System Dynamics in Software Project Management: towards the development of a formal integrated framework

ALEXANDRE G. RODRIGUES and TERRY M. WILLIAMS
Department of Management Science, University of Strathclyde

Key words: software project management, system dynamics

Software projects are of major importance. Technical aspects have advanced, but not software project management: this is now the key area in which improvement is urgently needed. It comprises the functions responsible to keep the project within cost, quality and duration targets; interfacing with the Client and sub-contractors, and managing the interactions between planning, monitoring, technical development, QA and configuration management. In this paper we focus on three main functions of project control: estimating, planning, and progress monitoring.

Traditional techniques aim to provide operational support, aiding detailed decision-making. However, most software projects still fail to meet targets. Studies suggest that the main causes lie in strategic areas outside the concern of traditional techniques, e.g. political/social environment, legal agreements, and *human* factors. There has been little emphasis on strategy and it is now recognised that a *holistic* model of the stategic management of projects is needed. Traditional models are inappropriate for four main reasons: while trying to capture the project in detail, the resulting complexity hinders quick and reliable strategic analysis; they do not incorporate explicitly the influence of human factors; they do not consider the rework phenomenon, and finally they fail to capture the dynamic interactions between technical development and management policies. The specific characteristics of software projects exacerbate these: the main resource is *people*; the software product remains *intangible* during most of the life-cycle; and a rigorous definition of product requirements is usually very difficult until later stages. Staff productivity and work quality is continuously affected by several factors such as learning, schedule pressure, training and communications overheads. Management decision-taking can be affected by work pressure. The intangible nature of the software product, in particular its quality, encourages poor QA. High error detection rates are often seen as poor technical development, and because software quality is not tangible in the early stages, staff can skip QA effort. However a large amount of defects would require time-extensions, also perceived as poor management. This hinders organisations improving the quality of designs. Finally, unstable system requirements exacerbate the introduction of changes. The side-effects of these changes, in particular work being done out of its natural sequence, often create dramatic problems of rework. Improvements in traditional models have been aimed towards increasing complexity, and the above problems remain. Models often need detailed information to which managers cannot give practical meaning. With traditional techniques proving inadequate, managers revert to strategies that they *believe* have worked in the past. Management faults remain uncovered and organisations fail to learn effectively from one project to another. This problem requires a more formal systemic analysis for which traditional methods are unsuited.

System Dynamics (SD) is a powerful approach, based on a *holistic* perspective of managerial problems and focusing on the human aspects of a system's behaviour. There has been a number of applications to the process of software-development project management. Three are of particular importance; however, they indicate a need for improvement in the current models:

- The first application was by Abdel-Hamid and Madnick, and led to the development of a generic model for the software development process, based on the *post mortem* analysis of a NASA software project. They provide a good survey on the human interactions in a software

project, and some empirical validation for the relationships within the model. The model considers explicitly errors continuously escaping throughout the life-cycle; it also keeps within the principles of the life-cycle model. However, it aggregates the project work at the highest level, with no breakdown, so cannot analyse intermediate schedule milestones in detail, and cannot consider a staff profile for the project nor therefore the "natural" changes in work intensity. The model also assumes stable requirements.

- Lin and Levary elsewhere in NASA proposed other SD models. These use a work breakdown, allowing a more detailed analysis of schedules, budgets and staff allocation; however, it is restricted to classical life-cycle phases, whereas several dissimilar software components are often developed by different teams and integrated at different times. One model focuses on requirements changes being introduced throughout the life-cycle, using an expert system to support model application. Another model describes a software project as a dual life-cycle process of engineering and management. They propose a procedure to support empirical validation of the model. Evidence of support to major on-going projects would be desirable.
- Cooper (at Pugh-Roberts Associates) reports major practical applications. He introduced the important concepts of the *rework cycle* and *monitoring ramps*, assuming rework to be generated in the project remain undiscovered until the later stages. The consequent gap between perceived and real progress explains the "90% syndrome." His "PMMS" system provides a more flexible capture of the project work structure, based on generic "building blocks". The system is used by calibrating for a "problem-free" scenario, followed by "what-if" analyses. However, the models still take a high level view to support strategic management of large programs, which often include several projects in parallel. The models are not specialised to capture the many specific characteristics of software development. It is therefore unlikely that they are suitable to provide support at the lower tactical level.

The above are important contributions for applying SD to software project management. In particular, Cooper provides evidence about its practical credibility. However, most reported cases are post mortem analyses, or within fictitious scenarios to support policy analysis. A major step forward would be to apply SD to support on-going software projects (Cooper claims such an application). However, this requires SD to be embedded within the traditional project management framework, since project managers still need tactical operational tools.

Ideally, an integrated framework should consider the benefits of both traditional and SD approaches and establish information links between them. We have previously reported initial development of a project management integrated model (PMIM). We describe here the roles of SD models, how the models are used within the management process, and the characteristics required of the SD models in general and validation and calibration procedures. Our current research is improving and testing the PMIM within a large software project at BAeSema Ltd.

The PMIM uses SD models at two levels. At the strategic level there is a high-level SD model which covers the whole life-cycle and captures major development milestones, providing quick and preliminary assessment of major strategic decisions and risks before a detailed plan is produced. At the operational level, a more complex model captures individual life-cycle phases in more detail, providing quantitative data for work scheduling and resource allocation. Both are based on an appropriate breakdown of the project into major sub-tasks, consistent with the WBS. SD models within the PMIM are continuously calibrated throughout the life-cycle and used to support planning and monitoring. In planning the model is used to assess the performance of the current plan and identify risks. Once a detailed plan has been produced into a logical network the dynamic characteristics of the *planned behaviour* are extracted. Calibrating the SD model to reproduce this *steady* behaviour requires explicit definition of several metrics, which otherwise

would remain "hidden" in the plans. The quantitative relationships within the model must be able to produce the planned behaviour, and this uncovering of metrics is useful as it helps to identify unrealistic assumptions and suggest adjustments to the plan. The model is then used to assess the plan's sensitivity to risks through the analysis of "what-if" scenarios which produce possible *unsteady behaviours*. Secondly, in monitoring, the model is used to diagnose past behaviour and help to identify the causes of deviations. Once traditional progress monitoring data has been collected, the dynamic characteristics of the project *past behaviour* are derived. As in planning, calibrating the model to reproduce past behaviour uncovers result metrics. Where numbers are not realistic the conclusion might be that progress data is not reliable. This calibration also provides an estimate of the number of defects that escape undiscovered, helping to avoid over-optimism about progress. After the progress data has been revised, the model can be used to investigate the causes for eventual deviations and to test whether alternative planning and control policies could have provided better results. This calibration reproduces not only the final engineering result, but also the management decision-making pattern (i.e. perceptions of progress and subsequent re-planning). Each time the model is re-calibrated to reproduce and diagnose segments of the past it provides new estimates for future behaviour. The PMIM uses the models at both levels in the process in the way described, but at different levels of detail.

The structure of an SD project model should capture the basic characteristics of the software development process. We have four basic principles. Firstly, there is a dual life-cycle view of engineering (e.g. life-cycle phases, product components, rework, work dependencies, reviewing techniques) and management (e.g. monitoring and managerial policies for re-planning work schedules and allocating resources). Secondly, the project is broken down into major sub-tasks. Each task will be simulated by an individual SD sub-model whose structure is specialised on the type of work; for the engineering process we propose sub-models for two main categories of tasks: development (of a particular sub-product, including developing, reviewing and reworking) and testing (running pre-defined tests on a sub-product, diagnosing faults, and reworking defects). The breakdown of the engineering process is based on the life-cycle definition, the product structure, and the technical team structure. We aggregate low-level sub-tasks into more complex tasks, aggregating sequential tasks horizontally and parallel tasks vertically. Considering the appropriate level of aggregation of the engineering process: (i) the type of work within a task should be of a similar nature and strongly inter-related; (ii) the major schedule milestones should limit the level of horizontal aggregation; (iii) it must be sufficiently detailed for management purposes; (iv) each task must have a minimum level of complexity so that the use of an SD model is appropriate (SD focuses on the interactions among a system's components; over-decomposition eliminates the effects of these interactions, and behaviour becomes characterised by discrete uncertain events, so continuous SD is inappropriate). Having decomposed the engineering process into tasks these can be linked by precedence relationships, defining a network of tasks to which a network of SD models will correspond (with sequential tasks overlapping depending on the specific characteristics of the development process and on management decisions). The third principle for the model structure is to consider *explicitly* that the engineering process comprises the continuous flow of two inter-related entities: work, and defects. As the work is developed, reviewed/tested and reworked throughout the life-cycle, defects are also generated, detected, and reworked. Some, however, are not detected by the review and testing activities and hence *escape*, into the next task. Finally, the fourth principle is the use of a high-level management sub-component which overviews the whole project. Based on progress and estimated completion information from the SD network, this mimics the high-level decision-making in the project (including adjusting completion schedules of individual tasks, re-scheduling staff among tasks, increasing the degree of concurrency and hiring/firing staff). The complete global model for a software project is now defined by linking all sub-

models. The conceptual structure for the global model developed for our current case-study is shown in the full paper.

The structure of the SD model is based on the breakdown of the project work, as are traditional models, e.g. the WBS. Therefore, a formal link can be established: to each engineering task of the SD network corresponds a specific set of WBS work packages. The relationship should be defined by a matrix identifying which WBS packages correspond to each SD task, and the proportion of effort, although the relationship is stronger if most packages are unique to a certain SD task. The technical development is implemented by sub-teams specified in the Organisation Breakdown Structure (OBS). These teams are usually specialised in a certain development activity which is performed continuously throughout the life-cycle. The relationship between the OBS and the SD network should also be specified in a matrix identifying which teams work in each task; this is important to support the translation of staff levels between the SD model and the logical network. The structure of the SD model should also relate to the information on life-cycle phases/stages by a clear identification of the life-cycle phase to which the SD task belongs. The use of SD within the PMIM assumes the exchange of qualitative and quantitative information with the traditional models (including estimating models, e.g. COCOMO, and network-based planning/monitoring models, e.g. PERT, PERT/Cost and earned value).

As the project progresses the network plan is continuously updated to incorporate the results of the technical development process (effort expenditures, schedules, staff levels) and the management process (schedule and manpower re-planning decisions and progress estimations). When the SD model is calibrated, it must reproduce acceptably not only the results of the engineering process, as recorded in the network, but also the management process decision-making. Calibrating the latter is more difficult since re-planning decisions often are not recorded in the (static) project information system. Re-planning often implies changing the network structure typically by increasing concurrency, so the unsteady past behaviour may be characterised by a continuous evolution of the network structure. Where accurate data about the management behaviour is not available, managers' judgement is essential to validate the calibration.

The plan is also updated to incorporate future plans; the future behaviour portrayed by a network assumes a "planned success", assuming either no deviations or successful management corrective actions. When the SD model is calibrated to reproduce this *steady* behaviour it must reproduce the expected results from the engineering process, but there is again usually no information available about the likely management decision-making pattern. As a project is perceived complex the tendency is to adopt a reactive attitude where deviations and management reactions are not anticipated. Once the future behaviour is assumed steady, the SD model must reproduce a behaviour with few deviations and management actions quickly solving problems.

Quantitative links are at the core of a good integration of SD models within the traditional approach, as in the PMIM: both models produce the same quantitative results for past behaviour and for the planned future behaviour; preliminary estimates produced by traditional models, like COCOMO, may also be tested in the SD model and if successful translated into the network plan; monitoring tools provide the necessary information to derive the project past behaviour.

We consider that this integrated tool provides a powerful support to the management of on-going major software-engineering projects. This has been proven by use during an actual project.

Acknowledgement -- This work has been funded by JNICT--Comissão Permanente INVOTAN/NATO and Programa PRAXIS XXI, Portugal; and supported by BAeSEMA Ltd., United Kingdom.