate, or interface details); require a high level of budget and progress reporting; or can tighten milestone schedules. Where the parties are working in collaboration, these effects can often be minimised. Where the client/contractor relationship starts to become fractious, positive feed-back loops can be set up: for example, high progress-reporting demands will reduce productivity, causing the schedule to slip and an untrusting client to demand even more progress-reporting. Many other such positive feedbacks will be discussed in this paper.

These effects must be analysed and quantified: to ensure that the project costs and time do not run out of control as vicious circles are set up; to estimate correctly the true cost of contract amendments; and

where necessary to make auditable claims against the client. The quantitative effect of a single client action on a simple project is straightforward to analyse. However, this is not so when many client actions are considered, and certainly when considering the subjective nature of some effects such as: the effect of increased schedule pressure on workers' morale; the effect of schedule slippage on the clients' trust on the project team, and the effect of this on increased reporting requirements.

Early advances in project management were based on decomposing the project into its constituent parts in a structured way (e.g. PERT, or the Work Breakdown Structure). It has become clear that these traditional techniques alone are inadequate for analysing and managing modern complex, integrated projects². In particular, systemic effects, such as the effect of multiple client actions, cannot be taken into account by such decomposition methods, and a holistic approach is necessary. Cooper³ blames lack of management understanding of the systemic effects of actions for much failure of development projects.

Cognitive maps and influence-diagrams provide a natural approach to modelling such effects qualitatively. A natural extension is to develop these into System Dynamics (SD) models, able to capture both hard systemic effects and softer "human" effects which are also important. An example of this process is given by Ackermann et al⁴, who discuss the post mortem litigation analysis of a project. This is also the route taken by Cooper³: he describes SD models of projects which, he claims, accurately quantifies both hard and soft influences within the model. The authors therefore discuss in the following paper the practical use of SD models to evaluate the impact of client behaviour in a project, based on a practical action case-study using SD models integrated with traditional tools.

APPLICATIONS OF SYSTEM DYNAMICS

Several applications of SD to project management have been reported since 1964. The first project management model was developed by Roberts⁵ to examine the dynamics of R&D projects. Since then, considerable academic work has continued, in particular at MIT⁶. It was not until the early 1980's that practical work started being developed at Pugh-Roberts Associates⁷ which culminated with the PMMS⁸, a complex and powerful SD tool intended to support the planning and control of large-scale projects. Of major practical relevance has also been the work developed at NASA⁹⁻¹¹, in particular the models developed by Abdel-Hamid⁹ and Lin et al¹⁰⁻¹¹. In Europe, other relevant practical work is the "Shuttle Wagons" model developed by Williams et al¹²⁻¹³. Both academic and practical work has continued to be developed while the usefulness of SD in project management has gained considerable recognition. A more detailed review can be found in Rodrigues and Bowers¹⁴.

It is important to note that the major applications refer to supporting dispute resolution^{7,12,15} where the user has recognised the practical and distinctive usefulness of the approach. In these applications SD models were used to identify and quantify the full impacts of disturbances introduced by one of the parties involved in the project. In particular, this includes delays in approving design documents and introducing changes in the requirements after development/production has been initiated.

THE CLIENT BEHAVIOUR AND THE PROJECT DYNAMICS

The difficulties in assessing the full impacts of Client behaviour on project performance derive in part from the subjectivity involved in identifying how this behaviour interferes with project implementation. We first need to identify the Client's actions that are likely to have a relevant impact. The relationship

between the Client and the contractor during the project can be characterised by two different communication processes, from which problems with the Client usually emerge.

- the continuous reporting of progress regarding the major milestones, needed to keep the customer confident that the contractual agreements are being respected by the contractor
- the continuous review of the system definition and its required functionality, aiming to ensure that parties have a clear common understanding of the definition of the product being developed.

Projects are novel endeavours, so rarely follow a pre-planned steady path. As they grow more complex they becoming more sensitive to external disturbances. In the authors' experience, the following sequences of events are typical (see also refs 8, 9, 12 and 15). As the contractor fails to meet milestones the Client feels the right to compensation while reducing his trust in the contractor's competence. Alternatively, the contractor's means of recovering the short-term milestones might be at the expense of worse over-runs later. On the other hand, a need to sacrifice early milestones in favour of later success may not be accepted by the Client and is often "culturally" seen within organisations as a premature indication of failure. The Client's reluctance to accept early delays can be reinforced by the contractor's difficulties in providing convincing arguments. Faced with a hostile Client attitude the contractor may choose to avoid reporting delays whenever these can be "hidden" hoping that staff will soon catch up with the work. As communication deteriorates, co-operation between Client and contractor reduces and conflicts become more and more counter-productive.

The continuous review of the system definition can also cause communication problems. As the system is developed throughout the life-cycle, intermediate sub-products are assessed to help identify misinterpretations of the system requirements. While this process aims to evolve towards a common and clear understanding, the result can a growing disagreement between the parties. Again, typically the Client tends to demand more or "better" system functionality than that which the contractor believes to have been agreed, either because of ambiguities in the contract or as compensation, say, for delays. The contractor's major concern might be to reduce costs, so might tend to accept low-cost changes while rejecting those which appear more expensive. However, the Client's perceived value may not relate to the implementation cost (which he might not know) but to the functional utility of the change. Thus a low cost change readily accepted by the contractor could be perceived by the Client as a valuable concession encouraging the demand for further changes, while refusal to accept costly changes perceived as trivial by the Client causes mutual trust to deteriorate.

Figure 1 is an influence diagram representing this feedback structure, with the key "vicious circles" identified. This figure was derived from interviews during the action case study described below, but similar effects are described in the literature quoted above, for example refs. 8, 9, 12 and 15. The control loop at the centre shows how the introduction of changes is usually balanced by schedule adjustments negotiated with the Client based on an estimate of perceived extra effort (i.e. the direct cost). The dashed lines identify the secondary effects, usually complex and subjective, which in the long term are the cause for over-runs. The first major effect of changes in the system's requirements is work having to be done out of its normal sequence (for example, coding having to proceed on the basis of unavailable or unstable design information). Other major causes for work out of sequence are Client delays in approving system design documents¹³, and staff under schedule pressure directing their efforts to those system areas they believe to be more stable. The immediate consequence of doing work out of sequence is an increase in the number of errors. As new errors start being detected in areas which were thought to be stable, staff progressively lose trust in the current system's requirements. Poorer understanding about the system functionality reduces development productivity

and when associated with schedule pressure causes the effectiveness of QA activities to deteriorate, so they are skipped or compressed. With lower productivity and a high number of defects escaping to the later testing stages major schedules are inevitably delayed. These slippages tend to cause the Client's trust in the project team to deteriorate, the Client becoming increasingly less tolerant in accepting schedule adjustments and more demanding in progress reports which divert staff from the real development work. Often, the only way of persuading the Client to agree with delays is to concede further changes in system requirements at no extra cost. This continuous introduction of changes exacerbates the lack of trust in the stability of the requirements, which motivates staff to do work out of sequence even in those areas where the requirements are stable. As stated by a senior manager in the case study below, "...while doing work out of sequence might be a good approach on the 10% of requirements that are in doubt, it could be disaster in the 90% where the requirements are clear and stable." The secondary ripple effects are identified in the influence diagram by reinforcing causal loops in which the Client's actions (squared boxes) play a core role.

Figure 2 shows how the Client behaviour interferes with the project dynamics. Faced with a slippage, management usually decides between readjusting the schedule or finding ways to increase the work rate, or a combination. Finding the appropriate balance is a difficult task and Client can exacerbate the problem: aggravating delays and imposing restrictions on schedule adjustments puts pressure on management to try increasing the work rate (e.g. hiring more staff, putting pressure on staff to work quicker, using over-time, reducing QA, and increasing activity concurrency). However, when over-applied, the several disruptive long-term effects become dominant reinforcing problems (as shown by the reinforcing loops in dashed lines). Once more, these secondary effects are difficult to estimate and quantify. Thus, if not managed properly the Client behaviour not only has the potential to disturb the normal work progress but also to encourage the implementation of inappropriate control decisions.

Client behaviour has a strong impact on the project dynamics and if not handled properly may disrupt project performance dramatically. Initial contractual agreements are often violated and changes need to be agreed. Both parties can play counter-productive adversarial roles, ending in costly legal disputes. Effective negotiation while the project is on-going is therefore essential but can only be achieved through an understanding and accurate quantification of the full impacts of changes.

USING THE SYDPIM

Traditional planning and control tools have proven not to match this need. System Dynamics (SD) provides an appropriate alternative to approach this problem. But such a method needs to be embedded within the existing tactical decision-making structure. The authors have therefore developed the System Dynamics-based Project-management Integrated Model (SYDPIM), a model integrating the use of SD models with traditional methods to support on-going project planning and control, as described in Rodrigues and Williams¹⁶. The use of SD models is focused on three major roles: (1) estimating, (2) risk analysis, (3) progress monitoring and diagnosis (assessing the impacts of Client behaviour is a major case of risk analysis). The SYDPIM uses SD models at both the strategic and the operational management levels, to provide continuous support to the planning and monitoring functions. This is based on the continuous calibration of the models to replicate past behaviour and to capture the future plans. Within the planning function the models are used to estimate the project outcome and to help identify better planning alternatives, ensuring that the plan is robust and based on realistic assumptions. In monitoring, the models are used to uncover several metrics about the project status, and as a diagnosis tool they help identify possible causes for observed deviations.

THE KDCOM CASE-STUDY

The SYDPIM was improved and validated in a large-scale software intensive project at BAeSEMA, developing a *Command and Fire Control System* for the Korean Destroyer “KDCOM”. The project, scheduled for 4 years, comprises the development of several sub-systems, both hardware and software. The prime contractor is BAeSEMA, and there are several sub-contractors from Europe and Korea. BAeSEMA is directly involved in the development of the software component of the Command and Control System (the C2 SW System), scheduled around 32 months, which is being implemented in two main builds (SWB1 and SWB2), overlapping with estimated durations of 14 and 30 months respectively.

Traditional planning and control procedures were being used based on a detailed Work Breakdown Structure. Gantt Charts were used to identify general schedules and major milestones, while logical networks identified major work dependencies and supported resource allocation and risk management¹⁷. The company was committed to exploring new approaches such as Boehm's spiral model¹⁸. KDCOM therefore provided an excellent case-study to explore the practicability of SD.

The strategy adopted for the action case-study was to use SWB1 as the basis to develop, validate, and calibrate the SD models. Validation focused on the accurate replication of the past behaviour of this build followed by a tentative *post mortem* analysis. Informal interviews with managers and staff provided the required feedback to improve confidence in the model. Once model development, calibration and validation had all been achieved successfully for SWB1, the models were then used amongst other tools to support on-going planning and control of SWB2.

Client behaviour is of crucial importance within the management of the KDCOM project. Of particular relevance was the subjectivity involved in interpreting the several contractual agreements. This problem tended to be exacerbated by cultural differences which might create a major obstacle to effective communication. Consequently, the threat of changes being introduced in the system requirements, particularly during the middle and later stages of the life-cycle, posed a major risk. Estimating and quantifying the final impacts of these changes on the major schedules, costs, and product reliability were recognised by management to be a major problem, as were the inadequacies of traditional tools in capturing their secondary downstream effects. Below we provide an example of how SD models can be used to cover this need providing valuable information to the manager. Although this particular example is fictitious the data used to calibrate the model is real.

MODEL DEVELOPMENT

Our approach to model development was based on the SYDPIM. Rodrigues and Williams¹⁹ discuss the definition of a model structure for a software project in more detail. This breaks down the software development process into individual sub-tasks, each implemented by an individual sub-model specialised to that type of work, mainly one of three generic types:

- (1) development task: developing a particular sub-product, reviewing and reworking (e.g. design and coding fall in this category);
- (2) testing task: checking and correcting the functionality of a sub-product (usually, running a set of pre-defined tests, identifying and diagnosing failures, and reworking the defects found; unit testing, component testing and system testing fall in this category);

(3) integration task: taking two or more sub-products and delivering an integrated product. This usually includes developing or changing connectivities between sub-components, testing these connectivities, and making any necessary changes in the sub-components.

While this classification is generic for the software development process, other more specific approaches may be adopted. There are precedence relationships between sequential tasks but their implementation may overlap, while tasks with no relationship can be implemented in parallel. Each task has an internal management process responsible for monitoring progress and taking short term internal corrective actions. A high-level management component implements the overall process of monitoring and re-planning the development and testing tasks, while a human resource management component captures the process of hiring/firing and training staff.

An informal outline of the generic structure of the SD model that captures a development task (the SD-DTModel), is represented in figure 3. The model considers two parallel processes:

- the engineering process, responsible for the physical development of the task output product. This comprises the life-cycle of two interrelated entities: work and defects. The work entity flows throughout activities of development, review and rework (although sequential implementation of these activities usually overlaps). Defects are generated while work is being developed and are eventually detected in the reviews and removed by rework. However, some defects are not detected in the review process and others are not properly reworked, so they *escape* and are incorporated into the task output product.
- the management process, comprising the functions of monitoring progress and re-planning. Monitoring consists of estimating the amount of work performed, effort spent, past and expected future productivity, then forecasting effort needed to complete, and cost and spend at completion. The planning function performs man-power allocation amongst the three engineering activities (development, review, and rework), adjusts man-power availability by using schedule pressure and over-time, and re-adjusts the schedule whenever necessary, these decisions being conditioned by higher level management policies represented explicitly in the model.

An aspect of crucial importance in this model is to consider and quantify explicitly the intangible “defects escaped” providing a measurement for the product *quality*. This enables a quantitative assessment of the triangular trade-off between cost, time, and quality.

Figure 3 then represents the key structural aspects of the SD-DTModel simulation model. The fully quantified structure is of course much more complex, incorporating several hundred equations and cause-effect relationships. The major feedback effects that produce the behaviour in the model stem from these detailed relationships, and some are those shown in the influence diagrams of Figures 1 and 2.

In our example, we use an SD-DTModel to model a development task: the design phase of a C2 SWB1 sub-component called SAM/STIR. The design phase of each C2 SWB1 sub-component comprises three stages: high-level design, requirements clarification, and detailed design. Throughout these stages a detailed logical design of the software sub-component is produced, along with the development of functional test schedules and specifications. Each stage is defined as having a set of input and output documents and a breakdown of steps to define the development process. The documents produced are the main input to the following life-cycle stages of coding and testing. The complexity of the work increases throughout these stages as well as the staff understanding about the system functional requirements.

MODEL VALIDATION AND CALIBRATION

We first need to assure ourselves that the model is validated and calibrated. In System Dynamics, validation requires not only that the model is in accordance with the known "physical laws" of the system, but also that it offers usefulness and transmits confidence to the end user (i.e. the manager)²⁰. Usefulness depends on whether the model is addressing the problem areas where the manager needs support, while confidence rests on the model's ability to produce results consistent with managers' mental models. After the model structure has been validated its practical application is based on calibration for real situations, in itself a delicate process which should also be subjected to validation.

Initially, then, we must ensure that the SD-DTModel reflects the specified software development processes. Informal interviews were carried out to check whether all relevant factors were included in the model. The process of testing and readjusting the model to achieve a stable and robust structure was based on three steps.

- (1) Test the model component that represents the engineering process. The model should produce a steady behaviour when calibrated for simple ideal scenarios with progress evolving towards planned targets. This minimises the role of the management component, which would be restricted to re-scheduling available man-power amongst the three engineering activities.
- (2) Test the management component. Disturbances were introduced so that deviations from targets would occur and be detected by the management component. This would test how the component would perceive the deviations (i.e. magnitude, delay), and the corrective decisions generated, such as using schedule pressure, re-adjusting the schedule, using over-time, or skipping QA. The behaviour produced by the model should reflect the managerial policies: for example, if a policy of minimising schedule slippage was considered then the model should react to delays by using schedule pressure and over-time (hiring staff was not included, as that would be part of the higher-level management component overseeing all sub-components; in reality, increasing team-size while the work was underway was not being considered).
- (3) As confidence in the model increased, "hyperbolic" tests were used, experimenting with very unlikely extreme scenarios (e.g. staff productivity extremely low or high). This was important as a confidence test to ensure that the core underlying logic of the model was stable and robust.

During the first two steps the model was calibrated with very simple data so that the behaviour produced would depend mostly upon the logic of the feedback structure, avoiding the effects of more subjective or not well known time-varying "natural feedback" (e.g. non-managerial feedback like the impact of learning on productivity, effort to detect and rework defects and impact of schedule pressure on error generation).

The model was then calibrated to reproduce observed past behaviour of the design phase of some software sub-components. (This required appropriate metrics data to be collected from the project information system: ensuring that this data can be obtained within a short period of time and with accuracy, while the project is underway, is a crucial issue for the practical application of SD models not discussed here). Such calibration is a delicate task, since the model considers many parameters which are either intangibles or not measured in practice. For example, several combinations of the parameters "real error generation index" and "real effort to detect an error" can generate the same number of errors detected, the difference being in the intangible variable "number of errors escaped". Another example is the effect of schedule pressure on productivity: under pressure, staff work extra hours without reporting over-time; once the model considers this phenomenon explicitly, *real* productivity will be lower than *reported* productivity, the difference depending on the unmeasured non-reported extra

hours. Thus a considerable number of variables needing to be calibrated cannot be directly deduced from the metrics collected.

We therefore define input metrics which are model parameters (e.g. real productivity, real error generation, planned schedule), and output metrics which are variables reflecting the results of the simulation (e.g. reported productivity, actual schedule). Some of the metrics collected from the project can be directly translated into model's input metrics, while others will have to be matched by the model's output metrics. This is achieved by testing several combinations of the unmeasured input metrics. It can be expected that more than one combination will reproduce the observed results and hence validation needs to be carried out so that the more appropriate one is selected. Once more, this validation process is based on confidence tests using "what-would-have-happened-if" type of scenarios. An appropriate calibration should not only be able to reproduce well the observed past behaviour, it should also ensure that the model reacts in an acceptable manner (i.e. according to managers' mental models) in other hypothetical scenarios.

In summary, before using the model two validation steps must be carried out carefully:

- the model's feedback structure must be able to capture the general dynamics of project behaviour (while a model should be subject to continuous improvement it is helpful if a stable structure is achieved so that management can concentrate on model application; another technical team could carry out structural improvements in parallel separately)
- the calibration parameters for a specific situation must be as close as possible to their real occurrences.

Both steps are strongly based on empirical judgement of what are "acceptable" model results, so the participation of managers is essential. In the practical example discussed in the next section both model development and the presented calibrations have been validated as described above.

PRACTICAL EXAMPLE

In this practical example the SD-DTModel will be used to analyse the impacts of introducing requirements changes to the SAM/STIR component during the later stages of the design phase. This situation is fictitious and the same type of analysis can be applied to other areas and phases. We focus here on the importance of managing changes in the early design phase, when a detailed plan is not available for the later stages. Some of the metrics herein presented have been normalised but are based on real data collected from the project.

SAM/STIR design: plans and model calibration

The design phase of the SAM/STIR component was planned according to the figures in table 1. The estimated size of this functional area measured in DSI (lines of code) represents about 12% of SWB1, which incorporates another eight functional areas. The breakdown of this budget was based on a project-wide policy of allocating 19% to each of QA (design reviews in this case) and rework activities. The average daily man-power was 2.88 full-time-staff (FTS), and given the short time-period management were not considering changing the staff level after the work had been initiated.

Metric	Planned Estimate	Unit
1. Estimated size	5 551	DSI
2. Schedule	84	day
3. Budget	242	man-day
(excluding management)	62% (150)	
3.1 Development	19% (46)	
3.2 QA (reviewing)	19% (46)	
3.3 Rework		
4. Productivity		DSI/man-day
4.1 Gross (1/3)	23	
4.2 Development (1/3.1)	37	
5. Daily man-power (3/2)	2.88	FTS

Table 1 - Planned metrics for SAM/STIR design phase

The first step was to calibrate the model to reproduce the steady behaviour implicit in the plan. While the metrics of table 1 could be directly translated into some of the model's parameters there were many other parameters needing to be calibrated, namely policy parameters under direct management control (e.g. length of the review period, delay in reworking detected defects, tolerance to adjust schedule), and intangible process parameters reflecting intrinsic characteristics of the development process, many of human nature. A typical example of the latter is the effect of learning on productivity and error generation: as staff become more familiarised they tend to make less errors and work quicker; on the other hand, as the design moves into more detail the increasing complexity of the work eventually counters this improvement; intangible factors such as schedule pressure, staff exhaustion, or error density also influence productivity and error generation. Such relationships are defined in the model by input equations: as these relationships cannot be measured directly from reality we used approximations based on past data, managers/staff experience (i.e. their mental models), and on available literature (a good review can be found in Abdel-Hamid and Madnick⁹).

This calibration process led to the definition of an appropriate set of parameters so that the model reproduced the plans. The general dynamic behaviour is shown in figure 4(a) where both targets of cost and schedule remain roughly constant (curves 1 and 2); curve 3 shows the work progress and curve 5 the amount of remaining rework. Figure 4(b) shows how the man-power is scheduled among the three engineering activities. In the calibration we considered that productivity would slow down in stage 2 (requirements clarification) due to an increased complexity in the design work. The consequence is an increase of man-power allocated to development in this stage and a slow down in the rework activity. As staff gets more familiarised with the details and productivity increases, man-power is re-directed to rework towards the end of stage 3; finally there is a week of revision and a week of rework. The model suggests a light "QA skipping" which is possibly caused by the difficulties in stage 2.

The immediate benefit from this calibration was the explicit definition of some important metrics which otherwise would remain hidden in the plans. For example, it was necessary to estimate the error introduction rate, effort required to rework an error, and to define how these would be influenced by the work progress. At this stage, such an exercise may help identify unrealistic assumptions and suggest readjustments in the plan. Our next step was to investigate what would be the impacts of introducing design changes.

Modelling the introduction of changes

Calibrating the SD model to analyse the impacts of changes requires the definition and quantification of some parameters:

- (1) a measure for the magnitude of the changes, defined as the percentage of work developed that needed to be re-done due to the changes
- (2) a description of how these changes are introduced over time, defined by an "S-shaped" curve identifying the cumulative fraction of (1) being introduced over time: the steeper the shape the more sudden the introduction of changes would be
- (3) identification and quantification of the *direct* effects of changes on several process metrics (e.g. error generation, development productivity, effort to rework). Based on managers' opinion some direct effects were selected and their impacts were tested in the model. Where the impact is irrelevant, inclusion of the parameter could not be justified. After experimentation, it was found that the most relevant effect was the impact on error generation (this result was previously highlighted in the influence diagram of figure 1). Besides, the later the change is introduced the stronger the effects of "work out of sequence", and hence we also considered that these would grow exponentially. These assumptions were tested separately with simple scenarios.

Modelling schedule adjustment policies

When changes are introduced there is an immediate need to reassess whether the final schedule should be readjusted. While the contractor reaches an agreement with the Client regarding the milestone, management has then to decide how to pass this onto staff. Typically, management agrees a shorter deadline with development staff as both a contingency measure and to ensure a minimum level of schedule pressure (the scheduled duration shown in table 1 refers to this date while the agreement with the client was of 100 days, hence 16 days of contingency). This involves deciding the following policies.

- (i) How management reacts to slippage reported by the development staff in order to avoid over-running the schedule milestone. This was captured in the model according to the curve of figure 5: the x-axis represents an ordinal measure of how close the completion date reported by the development staff is to the schedule agreed with the Client (a value of 0 represents that the staff is reporting within the initially agreed schedule, and 1 represents reporting a delay which will extend the completion date to the schedule milestone agreed with the Client); the y-axis represents the fraction of the consequent schedule extension requested by the staff that managers are willing to accept. As an example, if the staff reports a completion date of 92 days (half-way between 84 and 100, hence 0.5 in the x-axis), a delay of 8 days, then managers are only willing to accept 40% (y-axis) of that extension (i.e. $40\% \times 8 = 3.2$ days), and hence the schedule agreed with the staff would be extended to day 87. The overall policy, based on current practices, shows that while early reported slippages are not accepted and pressure is put on the staff, as slippage becomes closer to the schedule milestone agreed with the Client, management starts accepting a fraction of this slippage in order to avoid the negative effects of pressure. However, once these delays have been allowed, management again starts refusing further reported slippage threatening the schedule milestone.
- (ii) Given a certain amount of introduced changes how many extra days are given to the development staff in order to minimise slippage within an acceptable level of quality. This is captured in the model by a policy parameter representing the proportion of schedule adjustment given to the development staff in relation to the proportion of the changes introduced, ranging from 0% to 100%. As an

example, a 50% level means that given the duration agreed with the Client of 100 days, a 30% introduction of changes would be matched by giving to the staff a schedule adjustment of 15 days (i.e. $100 \times [30\% \times 50\%]$) - we herein refer to this parameter as the "adjustment fraction."

Using the SD model to assess the impacts of introducing changes

In this example we consider a 30% magnitude of total demanded changes being introduced smoothly, starting at 75% of the planned progress and ramping at 90%. The model allowed us to assess the impacts of these changes at the end of the design phase and to test several schedule adjustment policies. The main purpose of this experimental investigation was to provide a qualitative *and* quantitative analysis of the triangular trade-off between cost, quality, and time. While the long term cost savings of early error detection are considerable^{21,22}, it is difficult to estimate this during the design phase, when the product has not yet materialised. Therefore, the problem of managing changes within this phase is not so much about controlling the cost and schedule but assessing how these are traded against quality. Our experimental investigation focused on this problem.

The assessment of different scenarios was based on three output metrics: schedule achieved, cost, and defects escaped. Providing explicit estimation of this last intangible as a measure of quality is of crucial importance. In practice, using the numbers of detected errors leads to the self-fulfilling tendency of skipping QA after "enough" errors have been detected. In analysing schedule adjustment policies the curve of figure 5 was not changed since it represents a project-wide generic policy of managing schedule contingencies. Instead, the parameter "adjustment fraction" was changed in each simulation in order to investigate how the schedule agreed with the development staff should be readjusted in the face of introduced changes.

Figure 6 shows the behaviour produced when changes are introduced and no schedule adjustment is made. Here "schedule pressure" is put at its maximum as an attempt to cope with the changes and still achieve the original schedule. The dashed line (curve 4) in figure 6(a) shows the "S-shaped" introduction of changes starting roughly at day 65. The impacts on the general behaviour are an immediate steep decrease in the amount of work completed (curve 3) while later schedule slippage cannot be avoided (curve 1), and cost exceeds the original budget. Figure 6(b) shows how all man-power is suddenly directed into development in an attempt to implement the changes quickly. Consequently, rework is delayed and there is a cut in the man-power allocated to the reviews (curve 2). This "QA cut" does not affect the review progress: under high schedule pressure staff spend less time reviewing documents but still report a normal reviewing rate. As a result of this poorer QA activity more defects are likely to escape. After changes have been implemented and development is completed man-power is directed back to rework and reviewing.

Although this restrictive scheduling policy does not avoid a delay, the results still appear attractive: a 14% over-run and a 13% over-spend (less than half of the proportion of changes). Furthermore, the increase in the amount of defects detected was in the same proportion of the changes (about 30%), reflecting how staff implicitly assess QA progress when under schedule pressure: "if I have to re-design 30% more then I should find 30% more defects". However, the (intangible) number of defects escaped suffered a steep increase of 72%. This is a consequence of two major secondary effects identified in the influence diagram of figure 1: an increase in error generation, and a decrease in QA effectiveness. Although the long term consequences of a high defect escape rate are not visible at the

end of the design phase, they will emerge later in the project when corrective actions are more difficult and very expensive.

We used the model to explore the benefits of extending the schedule to reduce the negative effects of schedule pressure. Figure 7 shows deviations from the base case when no changes are introduced. Alleviating schedule pressure takes effect beyond the 20% level of the parameter "adjustment fraction", where increasing the schedule adjustment to balance the changes results in later completion and more over-expenditure. However, the gains in the number of defects escaped are of much greater magnitude. Beyond the 60% level the quality of the designs does not improve significantly while cost and schedule keep increasing. This suggests that an appropriate level of schedule adjustment would be around 60%. For example, given the duration agreed with the Client of 100 days, if 30% of changes are introduced, then 18 days (i.e. $100 \times [30\% \times 60\%]$) of delay should be passed on to staff. In our example the completion time would be extended from 84 days to 102 days, as changes are continuously introduced. Testing this policy in the model shows that the actual completion date would be day 104, a 4 day over-run in the eyes of the Client. The behaviour produced is shown in figure 8 where the patterns assume smooth shapes similar to the base case (figure 4). In the case where the schedule milestone of 100 days could not be extended the model would suggest an adjustment fraction around 40% (12 days). For a more detailed analysis, numerical results are shown in table 2.

Results	Base Case	Adjustment Fraction						Unit
		0%	20%	40%	60%	80%	100%	
Schedule	85	97	97	101	104	108	114	day
Cost	241	276	273	282	291	301	309	man-day
Productivity	23.0	20.1	20.3	19.7	19.1	18.4	18.0	DSI/man-day
Defects Detected	1.00	1.30	1.28	1.40	1.45	1.45	1.45	error/KDSI (*)
Defects Escaped	1.00	1.72	1.71	1.48	1.34	1.32	1.31	error/KDSI (*)

Table 2 - Numerical results: percentage deviations from base case

(*) Note that defect metrics have been normalised

The main conclusion from this investigation is that when changes are introduced later in the design phase management must keep a tight schedule. This is for two reasons: (1) implementing design changes under pressure exacerbates error generation and disrupts QA effectiveness, so a high number of defects are likely to escape; and (2) while the quality of the designs is not tangible at this stage the long term impacts of low quality are very important in later phases. A practical tool capable of providing quick qualitative and quantitative analysis to support negotiation of changes with the Client while the project is on-going, is of major value to the manager.

CONCLUSIONS

Avoiding legal disputes and achieving a collaborative relationship between Client and contractor, based on mutual trust, is a major priority for project-oriented organisations, particularly in the software development field. This requires an understanding of Client behaviour. Traditional tools and techniques alone have proven inadequate to provide quick and reliable information to the manager, necessary to support effective negotiation.

The *holistic*, integrative perspective of System Dynamics is more appropriate. SD models assume a higher level view of the whole project, focusing on human factors and managerial policies. They have an inherent flexibility which enables them to incorporate a wide range of influences specific to particular scenarios. Traditional models are more specialised, taking a detailed view of the individual project elements. They are thus more rigid, which can make their use easier but at the expense of reality: whilst ensuring rigorous monitoring of the project past, their view of the future is focused on a “planned success”. In contrast, SD simulation models provide a laboratory to test several different scenarios for the project, delivering a clearer and perhaps more realistic view of the possible futures.

While SD has been applied with considerable success in supporting post-mortem dispute resolution, a more proactive and beneficial application would be to support on-going negotiation with the Client before problems occur. Using SD models alone has significant disadvantages during the course of the project, as they do not provide the operational support and specific advice that project managers need. What is needed, therefore, is a well defined framework integrating SD models within the traditional approach. The authors have published previously such a framework (the SYDPIM) and in this paper we have described how it can be used in practice. In the practical example herein discussed, based on a large software project, we have used a SD model to investigate how schedule adjustments should be managed and renegotiated with the Client so that the downstream effects of introducing requirements changes during the later design stages can be minimised.

This has provided useful high-level management support. However, there is still a need for specific operational advice. Our current research therefore focuses on developing the integration of the SD models within the traditional procedures, to provide complementary and synergistic support.

Acknowledgement -- This work has been developed in the course of a PhD programme undertaken at the University of Strathclyde, UK, funded by Junta Nacional de Investigação Científica e Tecnológica (JNICT), Portugal; and supported by BAeSEMA Ltd., U.K. A longer version of this paper was presented at the 9 Young OR Conference, March 1996, York, UK having been given the *Mike Simpson Award for 1996* from the Operational Research Society, and was further presented in plenary at the SD'96 Conference, MIT, Boston, USA.

REFERENCES

1. Turner JR (1993) The handbook of project-based management. McGraw-Hill, London.
2. Williams TM (1995) Holistic methods in project management. In *Proc.INTERNET Symposium*, St.Petersburg, Russia, pp. 332-336. 14-16 September 1995.
3. Cooper KG (1994) The \$2,000 hour: how managers influence project performance through the rework cycle. *Proj.Mgmt.J.* **25**, March 1994, 1, 11-24
4. Ackerman F, Tait AJ, Williams TM and Eden C, (1994) COPE-ing with System Dynamics -a story about Soft and Hard OR. Young O.R. Conference, York U.K., March 1994.
5. Roberts E (1964) *The Dynamics of Research and Development*. New York, Harper & Row.
6. Kelly T (1970) *The Dynamics of R&D Project Management*. M.I.T. M.S. Mgt (300)
7. Cooper KG (1980) Naval ship production: a claim settled and a framework built. *Interfaces* **10**, 6, 30-36.
8. Pugh Roberts Associates - PA Consulting Group (1993) *PMMS - Program Management Modeling System*. PA Consulting, Cambridge MA.
9. Abdel-Hamid TK and Madnick S (1991) *Software Project Dynamics: An Integrated Approach*. New Jersey, Prentice-Hall.
10. Lin C and Levary R (1989) Computer-Aided Software Development Process Design. *IEEE Trans. Software Engineering* **15**, 9, 1025-1037.
11. Lin C (1993) Walking on Battlefields: Tools for Strategic Software Management. *American Programmer* **6**, 5, 33-40.
12. Williams TM, Eden C, Ackermann F and Tait A (1995) Vicious Circles of Parallelism. *Int.J.Proj.Mgmt.* **13**, 3, 151-155.
13. Williams TM, Eden C, Ackermann F and Tait A. (1995) The Effects of Design Changes and Delays on Project Costs. *J.Opl.Res.Soc.* **46**, 7, pp. 809-818.
14. Rodrigues A and Bowers J (1996) System Dynamics in Project Management: a comparative analysis with traditional methods. *Sys. Dyn. Rev.* **12**, 2, 121-139.
15. Weil HB and Etherton RL (1990) System Dynamics in dispute resolution. In *Proc. 1990 Int. System Dynamics Conf.*, pp. 1311-1324. System Dynamics Society, Lincoln, MA.
16. Rodrigues A and Williams T (1995) The application of system dynamics in project management: an integrated model with the traditional procedures. *Working Paper 95/2*. Dept. Management Science, University of Strathclyde.
17. Williams TM (1990). Risk Analysis using an embedded CPA package. *Int.J.Proj.Mgmt.* **8**, 2, 84-88.
18. Boehm B and Papaccio P (1988) Understanding and Controlling Software Costs. *IEEE Trans. Software Engineering* **14**, 10, 1462-1477.
19. Rodrigues A and Williams T (1996) System Dynamics in Software Project Management: towards the development of a formal integrated framework. (*To appear in Eur. J. Inf. Sys*)
20. Forrester J and Senge P (1980) Tests for building confidence in system dynamics models. *TIMS Studies in Management Sciences*, **14**, 209-208.
21. Boehm B (1981), *Software Engineering Economics*. New Jersey, Prentice-Hall.
22. Pressman RS (1987) *Software Engineering. A practitioner's approach*. McGraw-Hill, Singapore.

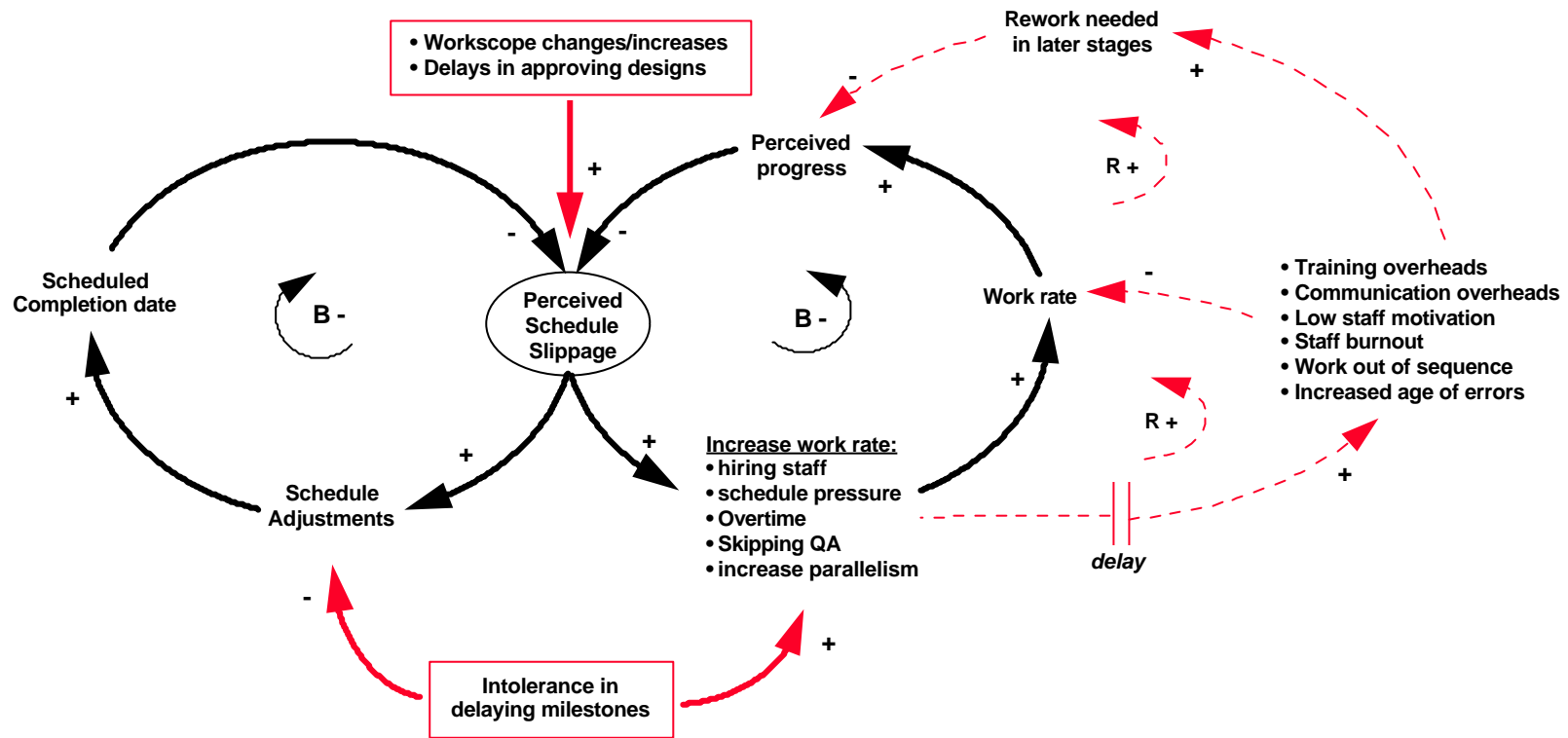


Figure 2 - Client behaviour can restrict effective control and exacerbates "vicious circles"

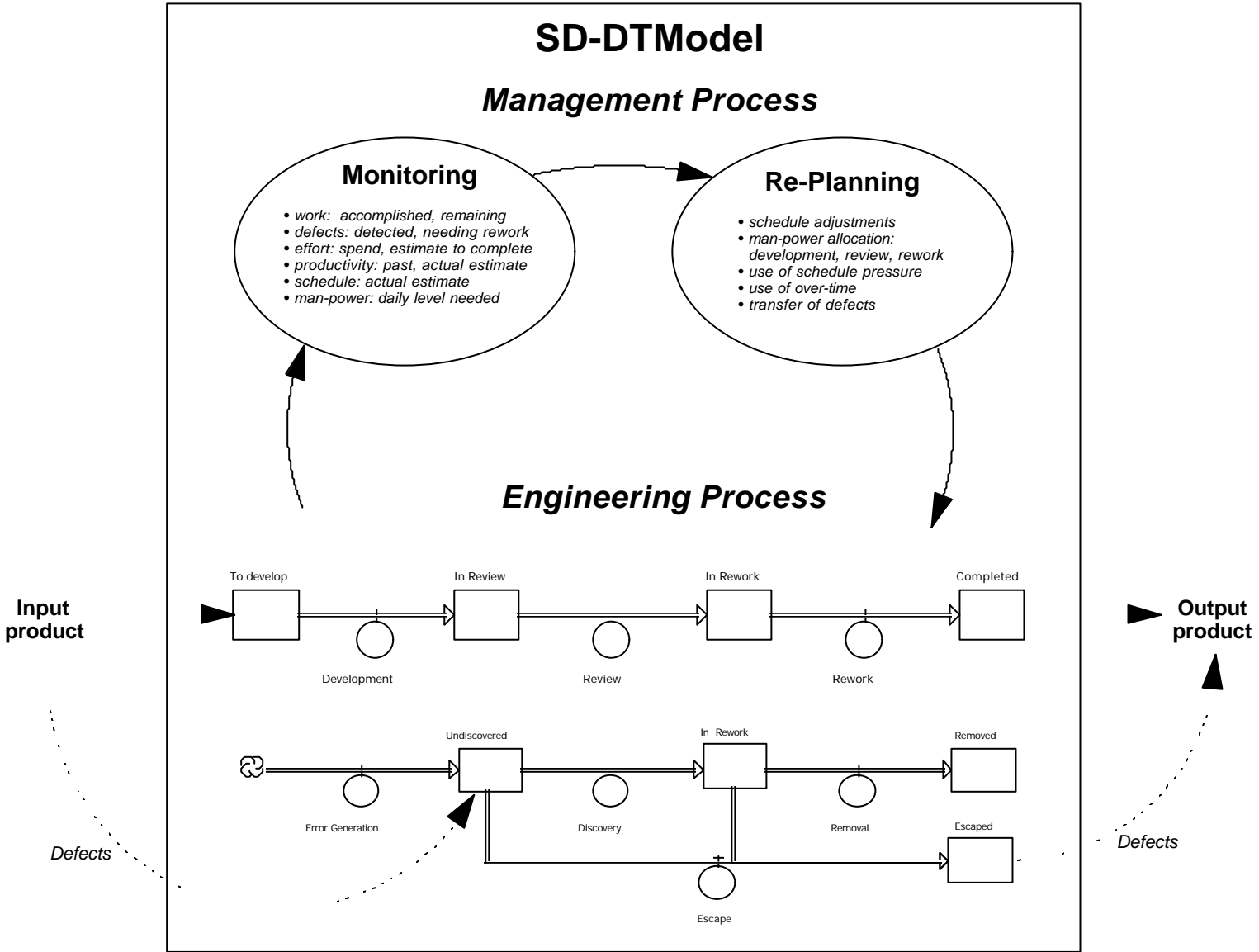
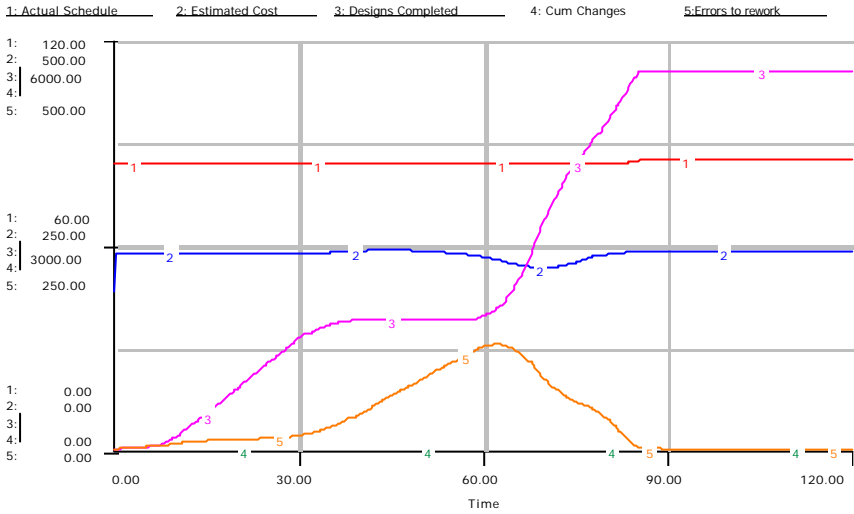
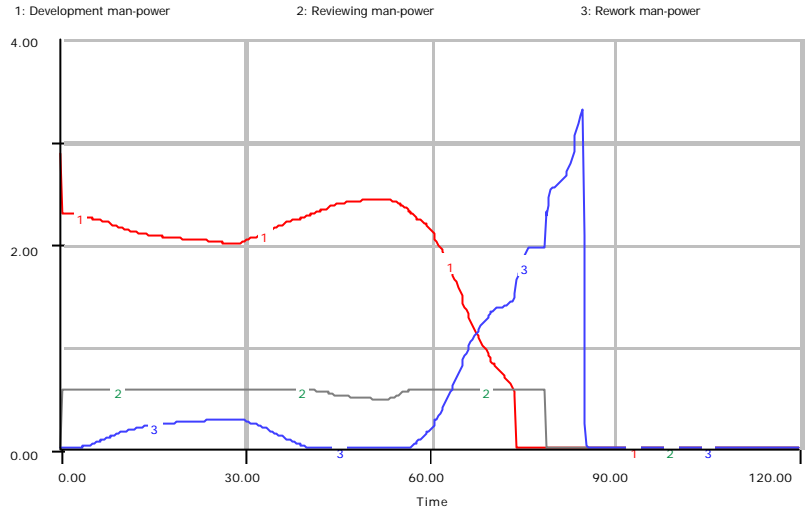


Figure 3 - The generic structure of the System Dynamics model of a development task



(a) General project behaviour (design phase)



(b) Man-power allocation

Figure 4 -- The base case: *steady* behaviour produced with the model calibrated according to the plans

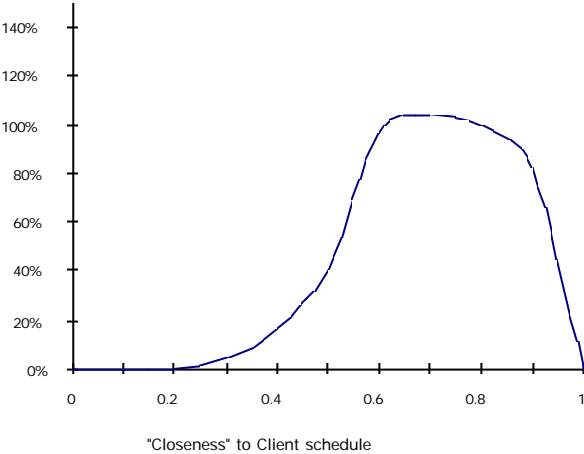
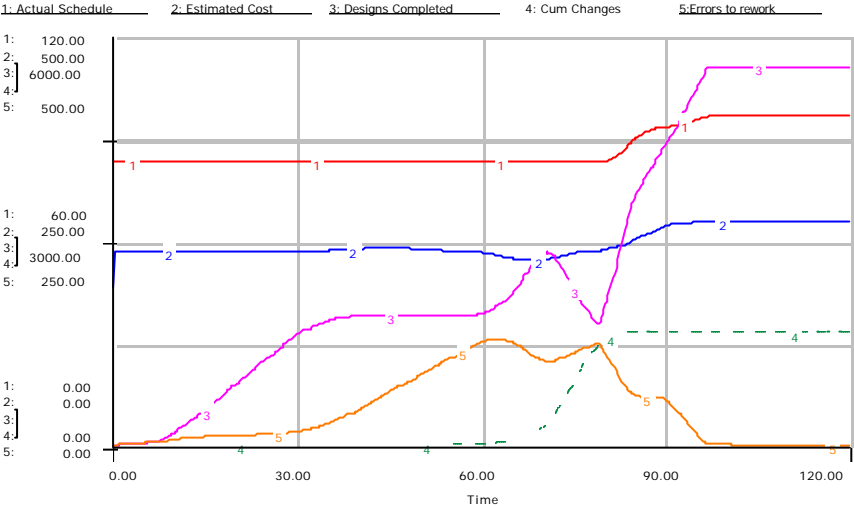
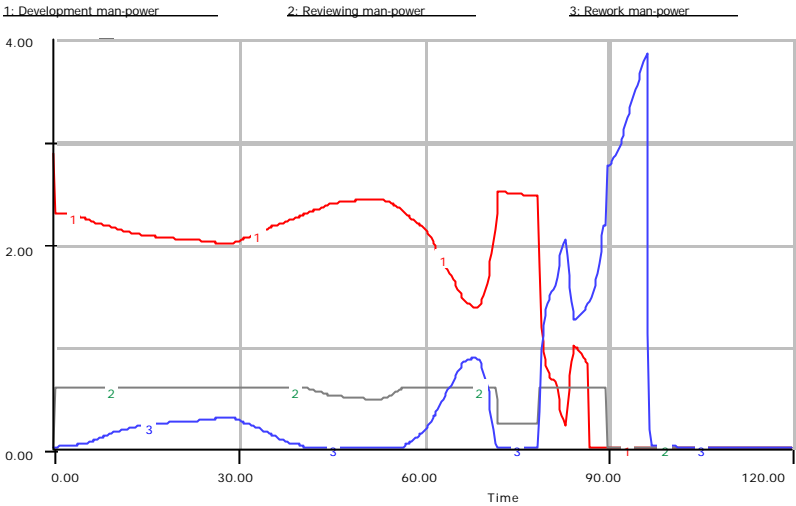


Figure 5 - Management policy: controlling schedule contingency



(a) General project behaviour (design phase)



(b) Man-power allocation

Figure 6 -- Unsteady behaviour produced when the model is calibrated with changes and no schedule adjustment

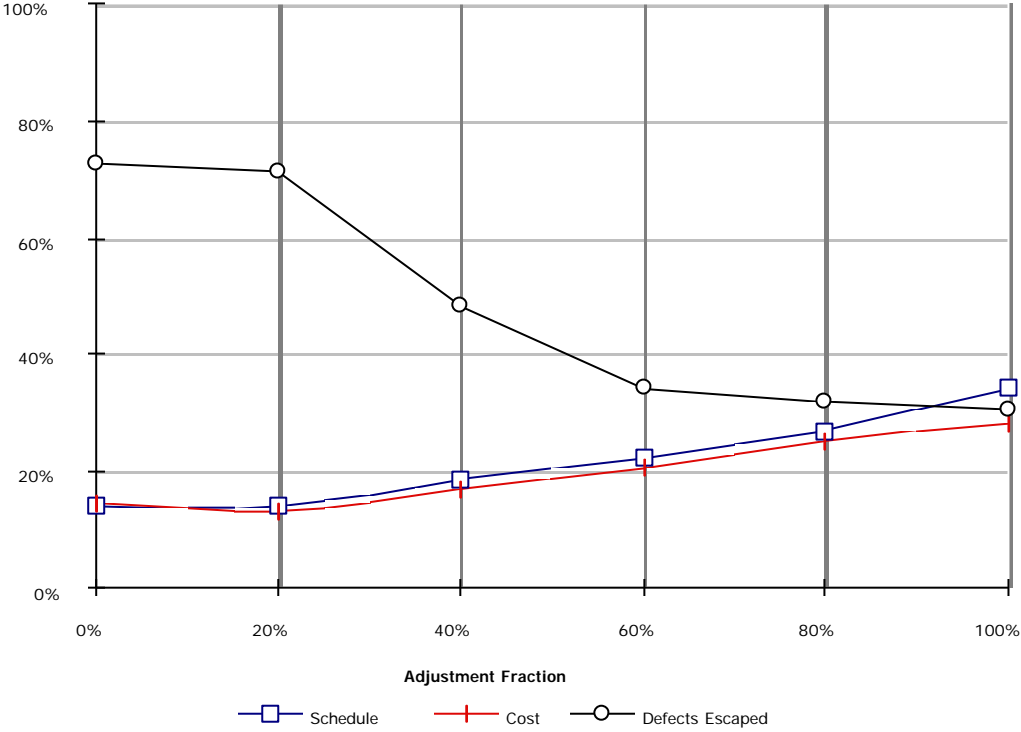
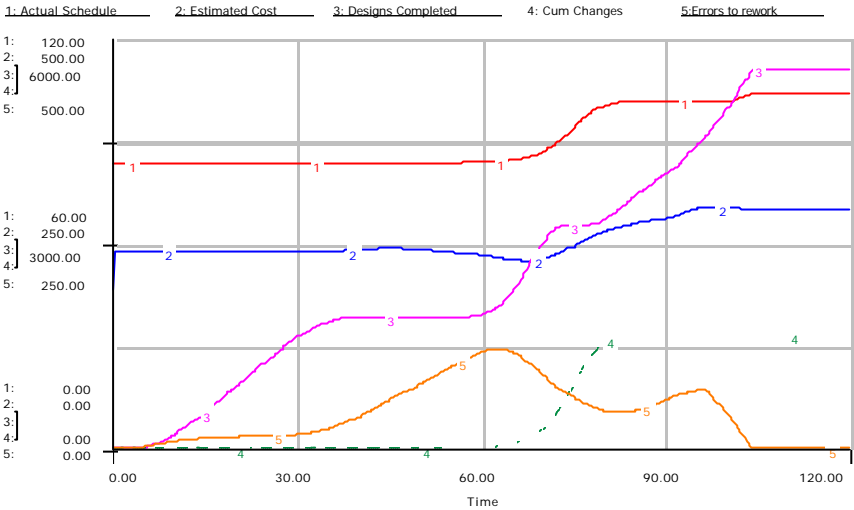
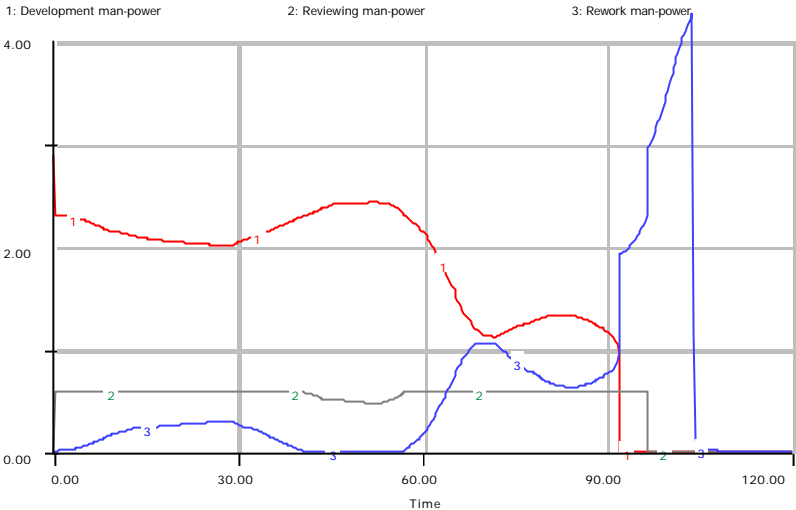


Figure 7 -- Impact on results of different levels of schedule adjustment



General project behaviour (design phase)



Man-power allocation

Figure 8 -- Behaviour produced when the model calibrated with changes and with a 60% schedule "adjustment fraction"