Modeling Rework Cycle: Comparing Alternative Formulations

Hazhir Rahmandad, Kun Hu, Thanujan Ratnarajah
Virginia Polytechnic Institute and State University, Industrial and Systems Engineering Department
Room 430, Northern Virginia Center, 7054 Haycock Road, Falls Church, VA 22043
Tel: +1 (703) 538-8434
hazhir@vt.edu, hukun@vt.edu, thanujan@vt.edu

Abstract

Rework cycle is at the heart of modeling projects, one of the major application areas of system dynamics. In this paper we introduce a new formulation for rework cycle in which multiple defects may exist in a task. We compare the performance of this model with three other formulations, two adopted from system dynamics literature and one agent-based formulation. This comparative study illustrates the impact of underlying assumptions about the nature of defects and homogeneity of tasks in different formulations on model behaviors and provides information needed for effective formulation selection decisions. The new formulation we introduce allows for capturing significant schedule over-runs due to a few tasks, with multiple defects, that may go through rework cycle multiple times. Its perfect mixing assumption, however, limits its precision in terms of simulating the final project quality. Sensitivity analysis informs the robustness of results to multiple projects parameters. We discuss the implications for selecting robust formulations in modeling project dynamics.

Keywords: *project dynamics, rework cycle, comparison, formulation, agent-based*

1- Introduction and motivation

Projects are central to how work is done in organizations. From construction to software development, systems engineering, and product development, projects are at the heart of organization of work in modern societies. Despite their diversity, projects have in common a set of pre-defined goals that are to be achieved by completion of a set of tasks within a defined budget and schedule and with a given set of resources. Project performance is often measured on multiple dimensions including time, costs, and quality. Performance is realized as a result of evolution of project through its life time, thus it is a dynamic concept. The vast variation in quality, timeliness, and cost performance of projects across different fields has provided ample opportunities for consultants and academics to study dynamics of projects. In fact, project dynamics has been one of the core areas of study in system dynamics which has produced much academic research and ample consulting fees. Lyneis and Ford provide a comprehensive review of the literature on applications of system dynamics to project management (Lyneis and Ford 2007).

Majority of system dynamics studies that focus on project dynamics include a simulation model of project evolution. These models vary in their level of complexity and the feedback effects they capture. However, a core feature of all these models, building on the ground breaking consulting project by Pugh Roberts Associates in the 70s, is the rework cycle (Cooper 1980). The basic insight in rework cycle formulation is the realization that tasks that are completed as part of a project may be flawed and may need rework. Given the widespread application of system dynamics in project modeling, the basic rework cycle idea is among the

most widely used formulation concepts in the field. In fact, multiple alternative formulations have been used to capture the rework cycle with different levels of complexity and different conceptions of defects. Despite widespread use, very little is written about comparative advantages and disadvantages of these formulations. Such understanding is needed for practitioners and researchers who seek to use the best formulation at hand for the application at hand. Moreover, a comparative study will inform the mapping of different parameters from one formulation to another, and thus enables better accumulation of knowledge about typical model parameterizations. This paper is an attempt to fill this gap by providing a comparison of multiple formulations of rework cycle with different levels of complexity and detail. We also introduce a new formulation of rework cycle that we think strikes a good balance between complexity and precision for many applications.

In the next section (1-1) we introduce the basic idea of rework cycle in more detail and establish it in the context of modeling project dynamics. Section 2 discussed the study design and the alternative formulations we are comparing in this paper, including a new formulation for rework cycle. We introduce multiple model formulations in Section 3 followed by the analysis of results in Section 4. Finally conclusions and implications are discussed in Section 5.

1-1- Rework Cycle at the Heart of Project Dynamics

Rework cycle may be the most important feature of system dynamics project models (Lyneis and Ford 2007). It recognizes that completion of a project task may be flawed, resulting in a need for rework. Rework can itself be flawed, requiring further rework in a recursive cycle that can extend project duration and work load far beyond what is originally conceived. Thus rework cycle captures some of the main mechanisms leading to projects going overtime and budget and accumulating flaws.

Figure 1 provides a very simple conception of rework cycle, adopted from one of the early models of projects (Chapter 4, Richardson and Pugh 1981). Here tasks, upon completion, may flow into "Tasks Approved" (if they are correctly done and accepted), or "Undiscovered Changes", those tasks that have to be reworked, but not yet recognized as such. The completion of tasks depends on available resources and their productivity, while quality of the work is captured by defect rate. The schematic below is chosen to represent the simplest conception of rework cycle which includes only two independent stocks. However, even this very simple model can be used as the building block for capturing much more complex project dynamics. For example effects of schedule pressure, morale, communication congestion, overtime, and experience on error (defect) rate and productivity can be directly applied here (Cooper 1980; Sengupta and Abdelhamid 1993). The changes in resource availability and speed of rework discovery can also be easily accommodated. Multiple simple blocks, with similar structures, can be connected in cascades that represent multiple phases of bigger projects and different feedback effects between up-stream and down-stream phases (Repenning 2000). Other modelers have expanded this basic formulation to include an explicit testing procedure (AbdelHamid and Madnick 1991), and to keep track of changes (or defects) in a separate stock and flow structures (Ford and Sterman 1998).

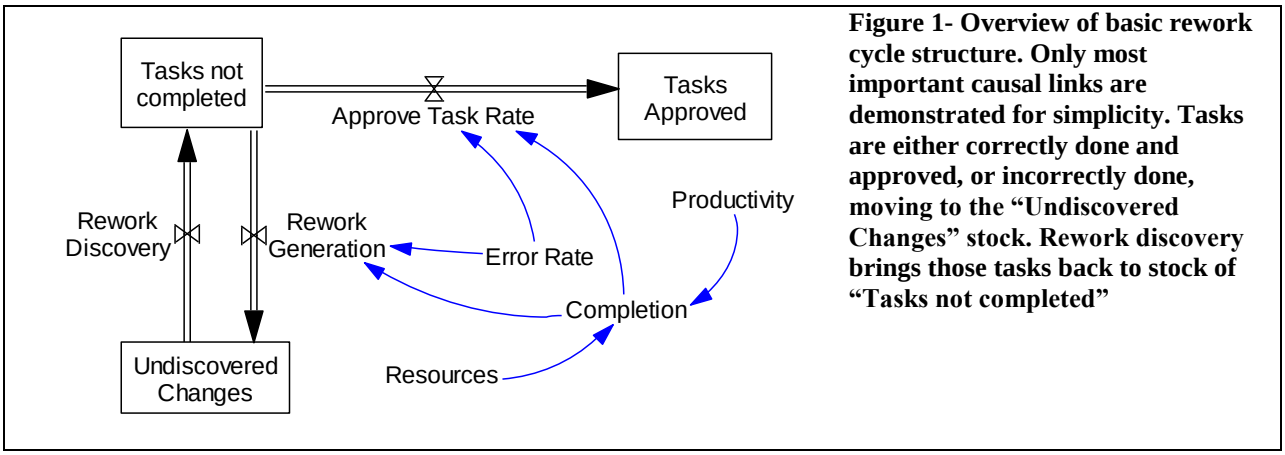


Figure 1- Overview of basic rework cycle structure. Only most important causal links are demonstrated for simplicity. Tasks are either correctly done and approved, or incorrectly done, moving to the "Undiscovered Changes" stock. Rework discovery brings those tasks back to stock of "Tasks not completed"

In fact this framework can be used in exceedingly detailed levels, e.g. to represent small homogeneous chunks of a project. In the extreme, one can envision a detailed agent-based model of project in which separate entities represent each task, as well as each unit of resource (typically individuals who will be assigned to tasks, but can also include machinery, money etc) (Levitt, Thomsen, Christiansen et al. 1999). In such formulation each task can go through multiple states, e.g. not worked on, worked on and pending testing, approved, or pending rework. Moreover each task entity will include properties such as number of defects that have accumulated in the process for working on that task. Growing computational power and software availability have encouraged the application of disaggregated models. Detailed representation improves external validity of the model (likeness to the real world) and thus potential precision of the results. However, the additional disaggregation often also complicates a comprehensive understanding of what model structures and processes are generating the observed behavior; model development and bug fixing is harder in these models; and communication of the insights to real-world decision-makers may prove more demanding. Moreover, typically detailed agent based models are stochastic in nature and require more computational resources than simpler differential equation representations, both to get a reliable view of the range of stochastic results they generate, as well as for conducting a single simulation. These computational costs can lead to reduced sensitivity analysis, longer turn around time for analysis, and smaller model boundary (i.e. the range of feedback effects and sub-systems included in the analysis) (Rahmandad and Sterman 2008).

In the context of these tradeoffs, two questions beg for an answer: 1) What kind of tradeoffs do we face in terms of precision vs. complexity, when we choose different formulations for rework cycle? 2) How do different formulations connect to each other? For example, if task level data suggests that 20% of tasks have at least one defect in their first implementation, should we use a defect rate of 0.2, or another number is more appropriate, in one specific formulation. The current study is an attempt to address these questions by comparing four different formulations for the rework cycle, with exceedingly higher levels of details. Our analysis sheds light on the relationships between different formulations as well as the tradeoffs observed in terms of performance, precision, and flexibility.

2- Study design

In this study we want to compare and analyze alternative formulations for the core rework cycle structure common to most models of project management. For this purpose we focus on analyzing a very simple project that consists of a pre-determined number of tasks. Tasks are completed by fixed resources with constant productivity. Completed tasks are then tested, and accepted or sent for rework. Rework is conducted by the same people who also do the initial work. Testing is only limited by the time it takes to conduct the tests (no resource bottleneck for testing phase). We keep the study manageable by including no feedback on productivity, quality of completion and rework, testing quality, amount of resources, or scheduled finish date. These feedbacks are central to important project dynamics however; therefore we conduct sensitivity analysis to test the robustness of results with respect to different parameter values.

Four different model formulations with different levels of detail are used to simulate the simple project. These models are built in parallel so that similar concepts, variables, and parameters are comparable across the models. On one extreme, the very simple model is used (See Section 3). This model uses only two independent stocks to capture the rework cycle. It does not distinguish testing from work completion, and the concept of defect is embedded in the definition of undiscovered changes. This formulation is adopted from Richardson and Pugh's "Introduction to System Dynamics Modeling" (1981), and modified for the study at hand. The second rework cycle structure is developed based on the work of Ford and Sterman (1998). In this formulation changes and tasks are explicitly differentiated through a co-flow structure (Sterman 2000) where tasks requiring change flow in a parallel structure to the tasks. We introduce a third formulation here, which adapts the second formulation to include the number of defects in the tasks, as the coflow, rather than the number of tasks which require a change (Rahmandad 2005). This adaptation recognizes that many tasks may include more than one defect that requires fixing during rework. Explicit testing formulations are developed here accordingly. Finally, an agent-based model with explicit task entities and development resources is used as the most detailed formulation. This latter formulation is much more complex than the others and creates stochastic trajectories of project completion (where as the other formulations are deterministic). In fact we use this formulation as the simulated "real-world" and see how the different deterministic formulations closely replicated the behavior of this model.

Parallel model development is achieved by starting from the agent-based model, and deriving the parameters for the successively more aggregate models from this version. Through the parallel model specification process we can see the relationship between parameter values at different levels of aggregation, and thus answer one of the research questions that motivate this study: How do different formulations connect to each other?

We address the other question through comparing the different models on different dimensions of costs and benefits of disaggregation. On the costs of additional detail we observe model complexity in terms of the number of stock variables in each formulation. We measure the benefits of disaggregation by comparing the flexibility and performance of different formulations. Flexibility measures the different parameter settings and assumptions about the nature of the work that can be changed in each formulation. It therefore informs the range of applications that a formulation can support. Performance metrics compare the behavior of the four alternative models on the quality and schedule dimension, as well as the difference between different formulations in terms of project evolution. Specifically, the "Finish Time" for the project is reported for different formulations (including both mean and standard deviation for the agent-based formulation). Defect rate in completed work is reported for the three more complex

models (the concept is not captured in the simplest formulation). Finally, average absolute error compares the four formulations head-to-head in terms of the fraction of tasks approved at different point of time in the project, and reports their difference. Figure 2 gives an overview of two of the performance metrics.

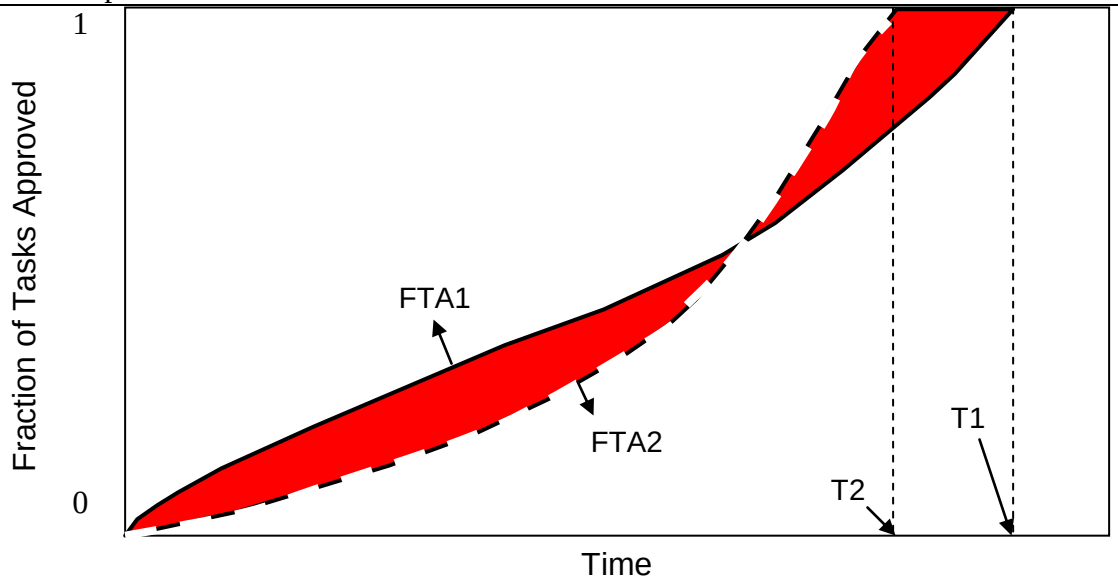


Figure 2- Finish time and average absolute error metrics. The results of hypothetical simulations with two alternative formulations (M1 in solid line and M2 in dashed line) are reported for a project. The two lines represent Fraction of Tasks Approved for M1 and M2 at different points in time (*FTA1* and *FTA2*). The Finish Times for two formulations (T1 and T2) are reported in months. The average absolute error between M1 and M2 (AAE12) is calculated as:

$$AAE12=AAE21=\left(\int_{0}^{Max(T1,T2)} |FTA1-FTA2|\right)/Max(T1,T2)$$

This metric is therefore normalized for project size and duration, thus allowing for comparison across different projects.

After describing the model formulations and deriving the parameter values from the most detailed model (Section 3) we present the base case results where the performances of the four models are compared using different metrics discussed above (Section 4). Next we will discuss the sensitivity of the results to different parameter values including alternative defect rates, project sizes, testing accuracy and speed, and different probability distributions for the stochastic parameters of the agent-based model (Section 4-2). The conclusions and implications for modeling project dynamics are provided in Section 5.

3- Model Formulations

The four different models are introduced in this section. We name these models according to their level of complexity, so the simplest model is "Model 1" or M1, etc. We first briefly describe each model and then discuss the derivation of the parameters of the more aggregate models based on the agent-based model (M4).

Model 1- This formulation is adopted from an early model of project dynamics (Richardson and Pugh 1981) because it includes the minimum number of independent stocks required to capture the core idea of the rework cycle. Figure 3 presents an overview of the stock and flow structure for this model. Basically the tasks are completed based on current Work Staff and Productivity. The completion rate is also bounded by minimum time to complete a task. A fraction of tasks are approved (either because they are done correctly, or because the quality assurance and testing process have not flagged them as requiring change). The rest of the tasks are sent to the "Undiscovered changes M1" where they remain until they are discovered as in need of rework, and thus sent back to the stock of tasks not completed.

It is important to note that testing process is not explicitly captured in this model. Therefore the probability of generating changes relates both to the defect generation and the quality of testing. Even with high real defect rates, a lousy testing procedure can reduce the probability of generating changes because most those defects are not captured. Moreover, the formulation does not distinguish between rework and initial work because it sends all the discovered rework into the tasks not completed. This assumption implies that M1 formulation does not have enough degrees of freedom to capture significant differences in the nature of work between first time work and rework. These considerations therefore require us to derive the parameters for "Probability of generating changes" and "Time to Discover Rework M1" from the parameter values in the more detailed models. Finally, this model does not keep track of potential problematic tasks that are approved, and therefore the quality of the final project. We therefore can not report one of our performance metrics, average defect in approved tasks, for this model. Standard and straight forward formulations are used in model one and complete model is available in the online appendix along with the other formulations.

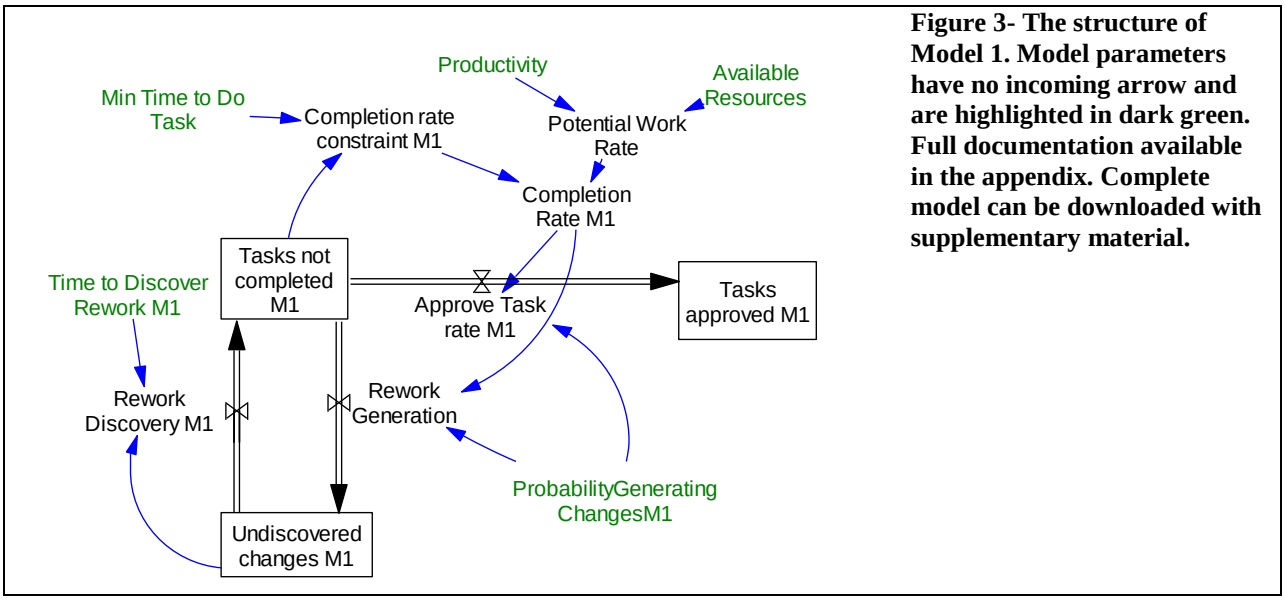


Figure 3- The structure of Model 1. Model parameters have no incoming arrow and are highlighted in dark green. Full documentation available in the appendix. Complete model can be downloaded with supplementary material.

Model 2- Ford and Sterman (Ford and Sterman 1998) introduced a more comprehensive project formulation for the rework cycle. While their paper focuses on feedbacks across multiple stages of a product development process, their model also included a co-flow structure to track defective tasks along with the tasks in the core rework cycle formulation. We adapted their

rework cycle formulation for the study at hand, calling it model 2 or M2. Figure 4 provides and overview of this model. Full model documentation is available from the original paper and in the online model posted with the paper.

This formulation includes 6 independent stock variables (the task stock and flow structure is closed and thus the sum of all stocks remain constant, reducing the number of independent stocks by one). Available resources are allocated between "Completion rate M2" and "Change task rate M2" proportional to the desired completion rates. Desired rates depend on stocks of "Tasks not completed" and "Known changes", as well as time to do the tasks or rework. "Probability of generating defect during completion" determines what fraction of completed tasks potentially requires a change. Not all such tasks are however discovered through the testing process. "Probability of missing a defect" informs what fraction of problematic tasks is discovered and sent to "Known changes" stock and what fraction flows into "Tasks approved" along with their "Changes approved." For tasks with known changes, rework is not always successful in removing the problem. While some of the problems are fixed in the rework process, some defects remain and flow back into "Undiscovered changes."

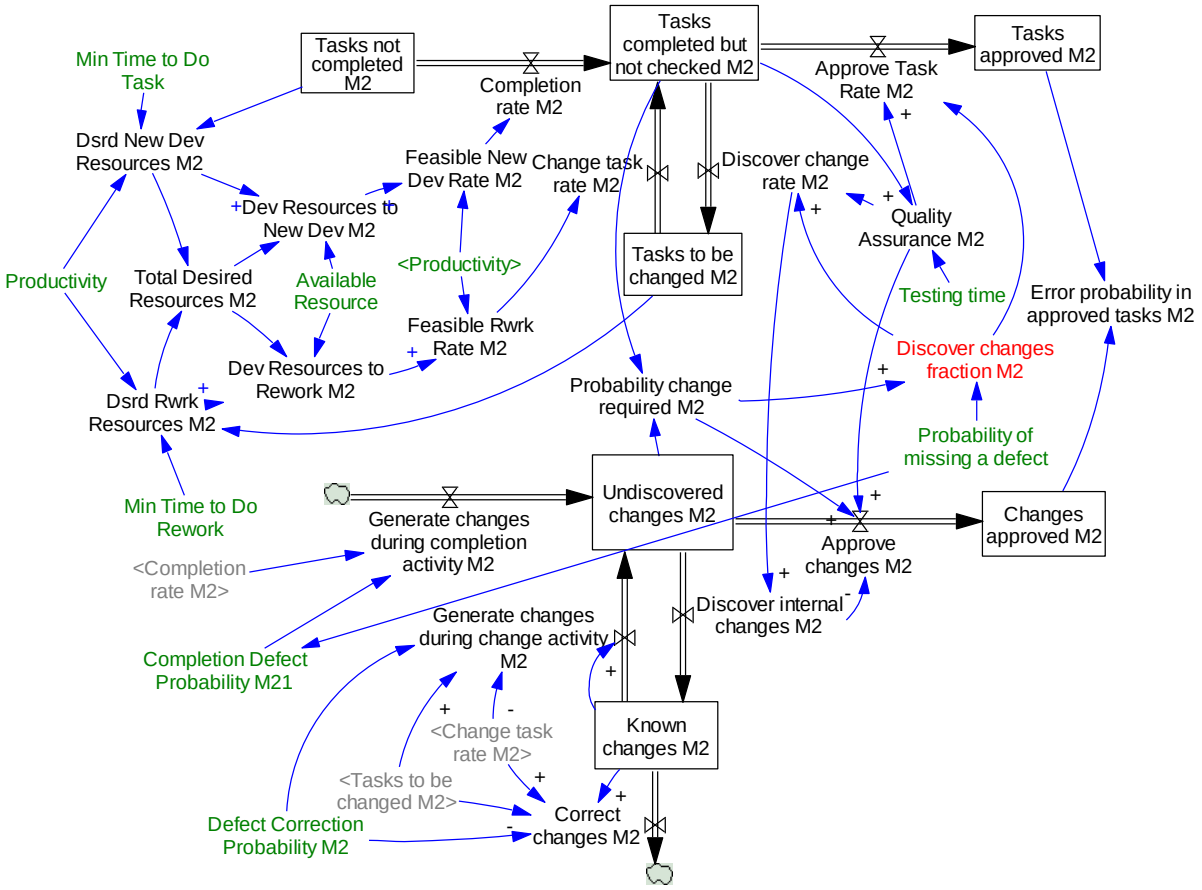


Figure 4- Complete structure of Model 2, adopted from Ford and Sterman, 1998. Model parameters have no incoming arrow and are highlighted in dark green. One variable ("Discover Change Fraction M2") is highlighted in Red and discussed in more detail in the text (under Parameter Relationships section) because it diverges from the original formulation by Ford and Sterman. Full documentation is available in the appendix.

This formulation allows the modeler to account for problematic tasks and their required changes. It therefore enables tracking of defects in approved tasks and therefore the quality of the finished project. Nevertheless, the conception of defect (or change) in this model is binary: a task is either defective, thus in need of change, or correct. The formulation therefore does not account for different levels of defectiveness, e.g. having multiple defects per task. Therefore parameters related to defect generation and correction probabilities (e.g. "Defect correction probability" and "Probability of generating defect during completion") should be calculated based on the detailed defect generation process in more detailed models.

Model 3- The third formulation we introduce builds on model 2 and allows for multiple defects per task. The basic stock and flow structure closely follow that of Model 2. Moreover, we explicitly introduce a stock and flow structure for testing. This additional complexity is required to consistently account for the testing process in presence of multiple potential defects per task. The current formulation was first used to model software development processes (Rahmandad 2005). This paper introduces the formulation in the literature for the first time and compares it with other alternatives.

A good rework cycle formulation with co-flow for defects/changes (rather than tasks that have a defect or need change, e.g. Model 2) should overcome some relatively complex modeling issues. Specifically, the perfect mixing assumption of differential equation models means we can only know about the average number of defects per task, but not the distribution of number of defects. However, to decide what fraction of tasks are approved in the testing process and what fraction are sent for rework, we need to know what fraction of tasks are defect-free. Finding that fraction requires some additional distribution assumptions as for how the defects are distributed among the tasks. Furthermore, if we consider the imperfections of testing process, we should take into account that not only defect-free tasks, but also some of the tasks with one, two, or more defects could be, by chance, accepted. Therefore a successful formulation should be able to determine what the defect density is in both approved and rejected tasks.

Moreover, the testing process often includes tests that cover more than a single task. For example consider integration testing in software development (vs. unit testing). Where unit testing makes sure a specific piece of code (a small module or function) is working properly given a set of inputs, the integration testing includes examination of interactions between multiple modules, thus increasing the chance that a test fails because any of the components have a defect. These larger tests could in fact be rejected more easily because a failure in one of many components leads to sending all those components for further rework.

Figure 5 presents the conceptual framework we use to tackle this formulation problem. Here we represent the tasks as the area of the rectangle in the figure: the larger the number of tasks, the bigger the rectangle. The number of tasks at each stage can be followed in the respective stock variable (See Figure 6 for a partial stock and flow diagram for Model 3). Defects are the red dots distributed in this area. We can track the number of these defects in a stock variable for "Undiscovered changes."[1] Perfect mixing nature of material in a stock means these defects are randomly distributed in the area. That is, the location of each defect is completely independent of the other defects, and all points in the area have the same chance of including a defect. Finally, a test is represented as a larger oval that covers some area (a number of tasks). A test is then rejected if at least one of the defects inside the test area are recognized in

[1] We use defect and change interchangeably for Model 3. While defect is conceptually more appropriate in this context, we want to keep the parallelism with the Model 2 and therefore use similar names for the stock and flow variables as used in Model 2.

the testing process. Otherwise the test is approved, accepting some part of the tasks (according to the test area) into the stock of "Tasks approved" (See Figure 6). The defects/changes that have been missed in a test flow in parallel into the "Changes approved" stock.

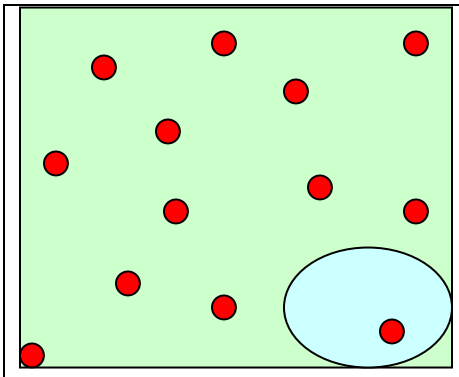


Figure 5- The conceptual framework for modeling multiple defects per task, used in Model 3. Here the area of the colored rectangle represents the tasks in one stock, the red dots represent the defects randomly distributed in this space, and the blue oval is the representation of a test which covers some tasks and potentially some defects. This conceptualization allows us to calculate the probabilities related to the number of errors in each area, e.g. the probability of rejection of a test and the expected number of errors per approved test.

The perfect mixing requirement proves to be very useful in this setting. The independence of placement of defects in the task area results in the Poisson distribution of the number of defects per any area (e.g. the area covered by a test). This is due to the fact that our setting satisfies the Poisson distribution's underlying assumptions: that probability of observing a defect over an area does not change over different areas, and that probability of observing a defect over an area is independent of the observation of defects in the rest of the space. We can therefore find the probability of finding k defects in a test area of size a (unit: task), where the defect density is d (unit: defect/task) based on the Poisson probability distribution:

$$P(\#errors = k) = \frac{e^{-da}(da)^k}{k!} \qquad \textbf{(1)}$$

For example, the probability that no defect is present in the test area, and thus the test is accepted (releasing the tasks in test area into the stock of approved tasks) is: e^{-da}.

Furthermore, we can calculate the impact of imperfect testing. A test is passed under this setting if no defects are present in the test area, or a single defect is present but that is missed with probability ε, or two defects are present and both are missed (which assuming independence between different defects will have a probability of ε^2), and so on. Therefore the probability of a test with area a passing can be calculated as:

$$P(TestPass) = \sum_{k=0}^{\infty} \frac{e^{-da}(da)^k}{k!}\varepsilon^k = e^{-da(1-\varepsilon)} \qquad (2)$$

The expected number of defects in a test that has passed is another variable we need to calculate in order to determine the flow of defects from stock of "Undiscovered changes" to stocks of "Changes approved" and "Known changes". This expected value can also be calculated:

$$E(DefectPerPassedTest) = \sum_{k=0}^{\infty} k\frac{e^{-da}(da)^k}{k!}\varepsilon^k = da\varepsilon e^{-da(1-\varepsilon)} \qquad (3)$$

Finally, this conceptualization allows us to consider lower-than-complete test coverage. Testing can be imperfect not only because we may miss some defect in the task area that is being tested, but also because we may not test all the tasks in the system. Under low test coverage, a part of task area is released without testing, along with the area covered by each test. The defects in the untested area are also released in the approved tasks with the average defect density for the

task area in general. In order to consider such imperfect testing scenarios, it is helpful to introduce explicit stocks and flows for the tests that are being run[2].

In Model 3 presented in Figure 6 we use "Discover changes fraction" to measure what fraction of tests fail. This is one minus the value found according to equation 2. We calculate the number of tasks that are sent to be changed using this fraction and the total amount of tasks covered by testing ("Quality assurance"). On the other hand, the tasks approved are potentially more, if test coverage is lower than 100%. Therefore we calculate the "Approve task rate" by subtracting the rejected tasks from the tasks that are either tested or approved without test. The number of these tasks is calculated by using the fraction of total tests that are conducted at any point in time ("Frctn Tst Running") and extending that fraction to all the tasks pending testing ("Totl Task to Tst"). For example if testing rate is conducted at 0.15 of tests per month, then 0.15 of tasks to be tested are flowing monthly out of the stock of "Tasks completed but not checked", out of which some are going to "Tasks to be changed" and the rest are approved.

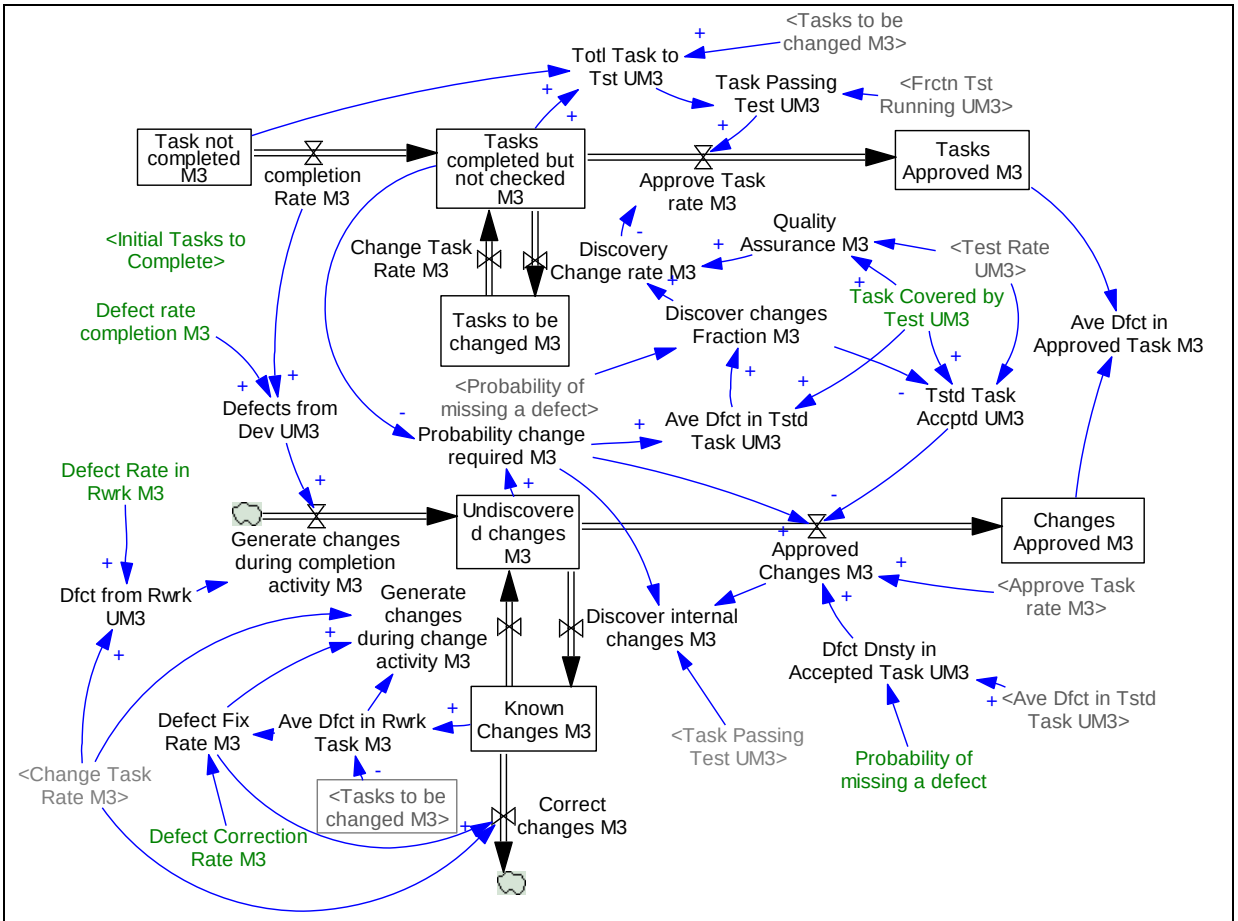


Figure 6- The structure of Model 3. For simplicity we do not show the variables determining "completion Rate M3" and "Change Task Rate M3". These concepts are formulated exactly as in Model 2. Moreover, the testing sector is shown in the next Figure and includes the specification of shadow variables in this view (highlighted by <> signs). Model parameters have no incoming arrow and are highlighted in dark green. Full documentation available in the appendix.

[2] While the additional stocks are not required if the test coverage is unchanged throughout a simulation, we discuss the additional structure for increased clarity and flexibility of formulation to be applied to dynamic test coverage.

We formulate the rate "Approved Changes" (see Figure 6) in the parallel defect co-flow by calculating the defect density in the tasks being accepted (which depends on the defect density of tasks tested and those approved without testing, properly weighted) and multiplying that by the "Approve Task Rate". Once the rate for approval of defects is determined, it is easy to calculate the rate of discovery of defects: from all the defects that are either passed or passing through testing (Task Passing Test* Probability Change Required) we subtract the approved ones ("Approved Changes") and let the rest flow into "Known Changes". The Model 3 formulation also allows us to consider the introduction of additional defects during the rework process ("Dfct from Rwrk") as well as the possibility that some defects are not corrected in the rework process ("Generate changes during change activity"). All these defects flow into "Undiscovered Changes".

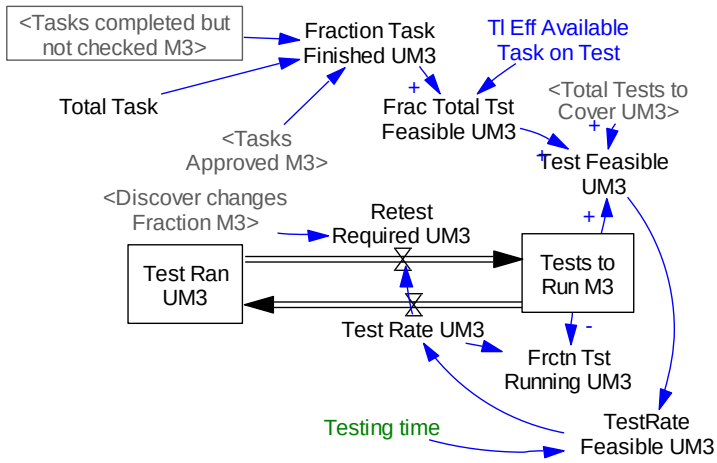


Figure 7- The structure for testing sector in Model 3. Table function for impact of task availability on test feasibility is highlighted in blue. Shadow variables, specified by <> signs, are coming from the main sector of the model shown in the previous figure.

Finally, the testing process starts with a fixed set of tests (in the stock "Tests to Run" (see Figure 7)). The fraction of tasks that are finished determines, based on the nature of work, how much testing can be done (through "Tl Eff Available Task on Test"). At one extreme all tasks completed can be tested because different pieces of project are tested independent from each other. Under these conditions the table function is the 45 degree line. At another extreme one may need all the pieces of the project to come together before anything can be tested. Such project has a table function that remains at 0 for all values but for input 1, which has an output of 1. Realistic projects are often somewhere between the two extremes, but we use the first case in this study because Model 1 does not capture testing explicitly. Once tasks are run, they flow into the "Test Ran" (see Figure 7) stock. On the other hand, the fraction of tasks that are rejected due to discovered defects are cycled back to "Tests to Run" so that they can be administered again later.

Model 3 allows us to consistently account for multiple defects per task. It also ties the testing process with alternative accuracy and coverage levels to the coflows of tasks and defects. It achieves these by including one more independent stock than Model 2 (total of 7 stock variables). However, by its nature, this structure continues to assume perfect mixing of tasks and defects and thus misses on heterogeneity in tasks, defect probability per task, stochastic variations in task completion, or interactions between different tasks. To account for some of these considerations an agent-based model may be required.

Model 4- The last model we compare the rest of structures with is an agent based model of rework cycle. This model can release several of assumptions necessary in the first three formulations and allow for more accurate simulations. A parallel project structure is used for this model. However, the agent-based model requires tracking of each task, rather than stock variables for different groups of tasks with similar characteristics. For example Model 2 included only 6 independent stock variables. In Model 4 each task can be in one of 4 alternative states (Not completed; completed but not tested; to be reworked; and approved). Moreover, each task includes dynamic variable such as the number of defects in the task, as well as, static (potentially stochastic) parameters including the completion or rework time for the task.

Figure 8 offers an overview of this model. Here the different stages of rework cycle along with important parameters of the model are presented. Each process stage includes a process (such as doing the work, testing, etc; similar to flow variables in SD models). Stages may also include queues before the process is executed. At the beginning of simulation all the tasks (of which there are "intialTasksToComplete") are released into the queue for the "completion". A task waits there until a resource is free to work on that task (from a pool of "numResource"). The "completion" process takes different amount of time for different tasks. For the base case analysis we assume the time per task is exponentially distributed with the expected value of "meanTaskSize", which directly relates to the productivity concept in the first three models. During the completion process defects are generated due to different reasons with a Poisson process with mean rate of "errorRateCompletion". Completed tasks are put into the queue for "testing". Testing resources are unlimited (for simplicity and keeping in parallel with other models) thus testing time has an exponential distribution with mean of "meanTestTime". During this time each of the defects is missed with the probability of "probMissingDefect". In absence of defects, or if all defects are missed, the task is approved, otherwise, it is sent to the "rework" stage. Here the task is reworked for some time (exponentially distributed with meanReworkSize). We assumed in previous models that productivity of personnel in doing the work initially or doing rework is the same, thus meanTaskSize= meanReworkSize. Defects can be both added, and removed, during the rework process. The two Poisson processes have rates of "errorCorrectionRate" and "errorRateRework". While generally errorCorrectionRate is bigger than errorRateRework (or otherwise the project does not finish in expectation), by chance some tasks may end up with more defects at the end of rework process than how they started. After rework tasks are sent back to the testing station for new tests. Resources are shared for completion and rework stages and a first-come first-served allocation policy is put into effect.

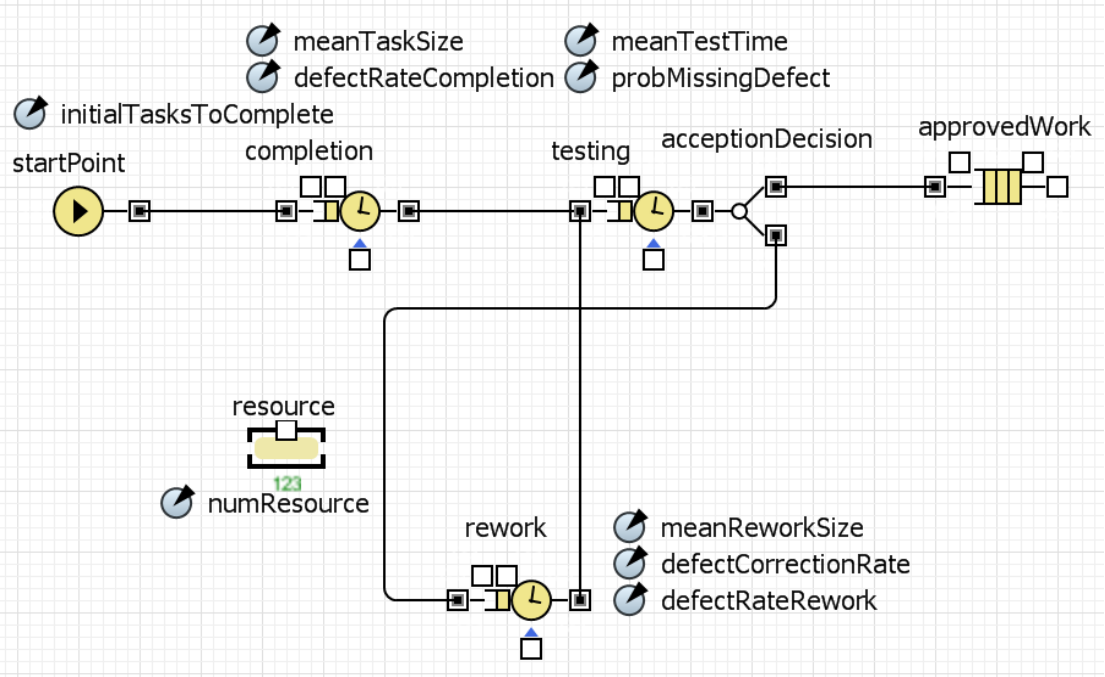


Figure 8- The structure of Model 4, the agent based formulation. Different parameters are shown in circles with small triangle on the side. The clock figures represent stages of work (completion, testing, and rework). The causal relationships are not specified in this graph, but are covered in the text.

The complexity of Model 4 relates directly to the number of tasks tracked. Each additional task requires keeping track of its state and defects and planning its related events, linearly increasing the model complexity with task size. Model 4 includes 9 parameters that govern its performance. In the next section we discuss how we derive the parameters for other models from these nine parameters. Before that however, we summarize the main characteristics and capabilities of the fours models discussed above. Table 1 provides this comparison. For each model we discuss the level of complexity of the model and what it adds, in terms of capability, to the previous model. We limit this summary to the capabilities of the models discussed here and not their ultimate capability after addition of structure to them.

Table 1- The overall comparison between the different formulations discussed in this paper. For each formulation its complexity (in terms of number of stocks or state variables) and the capabilities included in that formulation are discussed. The capabilities are cumulative, so each model includes what is available in the previous models, plus the items discussed here.

Model	Complexity	Capabilities (cumulative)
M1 (Richardson & Pugh)	2 independent stocks	• Basic rework cycle • Can include feedbacks to productivity, quality, and resources.
M2 (Ford & Sterman)	6 independent stocks	• Captures the quality of finished project • Includes precision of testing
M3 (Rahmandad)	7 independent stocks	• Can include multiple defects per task • Can include different test coverage levels • Can include different sizes for tests
M4 (Agent-	Linearly	• Includes task heterogeneity

based)	increases with number of tasks	• Includes stochastic behavior in completion and rework • Can include different priorities for tasks or precedence relationships • Can include interactions in task quality (e.g. a defect in task 3 increases the probability of defect in task 4)

Appendix 1 provides detailed formulation for M1-M3, all models are available in supplementary material for independent simulation and analysis. We first implemented all the models in Vensim ™ but later switched to Anylogic ™ for the ease of use in case of M4. We therefore report more user friendly graphs from Vensim, while the simulations are all completed in Anylogic. We supply both of these models for interested researchers.

3-1- Parameter Relationships

In this section we derive the parameters of the M1-M3 based on the micro level parameters we defined for M4. The derivation of parameters allows us to have a better understanding of the relationships between different formulations. It also enables building on calibrated parameter values from previous work to inform new models with alternative formulations.

The derivation of parameters for M3 is relatively straight forward. Most concepts remain unchanged. Namely Initial Tasks to Complete, Available Resource, Testing Time, Probability of Missing a Defect, and Error Correction Rate are equivalent conceptually and in their values, to counter parts in the agent based model (See Table 2). In fact many of these concepts are shared, without any change, across all models (thus no suffix included in parameter names). Productivity in M3 (and other models) is the number of tasks a resource can complete in one unit of time (e.g. month). This is therefore the reverse of the number of time units it takes to finish one task (or meanTaskSize in M4). Similarly rework productivity would be the reverse of meanReworkSize. In this study we assumed the productivity for rework and completion are equal, and thus the mean sizes for a task and its rework are equal (in expectation, since M4 has a distribution for task and rework sizes). Minimum time to do a task or to do rework also equals the meanTaskSize and meanReworkSize parameters because those parameters determine how long a single task will take to finish, which is the minimum time needed to finish the tasks.

Defect rates in rework and completion for M3 count the number of defects introduced per task completed (or reworked). Therefore they should take into account both the average defects created per unit of time (defectRateCompletion/Rework) and the length of time spent on each task (meanTaskSize or meanReworkSize). Similarly "Defect Correction Rate M3" is equal to defectCorrectionRate in M4 multiplied by the meanReworkSize. Finally, we introduced the parameter "Task Covered by Test UM3"[3] which is not used in any other model, because the other models do not include the possibility of having different types of tests with different numbers of tasks covered in each test. Such considerations are possible to implement in Model 4, however, for simplicity we do not include them and assume that a single task is covered by each test. We also assume all the tasks are included in the testing process, thus the initial number of "Tests to Run UM3" equal the initial number of tasks.

Models 1 and 2 have several parameters similar to Model 3, which are similarly derived from Model 4. The main conceptual difference between models 2 and 3 is the different formulation and understanding of defect and change. In model 2 tasks can either be defective

[3] UM3 is used to distinguish a concept unique to model 3, where as M3 specifies a concept that has parallels in other models.

(have a corresponding change in the co-flow) or not, a binary choice. Where as in Models 3 and 4 tasks can be defective to different degrees, because they can have multiple defects. This difference in conceptualization requires us to re-define the probability of introducing changes in models 1 and 2, as well as the defect correction and discovery processes. Let us first consider "Completion Defect Probability M21". This parameter is shared between Models 1 and 2 and specifies what is the probability a completed task is defective (in the binary conceptualization of task quality). We can relate this probability to the conceptualization in the Models 3 and 4 by finding what the probability that a task has no defect is, given the distribution of defects per task. That probability, calculated as one minus the probability of having no defects in a Poisson distribution, is[4]:

$$\textit{Completion Defect Probability M21} = 1 - e^{(-\textit{Defect rate completion M3})} \quad (4)$$

The same consideration comes into play for calculating the Defect Correction Probability M2. Here, however, both fixing of defective tasks and making mistakes that can make the task further defective come into play. We estimate this concept based on parallel parameters in M3 and M4 by assuming that a defective task is corrected if a)At least one defect is fixed in that task (one minus probability of fixing no error), AND b)No error is added to the task. This formulation is only an approximation. In fact the different concepts of task and error are not completely consistent across Models 3 and 2, and thus a one to one relationship does not exist between the two models. While in the Model 3 (and 4) a project can accumulate more defects in rework if "Defect Rate in Rwrk M3" exceeds "Defect Correction Rate M3", Model 2 does not include such possibility. If tasks are always either correct or incorrect, we can not make them more incorrect, and therefore the rework cycle ultimately finishes.

In light of the conceptual difference between Models 2 and 3, one of the equations in Model 2 requires further elaboration. As Figure 4 shows, "Discover Changes Fraction M2" is a function of "Probability of missing a defect" and "Probability change required M2". The straight forward, but incorrect, formulation may suggest multiplying "Probability change required M2" by one minus "Probability of missing a defect" (Call it *PM*). However, note that we missed one of the (possibility many) defects with the latter probability. If tasks are either correct or incorrect (binary view of M2), then probability of missing a defective task is lower than the same concept for a "defect" among many that can spoil a task in Model 3. Therefore we first find what the expected number of defects embedded in a defective task (call it *ED*) is, based on the "Probability change required M2" (call it *PC*). We then find what the probability of rejecting a task (*DC*) is, given the expected number of defects in that task (*ED)*:

$$\textit{PC} = 1 - \textit{Probability No Error Present} = 1 - e^{-ED} \text{ (from the Poisson distribution)}$$

$$\rightarrow \textit{ED} = -\ln(1-\textit{PC}) \quad (5)$$

$$\textit{DC} = 1 - \textit{Probability No Error Detected} = 1 - e^{(-\textit{ED}.\textit{PM})}$$

$$\rightarrow \textit{DC} = 1 - e^{(\ln(1-\textit{PC})\textit{PM})} \quad (6)$$

Note that *PC* is a dynamic variable and therefore *DC* can also change dynamically in a simulation. We use this latter formulation instead of the original formulation (Ford and Sterman 1998) to reduce the discrepancy between M2 and M3 performances due to different conceptions of task and defect.

Finally, Model 1 does not include several of the parameters used in Models 2-4 because it lacks formulations for testing, or separate stocks for rework and new work. The only parameter introduced by Model 1 is "Time to Discover Rework M1" which is somewhat similar to the delay involved in testing (though not exactly the same concept), and therefore we use the Testing

[4] For a Poisson distribution with mean M defects per task, the probability that no defect exists is e^{-M}.

Time as an approximation for it. Table 2 summarizes the relationship between different parameters across different models.

Table 2- The summary of parameters in different models and their relationships. For Model 4 only parameters are listed, while for other models their values in terms of Model 4 parameters is also included. Detailed discussion is available in the text.

Model 4	***Model 3***	***Model 2***	***Model 1***
initialTasksToComplete	*Initial Tasks to Complete= initialTasksToComplete*	*Initial Tasks to Complete= initialTasksToComplete*	*Initial Tasks to Complete= initialTasksToComplete*
numResource	*Available Resource= numResource*	*Available Resource= numResource*	*Available Resource= numResource*
meanTaskSize	*Productivity=1/meanTaskSize*	*Productivity=1/meanTaskSize*	*Productivity= 1/meanTaskSize*
	Min Time to Do Task= meanTaskSize	*Min Time to Do Task= meanTaskSize*	*Min Time to Do Task= meanTaskSize*
defectRateCompletion	*Defect rate Completion M3= defectRateCompletion* meanTaskSize*	*Completion Defect Probability M21= 1-$e^{(-\text{Defect rate completion M3})}$*	
meanTestTime	*Testing Time= meanTestTime*	*Testing Time= meanTestTime*	**NA**
NA	**NA**	**NA**	*Time to Discover Rework M1~ meanTestTime*
probMissingDefect	*Probability of missing a defect= probMissingDefect*	*Probability of missing a defect= probMissingDefect*	**NA**
Each test is assumed to cover a single task (no parameter)	*Task Covered by Test UM3*	**Each test is assumed to cover a single task**	**No explicit testing**
meanReworkSize	*Rework productivity* assumed to be equal to *completion productivity (=1/meanTaskSize)*		
NA	*Min Time to Do Rework= meanReworkSize= meanTaskSize*	*Min Time to Do Rework= meanReworkSize= meanTaskSize*	*Min Time to Do Rework= meanReworkSize= meanTaskSize*
errorCorrectionRate	*Defect Correction Rate M3= defectCorrectionRate* meanReworkSize*	*Defect Correction Probability M2= (1-$e^{(-\text{Defect Correction Rate M3})}$)*$e^{(-\text{Error Rate in Rwrk M3})}$*	**NA**
errorRateRework	*Defect Rate in Rwrk M3= defectRateRework* meanTaskSize*		**NA**

4- Simulation Analysis

In this section we report the simulation results comparing the four alternative models of rework cycle. We will first report the results of the base case analysis in which the models are simulated used the parameter values reported in Table 3. Given the stochastic nature of M4, 100 different simulations with different streams of random numbers are generated in this case and the statistics from this sample is reported. We will then conduct a comprehensive sensitivity analysis to understand how the alternative models compare under different parameter settings and what are the underlying structures that derive their differences.

Table 3- All important parameters used in all models, with their initial, low and high values, units and description

Parameter	**Base (Low, High)**	**Unit**	**Description**

probMissingDefect	0.3 (0.1, 0.9)	Dmnl	This parameter represents the quality of testing, as captured by probability of missing an error that exists in the task, while doing the testing.
meanTaskSize	0.2 (0.1, 0.4)	Month.Person/Task	This parameter defines the average size of tasks when doing work and test, as well as, the productivity of initial work and rework of the workers in M1-M3.
meanTestTime	0.1 (0.05, 0.2)	Month.Person/Task	This parameter defines the average time for doing the testing.
defectRateCompletion	2 (0.5, 8)	Defect/Month	The rate of creation of defects in initial work
errorRateRework	2 (0.5, 3.5)	Defect/Month	The rate of creation of defects while doing rework
errorCorrectionRate	4 (2.5, 8)	Defect/Month	The rate of correction of defects in the rework process
initialTasksToComplete	100 (20, 400)	Task	Total number of tasks included in the project
Distribution of Testing Time (M4)	*Fixed, Exponential(1)*	The base case includes no distribution, exponential distribution is used for the sensitivity analysis, where it is multiplied by mean value of the parameter	
Distribution of Task Size (M4)	*Fixed, Exponential(1)*	The base case includes no distribution, exponential distribution is used for the sensitivity analysis, where it is multiplied by mean value of the parameter	
Distribution of Rework Size (M4)	*Fixed, Exponential(1)*	The base case includes no distribution, exponential distribution is used for the sensitivity analysis, where it is multiplied by mean value of the parameter	
numResource	5	Person	No sensitivity analysis on the number of people

4-1- Base case

In the base case we investigate a project with 100 tasks which is taken up by five resources who can each complete five tasks per months. In absence of any errors, therefore, this project could be finished in four months. However, task completion is accompanied by introduction of defects, which can activate the rework cycle and lead to longer times for completion of the project. In fact models from M1 to M4 take exceedingly longer times to finish the project under these settings (see Table 4). Where as M1 is finished in 5.53 months, M4 takes in average 9.08 months to finish, and includes a relatively high standard deviation in this measure. Overall, the behavior modes are familiar for modelers: a smooth increase in the tasks approved, slowed down towards the end by the last iterations inside the rework cycle. Figure 9 presents the base case simulations for M1-M3 and a single random simulation for M4.

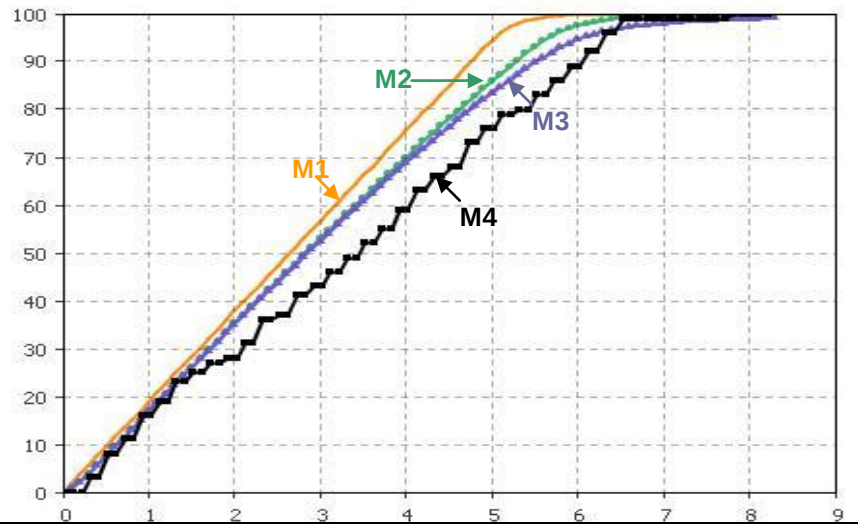


Figure 9- The time line of task approval for M1-M4. A single random run is shown for M4, while the statistics reported in the next tables and graphs are based on 100 simulations in each parameter setting. The finish time for different formulations in this case are: M1:5.53, M2: 6.47, M3: 8.04, M4: 7.78.

As the graph shows, M1 and M2 are very close in their base case behavior. M3 tends to better resemble the M4, specially in its longer finish time. The reason for significantly longer finish times for M3 and M4 is in the conception of errors in these models where a few tasks with multiple defects remain in the rework cycle and go through multiple cycles of rework before they are fully corrected, or incorrectly accepted. M2, with its binary conception of defective tasks, does not allow for such outliers.

In terms of final quality, the average defect in approved tasks can be calculated directly for M4 and M3, in terms of defects per approved task. For M2 a transformation is needed because the change coflow is conceptualized in a binary fashion. Therefore the Error Probability in Approved Tasks is telling us what fraction of tasks have some defect, but not the average number of defects. The latter quantity can however be calculated. Given Poisson distribution for the number of errors, the probability that a task is not defective can be calculated based on the density of defects in the tasks: Probability of Having No Defet=$1-e^{\text{Ave Dfct in Approved Task M2}}$, thus:

Ave Dfct in Approved Task M2=-ln(1-Error Probability in Approved Task M2) (7)

Having calculated this metric parallel to the similar concepts in M3 and M4, we can now compare the quality of resulting projects across different models. The second row in Table 4 report these results. In this case, somewhat surprisingly, M2 does a better job in replicating the results of the agent based model. M3 under counts the defect density in the approved work more significantly (0.116 in M3 vs. 0.164 in M2 and 0.185 defects per task in M4).

The reason behind this counter-intuitive behavior is that M3, based on perfect mixing assumption in differential equation models, assumes the distribution of errors remain Poisson in the tasks cycling through rework process. This assumption is in fact incorrect and consequential. Tasks with no errors are accepted in the testing process and go through without any problem. The remaining tasks therefore do not include any task with no defect, where as M3 assumes many of these tasks are indeed flawless (based on imposing the Poisson distribution). Given the same average number of defects per task in tasks cycled through rework process, M3 will also include more tasks with several defects, compared to M4. In short, M3 increases the heterogeneity of number of defects per task by assuming a Poisson distribution. The increased heterogeneity leads to lower number of problems that pass through the testing process in the next

round of tasks getting to testing. This is because the assumed flawless tasks have no defect to contribute to low quality accepted tasks.

An example illustrates this mechanism better. Consider two sets of four tasks with same average number of defects per task, which is one. First set includes two tasks with no defects and two tasks with two defect (higher heterogeneity, more similar to M3). Second set includes four tasks with one defect each (lower heterogeneity, more similar to M4). If these two sets go through a testing process with probability of p for missing a defect, then first set will have an expected number of errors equal to $2p^2$, while this number is equal to 4p for the second set. For all feasible values of p (between 0 and 1), the defect density will be lower for the first set, which resembles M3. This comparison also suggests the main reason why M2 does better in capturing the quality metric: the binary definition of task correctness in M2 makes a clear distinction between tasks with no defects and those defective, and follows that distinction throughout the testing and acceptance processes. It therefore does not suffer, as acutely, from the problem discussed for the M3.

Table 4- Values of Finish Time and Average Defect in Approved Tasks for all 4 models gotten under Base case. Results for 100 simulations are summarized for M4.

	M1	**M2**	**M3**	**M4 (Mean)**	**M4 (Stdev)**
Finish Time	5.528	6.468	8.044	9.081	2.576
Average Defect in Approved Tasks	Not Applicable	0.164	0.116	0.185	0.041

The differences in the time evolution between the four models are reported in the first row of **Error! Reference source not found.**. This graph reports the AAE metric we defined before (Figure 2) for comparison of different models (six different comparisons). Overall, M2 and M3 have the closest trajectory over time. After that M1 and M2 are most close, followed by M1-M3, M3-M4, M2-M4 and M1-M4. These results are consistent with the process of building the models, where M1 was the simplest, M2 and M3 followed, and M4 was the most detailed model. The differences between the models are not so significant and range between 1% (M2-M3) and 8% (M1-M4).

The standard deviations for M4 metrics are relatively large (e.g. 20-30 percent of the mean values), and therefore are not negligible. This suggests that variation in project evolution created through rework cycle, even in the absence of any other feedback structure, could still be significant. Given the significant costs associated with extremely late or low quality projects, this may suggest that having more detailed models are warranted for many practical problems where risk assessment is part of the problem definition.

In selecting among different aggregate formulations, M1 is the most aggregate, and the least precise model. It is most useful if a quick and simple insight model is needed to capture the basic idea of rework cycle. Selecting between M2 and M3 is less clear cut. While M2 offers more accurate quality estimates of the final project, M3 provides a more accurate evolutionary pathway for how long the project takes.

4-2- Sensitivity of results to different project types

We conduct additional sensitivity analysis to understand the scope and robustness of initial results and the level of difference between different models under alternative conditions. We change the model parameters from the base case values as reported in Table 3. Parameters are varied on a large interval to see how the models compare under relatively extreme conditions. In

each case we repeat 100 simulations for M4 and calculate the metrics similar to how we conducted the base case analysis. The detailed results of this analysis are reported in Table 5 and **Error! Reference source not found.**. The following conclusions emerge.

Increasing (decreasing) probability of missing an error decreases (increases) the finish time and has the reverse effect on average defect in accepted work. By reducing simulation time and the stochasticity induced by the rework cycle, poor testing procedure also brings the four models closer to each other. In fact in the extreme when no testing is done and every task is accepted, rework cycle is no more active, and thus it can not introduce randomness into M4. Consequently as the quality of testing process improves, the value of M4 increases in terms of stochastic behavior different from M1-M3. An improved testing process also increases the relative value of M3 over M2 because it leads to longer projects and higher quality, where as with poor testing, the two models become almost identical.

Besides the obvious impact on project length, increasing the task size has the effect of increasing the defect hazard for each task, leading to more significant impacts of rework cycle and the quality of tasks in the cycle. As a result for higher task sizes M4 becomes a more reliable model. Moreover, given the better performance of M2 than M3 in capturing project quality, larger task sizes tend to promote M2 over M3.

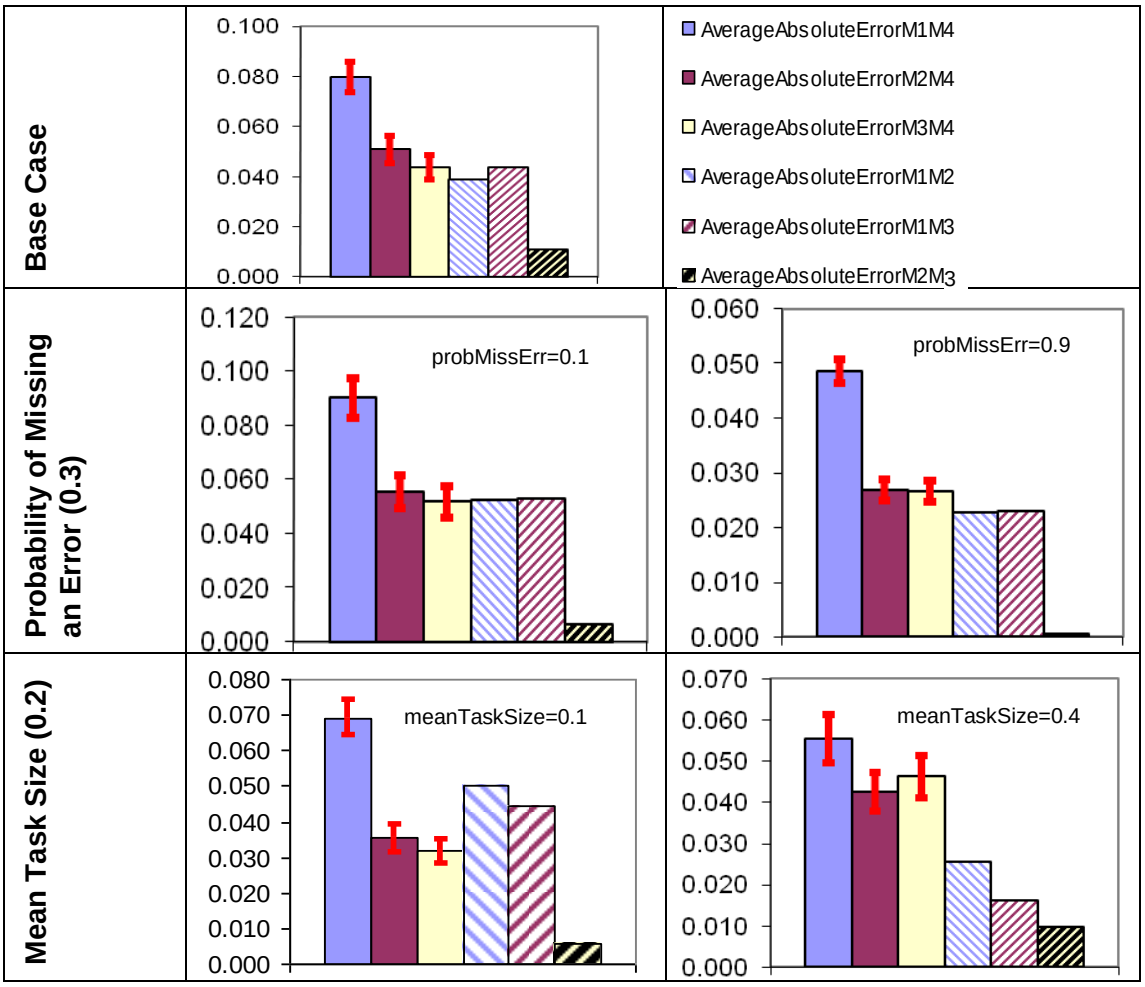
Increasing testing time increases the length of the rework cycle and thus its impact on the stochastic behavior of M4. Lack of explicit testing process in M1 leads to larger discrepancies between M1 and other modes as we increase testing time. The other comparative results are not changed significantly by testing time and impact on average quality is negligible.

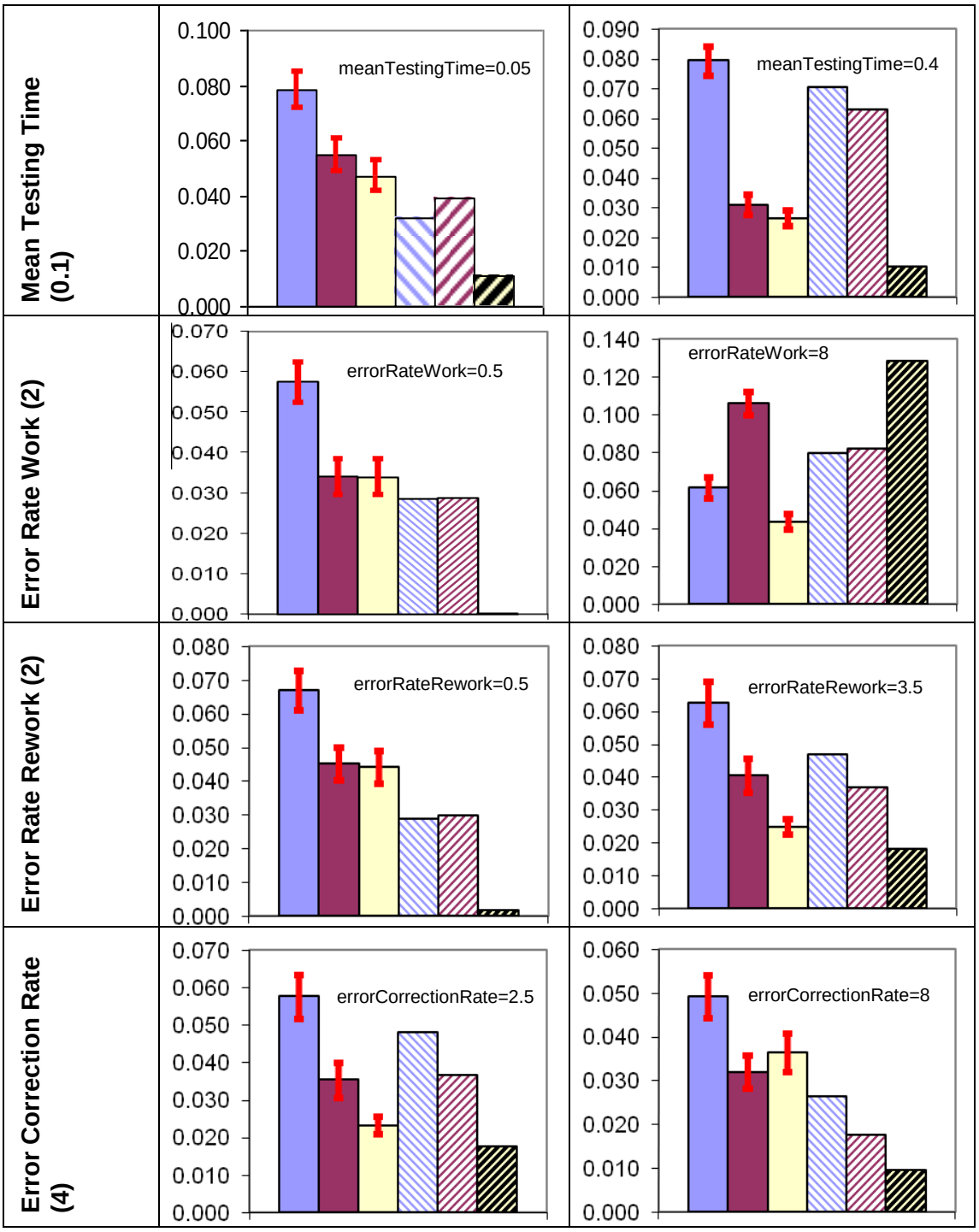
Completion defect rate has a significant impact on the behavior of the models. Increasing this parameter activates rework cycle significantly and leads to longer project times, increased discrepancy of M4 and M2, and increased quality difference between M4 and M3. Interestingly, the M3 remains very close in its evolutionary time line to M4, despite the fact that it underestimates the defects that pass through the testing process and are in approved work.

The impact of rework defect rate is somewhat more subtle. It has very limited impact on M2 and no impact on M1. Yet an increase in this parameter significantly increases the finish times for M3 and M4 because by introduction of more defects during rework these models will often include a few tasks which will accumulate multiple defects and will lengthen the project for a long time. Nevertheless, this process does not increase the gap (AAE) between M4 and M2 or M1 significantly, because for the majority of project life the models remain close to each other, and only the long final few tasks are extended. By this final stage all projects are largely completed and therefore the AAE measure remains small. In fact the impact of error correction rate is also very similar to the impact of defect rate in rework, but only in the opposite direction. This should come as no surprise since both processes of rework defect introduction and defect correction impact the outflow of stock of tasks with known changes.

Changing the number of tasks in the project scales the projects without changing the strength of underlying processes and therefore does not impact the mean performances of the models significantly. However, it has a major impact on the variations across different simulations for M4. With increased project size, the law of large numbers washes the variations away and leads to standard deviation values not so far from base case for both finish time and average defects (in fact standard deviation for average defect reduces for larger projects). Therefore increased project size actually reduces the relative value of M4, given the lower divergence in the behavior of the model from its mean performance.

Having distributions, rather than fixed values, for testing time, task size, and rework size have some minor impacts on M4 performance. Heterogeneous testing time slightly reduces finish time and has limited impact otherwise. Distributed task size increases finish time for M4 slightly because a few lengthy tasks may delay the project, and leads to slightly lower defect density in the approved tasks. The reason for these observations is similar to the mechanism underlying the difference between M3 and M4 in the average defect accepted. Higher heterogeneity in task size increases the heterogeneity in defect per task and thus leads to higher-quality final projects because more tasks are flawless (the smaller tasks) and the bigger containing many defects are more likely caught and recycled in the testing process. This mechanism also explains the slightly lengthier projects, as more rounds of rework cycle are needed for large, problematic, tasks.




Mean Testing Time (0.1)
meanTestingTime=0.05
meanTestingTime=0.4
Error Rate Work (2)
errorRateWork=0.5
errorRateWork=8
Error Rate Rework (2)
errorRateRework=0.5
errorRateRework=3.5
Error Correction Rate (4)
errorCorrectionRate=2.5
errorCorrectionRate=8

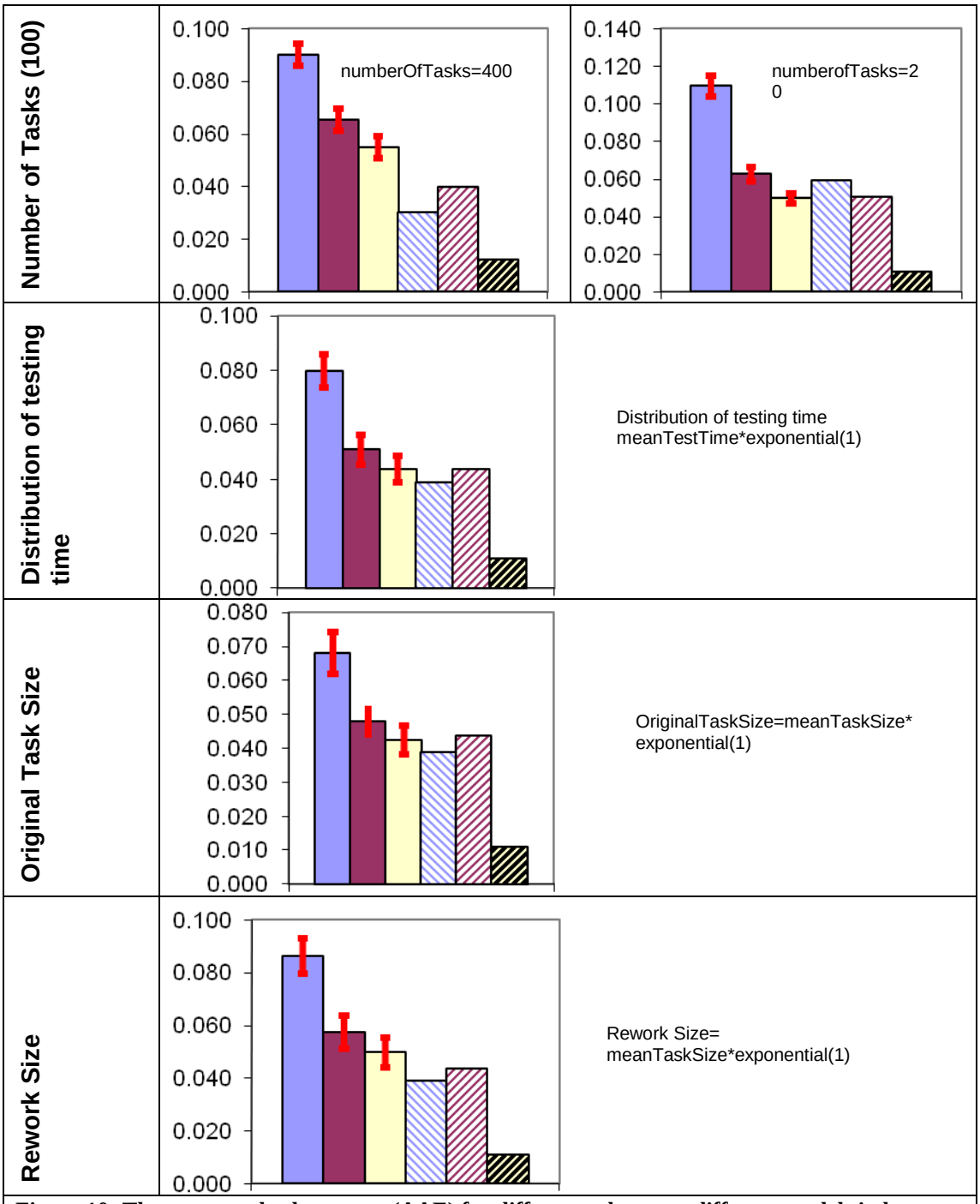


Figure 10- The average absolute error (AAE) for differences between different models in base case and different sensitivity analysis settings. For sensitivity cases each parameter is identified with its base value in parentheses and the sensitivity values on the respective graphs. For AAE between M4 and other models (the first 3 bars on charts), 95% confidence intervals are also graphed based on mean and standard deviation from the sample of 100 simulations.

Table 5- Detailed results of sensitivity analysis based on the change of each parameters listed in Table 3. Results for M4 are reported based on 100 simulations.

Metrics / Simulation Parameters	Finish Time M1	Finish Time M2	Finish Time M3	Finish Time M4, Mean	Finish Time M4, Std Deviation	Ave Dfct in Approved Task M2	Ave Dfct in Approved Task M3	Ave Dfct in Approved Task M4, Mean	Ave Dfct in Approved Task M4, Std Deviation
Base Case	5.528	6.468	8.044	9.081	2.576	0.164	0.116	0.185	0.041
probMissingDefect=0.1	6.005	7.641	8.924	10.822	3.088	0.059	0.034	0.069	0.032
probMissingDefect=0.9	4.300	4.441	4.454	4.423	0.167	0.376	0.380	0.396	0.068
meanTaskSize=0.1	2.416	3.134	4.315	4.743	1.971	0.103	0.081	0.126	0.037
meanTaskSize=0.4	14.574	15.816	15.374	18.259	2.823	0.269	0.141	0.246	0.047
meanTestTime=0.05	5.485	6.296	7.643	8.601	2.124	0.164	0.116	0.194	0.043
meanTestTime=0.4	5.999	7.900	10.960	12.182	4.984	0.164	0.115	0.185	0.040
defectRateCompletion=0.5	4.438	4.875	4.914	5.570	1.681	0.048	0.043	0.047	0.023
defectRateCompletion=8	13.046	9.938	20.984	20.575	4.139	0.361	0.142	0.523	0.088
errorRateRework=0.5	5.528	6.015	6.187	6.573	0.876	0.142	0.104	0.162	0.047
errorRateRework=3.5	5.528	7.013	26.742	31.756	28.907	0.186	0.119	0.223	0.060
errorCorrectionRate=2.5	5.528	7.085	26.742	32.375	28.921	0.189	0.119	0.251	0.057
errorCorrectionRate=8	5.528	5.920	5.648	6.101	0.569	0.136	0.091	0.128	0.038
initialTasksToComplete=20	20.976	23.531	26.528	29.682	3.563	0.164	0.117	0.188	0.024
initialTasksToComplete=400	1.795	2.485	4.011	3.093	2.215	0.164	0.115	0.179	0.087
Distribution of Testing Time (M4)	5.528	6.468	8.044	8.788	2.386	0.164	0.116	0.185	0.046
Distribution of Task Size (M4)	5.528	6.468	8.044	9.189	2.408	0.164	0.116	0.166	0.044
Distribution of Rework Size (M4)	5.528	6.468	8.044	9.084	2.442	0.164	0.116	0.203	0.047

5- Discussion and conclusions

In this paper we develop and compare four alternative formulations for modeling rework cycle dynamics. We show how to derive the parameters from models at different levels of aggregation and detail, and how their overall behaviors compare to each other. The basic mode of behavior for all models is the typical growth in the number of tasks approved. The final stages of the project can however be significantly slower for M3 and M4, because a few tasks with multiple defects may cycle through rework process for a long time. As a result M3 and M4 are much closer in their finish time and overall fit. However, the heterogenizing effect of perfect mixing assumption in M3 leads to higher quality levels for M3 compared to M4 (and M2).

Overall, similar modes of behavior are observable across different formulations. The major difference is the increased finish time for M3 and M4, due to existence of a few tasks with multiple errors. This difference, however, may be of important theoretical and practical significance. Previous research has attributed 90% syndrome, the long-time continuation of projects that are almost finished, to managerial responses to schedule pressure that increase the error rate (Ford and Sterman 2003). Our results suggest that a more basic process, the iteration of a few tasks with multiple defects, may create similar symptoms, without requiring any change in the defect rate, or other parameters of the system. In fact, the common formulations used in system dynamics for modeling projects hides this effect because they do not allow for multiple defects per task. Both M3 and M4 models introduced in this paper can capture these effects.

A detailed agent based model is also more informative in providing a range of behaviors not otherwise observable from deterministic models. Given the significant impact of extreme performances on expected costs and benefits of projects, a realistic cost-benefit analysis can benefit from the additional detail that allows for capturing stochastic variation endemic to project evolution. However, we show that the major modes of behavior and relationships between mean values remain largely similar across different formulations and therefore basic insights and high-level conclusions can be derived from simpler, more aggregate formulations. In fact, for larger projects with many tasks, the stochastic effects are more likely to be washed out, at least for projects with a common pool of tasks and no strict precedence relationship. Inclusion of precedence relationships is feasible both through introducing multiple phases of project (in differential equation models) and by explicit inclusion in agent-based models. If such precedence relationships are important and wide-spread in a project, the complexity of agent-based models may not exceed the multi-staged differential equation ones.

Sensitivity analysis offers additional insights on the extent of applicability and similarity of different formulations. In absence of defects or with very poor testing, rework cycle is largely removed and all models behave in very similar patterns. Moreover, very effective rework process (high error correction rate) reduces the impact of rework cycle and the differences across models. As we increase the defect rate and testing precision, and reduce rework effectiveness, the rework cycle is strengthened, and potential for divergence of the model behaviors increase. The simplest model, M1, is often the most apart from the more realistic M4 model. This is not surprising given that M1 does not include explicit testing process or defects. The difference between M2 and M4 also increase, specially in comparison of their finish times. M3 and M4 remain close in finish time under different parameter settings, yet a persistent bias is observed in that M3 gives higher quality estimates for approved work, than M4 does. This is due to perfect mixing assumption of M3 that counts tasks waiting for rework to include many with no defect, and thus in the next round of testing accepts them as flawless. The binary conception of errors in M2 is actually

closer to M4 formulation in this case, and thus leads to more precise quality levels in finalized projects. Note that we have reformulated M2 from the original concept offered by Ford and Sterman (Ford and Sterman 1998), to make it consistent with the conception of tasks and defects that allow for multiple defects per task. The original formulation indeed further diverges from M4. Nevertheless, this observation suggests that neither of M2 and M3 dominates the other for practical purposes. An open research question is to find a simple model formulation that captures both the quality and the finish time for a project more precisely than M2 or M3.

The four formulations in this paper provide slightly different opportunities for expansion and inclusion of alternative feedbacks. M1 can include feedbacks to quality and productivity, as well as defect discovery rate. M2 has additional flexibility to capture feedbacks to testing quality. M3 allows for different test sizes (testing one or multiple tasks, e.g. unit testing vs. integration testing) and changes in rework quality. Finally M4 is the most flexible model in terms of possible feedback and additional effects that it can capture, including precedence relationships, task priorities, heterogeneity of task size, resource productivity, and testing and rework size, among others. Benefiting from these flexibilities in M4 however requires more detailed modeling and coding.

In this research we focused on a central, but narrow, project conception and compared multiple formulations. We hope these results provide a set of contingency rules of thumb for selection of appropriate formulations for specific problems. Depending on the shape and amount of available data, computational and modeling resources at hand, the level of quantitative precision required, and the client's main goals, we think all four formulations could be the best option to be used in a setting, and none is dominated by others. Future research can offer new formulations that build on this comparative analysis to address the short-comings of models M1, M2, and M3 without the need to disaggregate to the agent-based level. One can also study the effect of different precedence relationships and task priorities, as well as feedback effects, on the comparative performance of these and other formulations. Despite these shortcomings, we hope this research provides interested scholars with a better map of available formulations and more appropriate assumptions for modeling project dynamics.

References:

AbdelHamid, T. and S. Madnick (1991). Software project dynamics: An integrated approach. Englewood Cliffs, NJ, Prentice-Hall.

Cooper, K. G. (1980). "Naval Ship Production: A Claim Settled and a Framework Built." Interfaces **10**(6).

Ford, D. N. and J. D. Sterman (1998). "Dynamic modeling of product development processes." System Dynamics Review **14**(1): 31-68.

Ford, D. N. and J. D. Sterman (2003). "Overcoming the 90% syndrome: Iteration management in concurrent development projects." Concurrent Engineering-Research and Applications **11**(3): 177-186.

Levitt, R. E., J. Thomsen, T. R. Christiansen, J. C. Kunz, Y. Jin and C. Nass (1999). "Simulating project work processes and organizations: Toward a micro-contingency theory of organizational design." Management Science **45**(11): 1479-1495.

Lyneis, J. M. and D. N. Ford (2007). "System dynamics applied to project management: a survey, assessment, and directions for future research." System Dynamics Review **23**(2-3): 157-189.

Rahmandad, H. (2005). Three essay on modeling dynamic organizational processes. Sloan School of Management. Cambridge, Massachusetts Institute of Technology. **Ph.D.**

Rahmandad, H. and J. D. Sterman (2008). "Hetergeneity and network structure in the dynamics of contagion: Comparing agent-based and differential equation models." Management Science **Forthcoming**.

Repenning, N. P. (2000). "A dynamic model of resource allocation in multi-project research and development systems." System Dynamics Review **16**(3): 173-212.

Richardson, G. P. and A. L. Pugh (1981). Introduction to system dynamics modeling with DYNAMO. Cambridge, Mass., MIT Press.

Sengupta, K. and T. K. Abdelhamid (1993). "Alternative Conceptions of Feedback in Dynamic Decision Environments - an Experimental Investigation." Management Science **39**(4): 411-428.

Sterman, J. (2000). Business Dynamics: systems thinking and modeling for a complex world. Irwin, McGraw-Hill.

Appendix 1- Complete Model Formulations

Model 1

Approve Task rate M1 = (1 - Completion Defect Probability M21) * Completion Rate M1
    Units: Task/Month
Available Resources = 5
    Units: dmnl
Completion Rate M1 = MIN (Completion rate constraint M1 , Potential Work Rate
)
    Units: Task/Month
Completion Defect Probability M21 = 1 - EXP (- Defect rate completion M3)
    Units: dmnl
Completion rate constraint M1 = Tasks not completed M1 / Min Time to Do Task

    Units: Task/Month
Cumulative Work Done = INTEG(Rate of Doing Work , 0)
    Units: Task
Defect rate completion M3 = Error Rate Initial Work * Mean Task Size
    Units: dmnl
Initial Tasks to Complete = ELMCOUNT(Task)
    Units: Task
Mean Task Size = 0.2
    Units: Month
Min Time to Do Rework = Mean Task Size
    Units: Month
Min Time to Do Task = Mean Task Size
    Units: Month
Potential Work Rate = Available Resources * Productivity
    Units: Task/Month
Probability of missing a defect = 0.2
    Units: dmnl
Productivity = 1 / (Mean Task Size)
    Units: Task/(Month*Person)
Rate of Doing Work = Rework Generation + Approve Task rate M1
    Units: Task/Month
Rework Discovery M1 = Undiscovered changes M1 / Time to Discover Rework M1
    Units: Task/Month
Rework Generation = Completion Defect Probability M21 * Completion Rate M1
    Units: Task/Month
Tasks approved M1 = INTEG(Approve Task rate M1 , 0)
    Units: Task
Tasks not completed M1 = INTEG(Rework Discovery M1 - Rework Generation - Approve Task rate M1
    , Initial Tasks to Complete)
    Units: Task
Testing time = 0.1
    Units: Month

TIME STEP = 0.015625
Units: Month
Time to Discover Rework M1 = Testing time
Units: Month
Undiscovered changes M1 = INTEG(Rework Gene

Model 2

Approve changes M2 = (Quality Assurance M2 * Probability change required M2) - Discover internal changes M2
Units: Task/Month
Approve Task Rate M2 = Quality Assurance M2 * (1 - Discover changes fraction M2)
Units: Task/Month
Available Resource = Available Resources
Units: Person
Available Resources = 5
Units: dmnl
Ave Dfct in Approved Task M2 = - ln ((1 - Error probability in approved tasks))
Units: **undefined**
Change task rate M2 = Feasible Rwrk Rate M2
Units: Task/Month
ChangeDensity UM2 = ZIDZ (Known changes M2 , Tasks to be changed M2)
Units: dmnl
Changes approved M2 = INTEG(Approve changes M2 , 0)
Units: Task
Completion Defect Probability M21 = 1 - EXP (- Defect rate completion M3)
Units: dmnl
Completion rate M2 = Feasible New Dev Rate M2
Units: Task/Month
Correct changes M2 = XIDZ (Known changes M2 , Tasks to be changed M2 , 0) * Defect Correction Probability M2 * Change task rate M2
Units: Task/Month
Defect Correction Probability = 1 - EXP (- Defect Correction Rate M3)
Units: dmnl
Defect Correction Probability M2 = Defect Correction Probability * EXP (- Defect Rate in Rwrk M3)
Units: dmnl
Defect rate completion M3 = Error Rate Initial Work * Mean Task Size
Units: dmnl
Defect Rate in Rwrk M3 = Error Rate Rework * Mean Task Size
Units: dmnl
Dev Resources to New Dev M2 = Dsrd New Dev Resources M2 * MIN (1, ZIDZ (Available Resource , Total Desired Resources M2)) * (1 - SW Project Finished M2

)
Units: Person
Dev Resources to Rework M2 = Dsrd Rwrk Resources M2 * MIN (1, ZIDZ (Available Resource
, Total Desired Resources M2)) * (1 - SW Project Finished M2
)
Units: Person
Discover change rate M2 = Discover changes fraction M2 * Quality Assurance M2

Units: Task/Month
Discover changes fraction M2 = 1 - EXP (ln (1 - Probability change required M2
) * (1 - Probability of missing a defect))
Units: dmnl
Discover internal changes M2 = Discover change rate M2
Units: Task/Month
Dsrd New Dev Resources M2 = Tasks not completed M2 / Min Time to Do Task / Productivity
Units: Person
Dsrd Rwrk Resources M2 = Tasks to be changed M2 / Min Time to Do Rework / Productivity

Units: Person
Error probability in approved tasks = ZIDZ (Changes approved M2 , Tasks approved M2)
Units: dmnl
Feasible New Dev Rate M2 = Dev Resources to New Dev M2 * Productivity
Units: Task/Month
Feasible Rwrk Rate M2 = Dev Resources to Rework M2 * Productivity
Units: Task/Month
Generate changes during change activity M2 = (XIDZ (Known changes M2 , Tasks to be changed M2
, 0)) * (1 - Defect Correction Probability M2) * Change task rate M2

Units: Task/Month
Generate changes during completion activity M2 = (Completion rate M2 * Completion Defect Probability M21
)
Units: Task/Month
Initial Tasks to Complete = ELMCOUNT(Task)
Units: Task
Initial Tasks to Complete M2 = 10000
Units: Task
Known changes M2 = INTEG(Discover internal changes M2 - Correct changes M2
- Generate changes during change activity M2 , 0)
Units: Task
Mean Task Size = 0.2
Units: Month

Min Time to Do Rework = Mean Task Size
Units: Month
Min Time to Do Task = Mean Task Size
Units: Month
Probability change required M2 = ZIDZ (Undiscovered changes M2 , Tasks completed but not checked M2
)
Units: dmnl
Probability of downstream phase discovering changes = 0.2
Units: dmnl
Probability of missing a defect = 0.2
Units: dmnl
Productivity = 1 / (Mean Task Size)
Units: Task/(Month*Person)
Quality Assurance M2 = Tasks completed but not checked M2 / Testing time
Units: Task/Month
Quality Assurance rate in downstream phase = 1
Units: dmnl
Release package size = 0.2
Units: dmnl
SW Project Finished M2 = if then else (Initial Tasks to Complete M2 * Threshold Finish Task
< Tasks approved M2 , 1, 0)
Units: dmnl
Tasks approved M2 = INTEG(Approve Task Rate M2 , 0)
Units: Task
Tasks completed but not checked M2 = INTEG(Change task rate M2 + Completion rate M2
- Discover change rate M2 - Approve Task Rate M2 , 0)
Units: Task
Tasks not completed M2 = INTEG(- Completion rate M2 , Initial Tasks to Complete
)
Units: Task
Tasks to be changed M2 = INTEG(Discover change rate M2 - Change task rate M2
, 0)
Units: Task
Tasks to be changed M3 = INTEG(Discovery Change rate M3 - Change Task Rate M3
, 0)
Units: Task
Testing time = 0.1
Units: Month
Total Desired Resources M2 = Dsrd New Dev Resources M2 + Dsrd Rwrk Resources M2

Units: Person
Threshold Finish Task = 0.99
Units: dmnl
Undiscovered changes M2 = INTEG(Generate changes during change activity M2
+ Generate changes during completion activity M2 - Approve changes M2

- Discover internal changes M2 , 0)
Units: Task

Model 3

Approve Task rate M3 = Task Passing Test UM3 - Discovery Change rate M3
Units: Task/Month

Approved Changes M3 = (Tstd Task Accptd UM3 * Dfct Dnsty in Accepted Task UM3
+ (Approve Task rate M3 - Tstd Task Accptd UM3) * Probability change required M3
)
Units: Task/Month

Available Resource = Available Resources
Units: Person

Available Resources = 5
Units: dmnl

Ave Dfct in Approved Task M3 = ZIDZ (Changes Approved M3 , Tasks Approved M3
)
Units: **undefined**

Ave Dfct in Rwrk Task M3 = ZIDZ (Known Changes M3 , Tasks to be changed M3
)
Units: dmnl

Ave Dfct in Tstd Task UM3 = Task Covered by Test UM3 * Probability change required M3

Units: dmnl

Change Task Rate M3 = Feasible Rwrk Rate M3
Units: Task/Month

Changes Approved M3 = INTEG(Approved Changes M3 , 0)
Units: Task

completion Rate M3 = Feasible New Dev Rate M3
Units: Task/Month

Correct changes M3 = Change Task Rate M3 * Defect Fix Rate M3
Units: Task/Month

Defect Correction Probability = 1 - EXP (- Defect Correction Rate M3)
Units: dmnl

Defect Correction Rate = 4
Units: Error/Month

Defect Correction Rate M3 = Defect Correction Rate * Mean Task Size
Units: Error/Month

Defect Fix Rate M3 = MIN (Ave Dfct in Rwrk Task M3 , Defect Correction Rate M3
)
Units: dmnl

Defect rate completion M3 = Error Rate Initial Work * Mean Task Size
Units: dmnl

Defect Rate in Rwrk M3 = Error Rate Rework * Mean Task Size
Units: dmnl

Defects from Dev UM3 = completion Rate M3 * Defect rate completion M3

Units: Task/Month
Dev Resources to New Dev M3 = Dsrd New Dev Resources M3 * MIN (1, ZIDZ (Available Resource
, Total Desired Resources M3)) * (1 - SW Project Finished
)
Units: Person
Dev Resources to Rework M3 = Dsrd Rwrk Resources M3 * MIN (1, ZIDZ (Available Resource
, Total Desired Resources M3)) * (1 - SW Project Finished
)
Units: Person
Dfct Dnsty in Accepted Task UM3 = EXP (- (1 - Probability of missing a defect
) * Ave Dfct in Tstd Task UM3) * Ave Dfct in Tstd Task UM3
* Probability of missing a defect
Units: dmnl
Dfct from Rwrk UM3 = Change Task Rate M3 * Defect Rate in Rwrk M3
Units: Task/Month
Discover changes Fraction M3 = 1 - EXP (- Ave Dfct in Tstd Task UM3 * (1
- Probability of missing a defect))
Units: dmnl
Discover internal changes M3 = Task Passing Test UM3 * Probability change required M3
- Approved Changes M3
Units: Task/Month
Discovery Change rate M3 = Quality Assurance M3 * Discover changes Fraction M3

Units: Task/Month
Dsrd New Dev Resources M3 = Task not completed M3 / Min Time to Do Task / Productivity

Units: Person
Dsrd Rwrk Resources M3 = Tasks to be changed M3 / Min Time to Do Rework / Productivity

Units: Person
Error Rate Initial Work = 2
Units: Error/Month
Error Rate Rework = 2
Units: Error/Month
Feasible New Dev Rate M3 = Dev Resources to New Dev M3 * Productivity
Units: Task/Month
Feasible Rwrk Rate M3 = Dev Resources to Rework M3 * Productivity
Units: Task/Month
Frac Total Tst Feasible UM3 = Tl Eff Available Task on Test (Fraction Task Finished UM3
)
Units: dmnl
Fraction Task Accepted UM3 = ZIDZ (Tasks Approved M3 , Total Task)
Units: dmnl

Fraction Task Finished UM3 = ZIDZ (Tasks Approved M3 + Tasks completed but not checked M3
, Total Task)
Units: dmnl
Frctn Tst Running UM3 = ZIDZ (Test Rate UM3 , Tests to Run M3)
Units: 1/Month
Generate changes during change activity M3 = Change Task Rate M3 * (Ave Dfct in Rwrk Task M3
- Defect Fix Rate M3)
Units: Task/Month
Generate changes during completion activity M3 = Defects from Dev UM3 + Dfct from Rwrk UM3

Units: Task/Month
Initial Tasks to Complete = ELMCOUNT(Task)
Units: Task
Known Changes M3 = INTEG(Discover internal changes M3 - Generate changes during change activity M3
- Correct changes M3 , 0)
Units: Task
Mean Task Size = 0.2
Units: Month
Min Time to Do Rework = Mean Task Size
Units: Month
Min Time to Do Task = Mean Task Size
Units: Month
Probability of missing a defect = 0.2
Units: dmnl
Probability change required M3 = ZIDZ (Undiscovered changes M3 , Tasks completed but not checked M3
)
Units: dmnl
Productivity = 1 / (Mean Task Size)
Units: Task/(Month*Person)
Quality Assurance M3 = Task Covered by Test UM3 * Test Rate UM3
Units: Task/Month
Retest Required UM3 = Test Rate UM3 * Discover changes Fraction M3
Units: Task/Month
SW Project Finished = if then else (Initial Tasks to Complete * Threshold Finish Task
< Tasks Approved M3 , 1, 0)
Units: dmnl
Task Covered by Test UM3 = 1
Units: dmnl
Task not completed M3 = INTEG(- completion Rate M3 , Initial Tasks to Complete
)
Units: Task

Task Passing Test UM3 = Totl Task to Tst UM3 * Frctn Tst Running UM3
Units: Task/Month
Tasks Approved M3 = INTEG(Approve Task rate M3 , 0)
Units: Task
Tasks completed but not checked M3 = INTEG(completion Rate M3 + Change Task Rate M3 - Approve Task rate M3 - Discovery Change rate M3 , 0)
Units: Task
Tasks to be changed M3 = INTEG(Discovery Change rate M3 - Change Task Rate M3 , 0)
Units: Task
Test Feasible UM3 = Max (0, Frac Total Tst Feasible UM3 * Total Tests to Cover UM3 - (Total Tests to Cover UM3 - Tests to Run M3))
Units: Task
Test Ran UM3 = INTEG(Test Rate UM3 - Retest Required UM3 , 0)
Units: Task
Test Rate UM3 = TestRateFeasible UM3
Units: Task/Month
Testing time = 0.1
Units: Month
TestRateFeasible UM3 = Test Feasible UM3 / Testing time
Units: Task/Month
Tests to Run M3 = INTEG(Retest Required UM3 - Test Rate UM3 , Initial Tasks to Complete / Task Covered by Test UM3)
Units: Task
TIME STEP = 0.015625
Units: Month
Time to Replan = 1
Units: Month
Tl Eff Available Task on Test ([(0,0)-(10,10)],(0,0),(1,1))
Units: dmnl
Total Desired Resources M3 = Dsrd New Dev Resources M3 + Dsrd Rwrk Resources M3

Units: Person
Total Development = completion Rate M3 + Change Task Rate M3
Units: Task/Month
Total Task = Tasks Approved M3 + Totl Task to Tst UM3
Units: Task
Total Tests to Cover UM3 = Test Ran UM3 + Tests to Run M3
Units: Task
Totl Task to Tst UM3 = Tasks completed but not checked M3 + Task not completed M3 + Tasks to be changed M3
Units: Task
Threshold Finish Task = 0.99
Units: dmnl
Tstd Task Accptd UM3 = Task Covered by Test UM3 * (1 - Discover changes Fraction M3) * Test Rate UM3

Units: Task/Month

Undiscovered changes M3 = INTEG(Generate changes during completion activity M3
+ Generate changes during change activity M3 - Approved Changes M3
- Discover internal changes M3 , 0)

Units: Task

Model 4

Given the complexity of presenting model 4 in equations similar to Models 1-3, we only provide it as part of the online supplementary material for the paper.