System Dynamics Review
*System Dynamics Review* vol 26, No 4 (October–December 2010): 291–315
Published online 3 August 2010 in Wiley Online Library
(wileyonlinelibrary.com) **DOI**: 10.1002/sdr.435

# Modeling the rework cycle: capturing multiple defects per task

Hazhir Rahmandad[a]* and Kun Hu[a]

*Abstract*

The rework cycle is at the heart of modeling projects, one of the major research and application areas in system dynamics. The current formulations for the rework cycle assume each task is either defective or not. Yet in many projects multiple defects can occur in one task. In this study we introduce a new rework cycle formulation that accounts for multiple defects per task and flexibly captures the testing process. We compare this formulation with two established formulations from the literature. Analysis shows some differences in simulated projects' finish time and delivered quality across different models and we discuss how these differences depend on project characteristics and task structure. The new formulation could be especially useful when data on tasks, defects, and testing have different measures, and when each new defect in a task significantly increases the chances of the task being rejected. 

*Syst. Dyn. Rev.* **26**, 291–315 (2010)

Supporting information may be found in the online version of this article.

## Introduction

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. Project performance, in terms of schedule, cost, and quality, evolves through time; thus it is a dynamic concept that lends itself to the application of system dynamics modeling. The vast variation in quality, timeliness, and cost performance of projects across different fields has motivated many applications of system dynamics for project management. 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).

The majority of system dynamics studies that focus on project dynamics include a simulation model of project evolution. These models vary in their levels of complexity and the feedback effects that they capture (e.g. see Cooper, 1980; AbdelHamid and Madnick, 1991; Ford and Sterman, 1998; Taylor and Ford, 2006; Lee and Pena-Mora, 2007). However, the core feature of all these models, building on the ground-breaking consulting project by Pugh Roberts Associates in the 1970s, is the rework cycle (Cooper, 1993a,b,c). Rework cycle recognizes that the completion of a project task may be flawed,

[a] Industrial and Systems Engineering Department, Virginia Tech, Northern Virginia Center—Rm 430, 7054 Haycock Road, Falls Church, VA 22043, USA.
* Correspondence to: Hazhir Rahmandad. E-mail: hazhir@vt.edu
*Received November 2008; Accepted September 2009*

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. In the absence of the rework cycle, project completion is a function of the number and scope of tasks, the available resources and their productivity. By considering defects, quality and testing through rework cycle many path-dependent reinforcing loops are generated (e.g. burnout, error building on error, cutting corners) that critically impact the fate of projects. Thus the rework cycle is central to understanding project delays and disruptions (Lyneis and Ford, 2007).

Figure 1 provides a simple conception of the rework cycle, adopted from one of the early models of projects (Richardson and Pugh, 1981, ch. 4). Here tasks, upon completion, may flow into *Tasks Correctly Approved* (if they are correctly completed and accepted), or *Undiscovered Defective Tasks*: 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 the quality of the work (defect rate) determines the split into correct and defective categories. Even this very simple second-order[1] 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 the defect rate and productivity can be directly applied here (Cooper, 1980; Sengupta and AbdelHamid, 1993). The changes in resource availability and the 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 upstream and downstream projects (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 co-flow structure (Ford and Sterman, 1998; Sterman, 2000).

Despite multiple derivations, current rework cycle formulations conceptualize defects as binary: each task that is completed can either be defective or correct. If defective, that task is routed into *Undiscovered Defective Tasks* stock in Figure 1. More detailed formulations physically separate the tasks from the defects in two different stock and

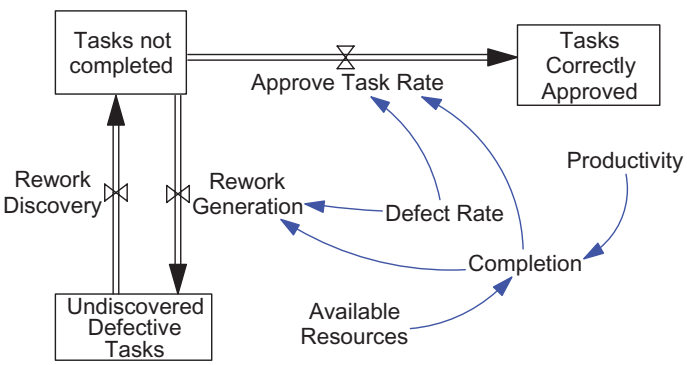


Fig. 1. Overview of the basic rework cycle structure. Only the most important causal links are demonstrated, for simplicity. Tasks are either correctly done and approved, or incorrectly done and moved to the *Undiscovered Defective Tasks* stock. *Rework Discovery* flow brings those tasks back to the stock of *Tasks not Completed*. *Defect Rate* specifies what fraction of those tasks completed are defective



DOI: 10.1002/sdr

flow chains (e.g. Ford and Sterman, 1998), yet they have sustained the assumption that each task is either accompanied by a defect or is defect-free. The binary conception of defect has been a convenient and useful assumption as it has allowed for modeling the rework cycle with straightforward and intuitive formulations. For example, the fraction of tasks in *Undiscovered Defective Tasks* stock specifies a simple measure of project quality.

However, the simplicity of the common one-to-one relationship between task and defect has some drawbacks. In many real-world projects stakeholders do not perceive the tasks and defects to be of the same nature, or to have a one-to-one relationship with each other. For example, software engineers typically think about the units of tasks as lines of code, function points, or modules (Grimstad *et al.*, 2006). Multiple defects, or faults, can show up in any of these units of task. Therefore, calling a function point defective means there is at least one fault in that function point, but does not clarify the exact number of faults. Similarly, typical product development projects are organized around developing distinct components and integrating them. The defects in a component could be numerous. Conceptualizing a component as either defective or correct cannot distinguish between a component with five defects and one with a single defect. However, a distinction between these two is important because each of the defects will require some distinct rework effort later on. Even after much rework on fixing four defects out of five, the component is still "defective", while much less rework can fix a component with a single defect.

Moreover, measuring the project performance and quality depends on how we define the tasks and defects. In the traditional rework cycle formulations a defect rate of 0.5 means that there is 50 percent chance that a completed task is defective. However, if we consider the possibility that many defects may occur in doing a unit of task, defect rate should specify how many defects are on average introduced as a result of the completion of a single unit of task. This number may still be smaller than one, but it would point to the mean of a probability distribution on the number of defects per task. A defect rate of 0.5 means on average 0.5 defects are introduced on the completion of a task, but there is a chance that one, two, or more defects are introduced, depending on the probability distribution for defects per task. Estimating parameters of simulation models from the empirical data requires conceptualizing the tasks and defects as they are measured in the data. Therefore, where defects are measured without imposing an upper bound of one defect per task, we need a formulation for the rework cycle that allows for capturing multiple defects per task. For example, in using data to estimate software project parameters we often face such a situation: faults (which are identified in testing) data are coming from the change management system, while data on tasks completed (in function points or other metrics) come from a different source such as the version control system. The defect rate can therefore only be estimated in terms of faults per function point or similar metrics that potentially go beyond one (Stutzke and Smidts, 2001).

The goal of this paper is to introduce a new rework cycle formulation that allows for multiple defects per task. Capturing this possibility makes a difference in accounting for the amount of rework in a project and for empirically estimating the parameters of a simulation model based on numerical data. It also may provide insights into explaining different observed behavior modes in typical projects. We note that our goal is primarily theoretical: realistic project models are not limited only to the rework cycle



DOI: 10.1002/sdr

and interesting dynamics are generated through feedback effects and integration of the rework cycle with other relevant parts of a project. We also elaborate on the relationship of this new formulation with the current formulations of the rework cycle common in the literature. This helps other researchers and modelers to know about the similarities and differences among these formulations and select the one that is the most appropriate for their specific application.

## Modeling and analysis

In this section we first derive the new rework cycle formulation based on micro-level assumptions about tasks, defects, and testing process. After conducting a set of simulation tests to establish the robustness of this new formulation, we compare it with two well-established formulations from the extant literature. We specify metrics of comparison, introduce these two formulations, and elaborate on their relationships with the new formulation. We then conduct extensive sensitivity analysis to find out the similarities and differences of these formulations under alternative project characteristics. Discussion of results and implications for modeling project dynamics are presented after the analysis section.

### *Capturing multiple defects per task with a new rework cycle formulation*

The new formulation for the rework cycle should explicitly distinguish between tasks and defects by having separate stock and flow chains for each. The critical issue is that these stock and flow structures are tightly coupled; that is, they follow a co-flow structure (Sterman, 2000), where for each stock of tasks (those completed pending test, those tested and rejected, and those approved after test) there is a parallel stock capturing defects in those (completed) tasks (See Figure 2). For example, there might be 100 tasks pending test, with 130 defects among them, leading to 1.3 defects per task. However, contrary to typical co-flow formulations, the outflows from *Tasks Pending Test* have different densities of defect. Those tasks that are approved are likely to have fewer defects than those which are rejected. For example *Tasks Approved* may have a 0.3 defect density (note that some defects are not caught in testing) while *Tasks Pending Rework* may have a 1.55 density of defects. Moreover, due to the perfect mixing assumption inherent in differential equation models, the weighted average of the density of defects in the approved and rejected tasks should add up so that the total flow of defects out of *Tasks Pending Test* does not change the density of defect in *Tasks Pending Test*. In our example above, out of each 10 tasks tested, two would be approved and eight should be rejected to keep this balance ($1.3 = 8/10 \times 1.55 + 2/10 \times 0.3$). Finally, we should establish a formulation specifying what fraction of tasks are accepted as a result of testing, given each level of defects density in *Tasks Pending Test*. In our example a defect density of 1.3 has led to 80 percent rejection, but what would be the rejection fraction for other defect densities?

These requirements for an internally consistent formulation create an interesting modeling challenge. At the heart of this challenge is the fact that with two stock variables for tasks and defects we can only know about the average number of defects per task, but not the distribution of the number of defects. However, to decide what fraction of tasks are approved in the testing process and what fraction are sent for rework, we



DOI: 10.1002/sdr

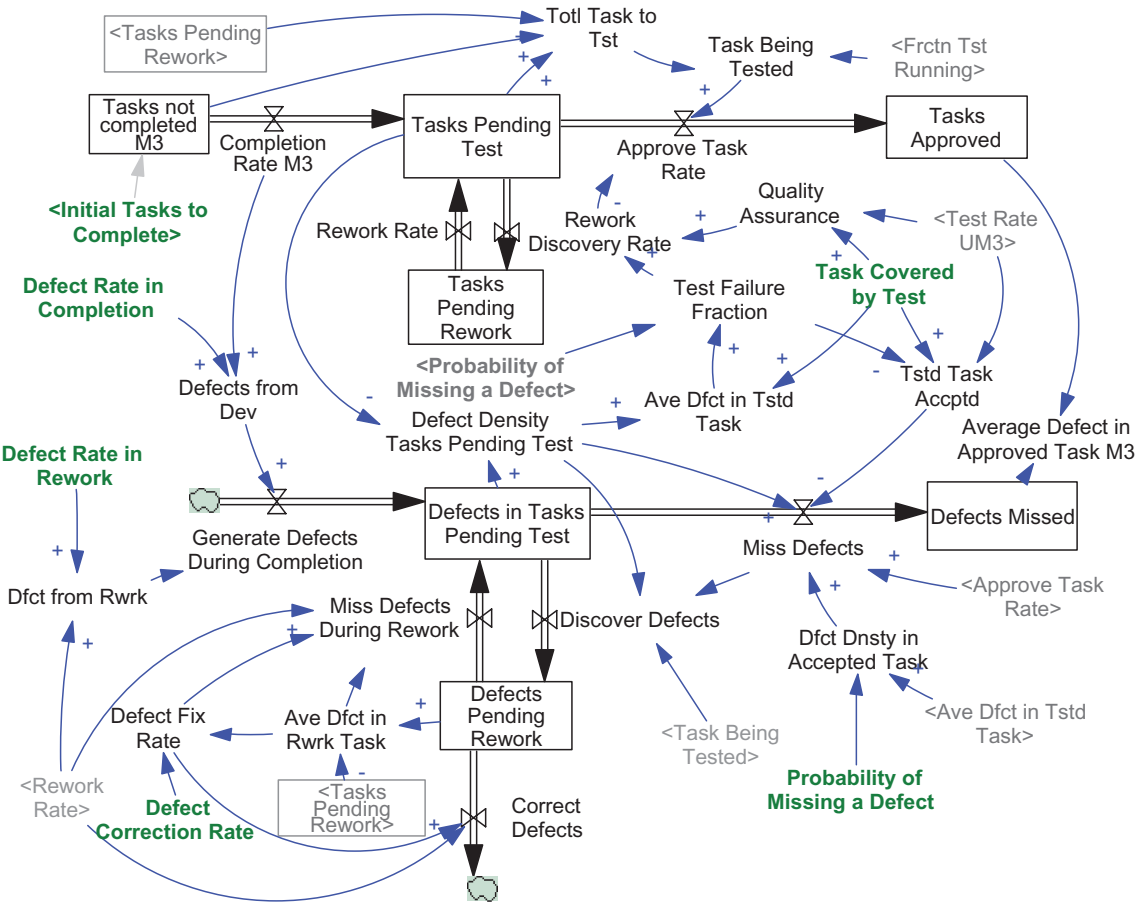


Fig. 2. The structure of the new model, simplified for clarity. The full model in Vensim™ is available in the online Appendix[2]

need to know what fraction of tasks are defect-free. Finding the defect-free fraction of tasks requires some additional distributional assumptions concerning how the defects are distributed among the tasks. Furthermore, if we consider the imperfections of the 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, by chance, be approved. Thus a flexible formulation should be able to determine what the defect density is in both approved and rejected tasks, depending on the effectiveness of testing.

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 examines interactions between multiple modules or functions, thus increasing the chance that a test fails because any of the components have a defect. These larger tests could lead to more task



DOI: 10.1002/sdr

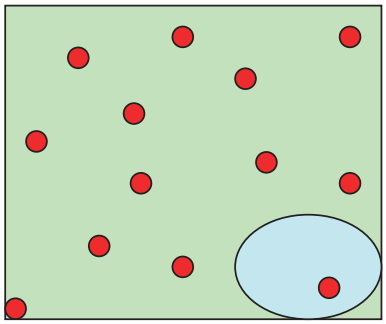

Fig. 3. The conceptual framework for modeling multiple defects per task. The area of the rectangle represents the tasks in one stock, the dots represent the defects, and the oval represents a test covering some tasks and potentially some defects

rejection. For example, if a test covers five tasks, its failure leads to sending all those tasks for further rework even though only one task may have a defect.

Figure 3 presents the conceptual framework we use to incorporate these considerations. Here we represent the tasks as the area of the rectangle: the larger the number of tasks, the bigger the rectangle. The number of tasks at each stage is represented in a stock variable (see Figure 2 for a partial stock and flow diagram of the model). Defects are shown as the dots distributed in the task area. The number of these defects is tracked in the stock variable *Defects in Tasks Pending Test* (Figure 2). A perfect mixing nature of material in a stock requires these defects to be randomly distributed in the area. That is, the location of each defect is independent of other defects, and all points in the area have the same chance of including a defect. Finally, a test is represented as an oval that covers some area, i.e. a number of tasks. A test is then rejected if at least one of the defects inside the test area is identified in the testing process. Otherwise the test is approved, accepting the part of the area covered by test into the stock of *Tasks Approved* (see Figure 2). The defects that have been missed in that test flow in parallel into the *Defects Missed* stock.

The perfect mixing requirement leads to independence of the placement of defects in the task area. That independence, in return, results in a Poisson distribution for the number of defects per any area in the task space because it satisfies the necessary requirements for observing a Poisson distribution: 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. Establishing the Poisson distribution for the number of defects per task, we can calculate the probability of finding $k$ defects in a test area of size $a$ (unit: task), where the defect density is $d$ (unit: defect/task):[3]

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

For example, the probability that no defect is present in the test area, and thus the test is approved (releasing the tasks in the test area into the *Tasks Approved* stock (Figure 2) ) is $e^{-da}$. In the numerical example, with a test area of $a = 1$ task, there is a 27 percent



DOI: 10.1002/sdr

(=$e^{-1.3}$) chance that no defect exist in the tested task, a 35 percent (=1.3 × $e^{-1.3}$) chance that one defect exists, and so on.

Furthermore, we can calculate the impact of imperfect testing. A test is passed under this setting if no defect is present in the test area, or a single defect is present and it is missed (with probability $\varepsilon$), or two defects are present and both are missed (which, assuming independence among different defects, will have a probability of $\varepsilon^2$), and so on. Therefore the probability of a test with area $a$ passing can be calculated as the sum of the probabilities of that test passing, conditional on different numbers of defects that could be observed:

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

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 the stock of *Defects in Tasks Pending Test* to stocks of *Defects Missed* and *Defects Pending Rework* (Figure 2). This expected value can also be calculated in closed form:

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

Finally, this conceptualization allows us to consider lower-than-complete test coverage. Testing can be imperfect not only because we may miss some defects in the task area that is being tested, but also because we may not test all the tasks and their interactions. Under a low-test–coverage scenario, a part of the task area is released without testing, along with the area covered by each test. The defects in the untested area are also released, with the average defect density, into the *Tasks Approved* (Figure 2). In order to consider such imperfect testing scenarios, it is helpful to introduce explicit stocks and flows for the tests that are being run.[4]

We use *Test Failure Fraction* (Figure 2) to measure what fraction of tests fail. This is one minus the fraction of tests that pass, found from Eq. 2. We calculate the number of tasks that are sent for rework using this fraction and the total amount of tasks covered by the testing flow (*Quality Assurance*). On the other hand, the tasks approved are potentially more, if the test coverage is lower than 100 percent. Therefore we calculate the *Approve Task Rate* by subtracting the rejected tasks from the total tasks that either go through testing or are 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 the testing rate is conducted at 0.15 tests per month, then 15 percent of tasks to be tested are each month flowing out of the stock of *Tasks Pending Test*, out of which some are caught in tests and sent to *Tasks Pending Rework* and the rest are approved.

We formulate the rate *Miss Defects* (see Figure 2) in the defect co-flow by adding up the two sources of defects that are missed. First, those defects missed in testing are captured in *Dfct Dnsty in Accepted Task* (See Eq. 3) multiplied by *Approve Task rate*. Second, the tasks approved without testing carry defects with the density of *Defect Density Tasks Pending Test*. Once the rate for approval of defects is determined, it is easy to calculate the rate *Discover Defects*: from all the defects that undergo testing



DOI: 10.1002/sdr

(*Tasks Being Tested* × *Defect Density Tasks Pending Test*) we subtract those missed (*Miss Defects*) and let the rest flow into *Defects Pending Rework*. This model formulation 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 (*Miss Defects During Rework*). All these defects flow back into *Defects in Tasks Pending Test* (Figure 2).

Finally, the testing process starts with a fixed set of tests (in the stock *Tests to Run* (see Figure 4)). The fraction of tasks that are finished determines, based on the nature of the work, how much testing can be done (through table function *Tl Eff Available Task on Test*). Inter-phase independencies have been formulated similarly (Ford and Sterman, 1998). At one extreme, all tasks completed can be tested because different pieces of the project are tested independently from each other. Under this condition the table function represents the line $x = y$. At another extreme, one needs all the pieces of the project to come together before anything can be tested. Such a project has a table function that remains at zero for all values but for input value of one, which has an output of one. Realistic projects are often somewhere between the two extremes, but we use the first case in our theoretical simulations for compatibility with other models examined. Moreover, given the space limitations, we do not consider constraints on testing resources, assuming that the test rate is only limited by *Testing Time*. Once tasks are run, they flow into the *Tests Ran* (see Figure 4) 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.

This model allows us to consistently account for multiple defects per task. It also ties the testing process with different accuracy and coverage levels to the co-flows of tasks and defects. These capabilities distinguish the current formulation from the previous work and provide an additional building block for modelers studying project dynamics where multiple defects per task are a potentially important consideration. This building

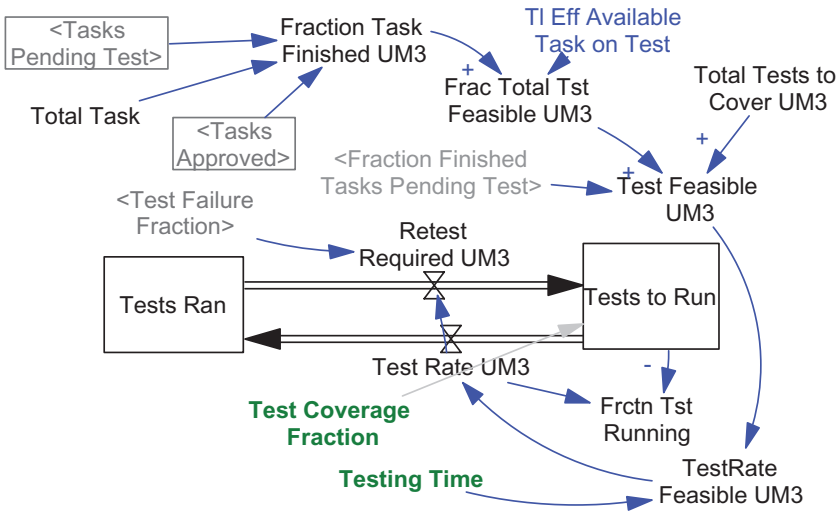


Fig. 4. The structure for the testing sector of the new model. Shadow variables, specified by < > signs, are coming from the main sector of the model shown in Figure 2



DOI: 10.1002/sdr

block can be integrated into extended models of project dynamics. The parameters *Productivity*, *Defect Rate in Completion*, *Defect Rate in Rework*, *Defect Correction Rate*, *Task Covered by Test*, *Probability of Missing a Defect* and *Test Coverage Fraction* can be made dynamic to capture the feedback effects on productivity, quality, and testing in different stages of a project.

*Analyzing and assessing the new formulation*

We analyze the behavior of this new formulation and assess its overall validity in a multi-tiered process. Given the theoretical nature of the current study we focus on behavioral mode plausibility, internal consistency, and robustness of formulations in extreme conditions. We will also compare the model's behavior and characteristics against two well-established formulations of the rework cycle to assess the reliability of the results and specify the relationships among these models. The comparative analysis would clarify the conditions under which the alternative models converge and diverge, and the reasons for their similarities and differences, thus enabling modelers to pick the model that is best suited for their applications at hand.

Base case model behavior

In the base case we simulate a project with 100 tasks, using five resources, each with the productivity to finish five tasks per month. The defect rate in completion is assumed to be 0.4 defects per task. Other parameters of the base case model are reported in Table 1. Figure 5 reports this project's performance, as simulated by the new formulation. The

Table 1. Base case parameter settings and parameter relationships among the three models (M1, M2, and M3). Units are specified under M3 column

| | Model 3 (M3) | Model 2 (M2) | Model 1 (M1) |
|---|---|---|---|
| Project scope and resources | *Initial Tasks to Complete* = 100 tasks | Same | Same |
| | *Available Resources* = 5 persons | Same | Same |
| Task completion | *Productivity* = 5 tasks/person/month | Same | Same |
| | *Min Time to Do Task* = 0.2 month | Same | Same |
| | *Defect Rate in Completion* = 0.4 defects/task | *Defective Completion Probability M2* = 0.33 (Eq. 4) | *Defective Completion Probability M1* = 0.33 (Eq. 4) |
| Testing | *Testing Time* = 0.1 month | Same | NA |
| | NA | NA | *Time to Discover Rework M1* = 0.41 (Eq. 7) |
| | *Probability of Missing a Defect* = 0.3 Dmnl | Same | NA |
| | *Task Covered by Test* = 1 task/test | NA | NA |
| | *Test Coverage Fraction* = 1 | NA | NA |
| Rework | *Min Time to Do Rework* = 0.2 month | Same | NA |
| | *Defect Correction Rate* = 0.8 defect/task | *Task Correction Probability M2* = 0.37 (Eq. 6) | NA |
| | *Defect Rate in Rework* = 0.4 defect/task | | NA |



DOI: 10.1002/sdr

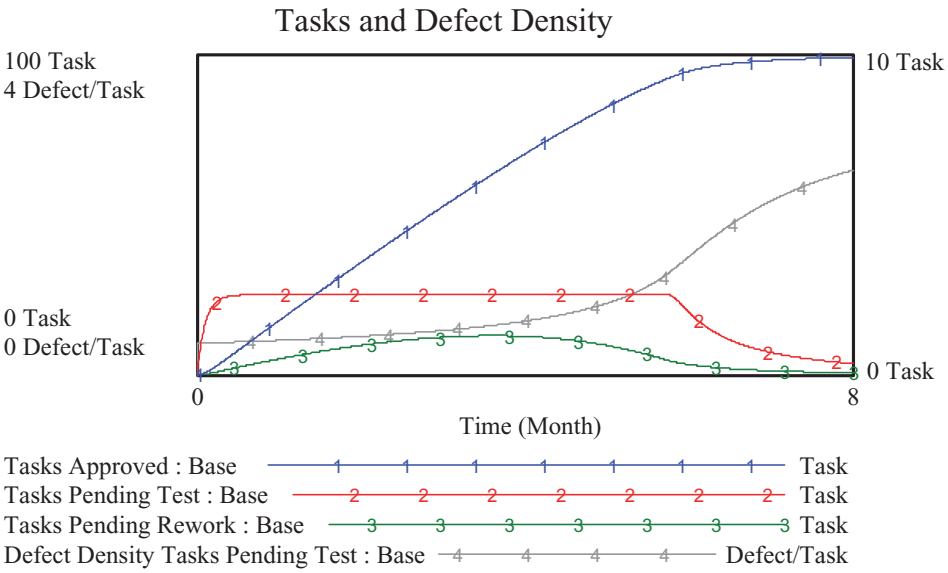


Fig. 5. The project's performance with the new formulation under base case parameter setting. Note different scale (right-hand side) used for "Tasks Pending Test" (graph marked with "2")

project goes through the typical rework cycle dynamic in which tasks are initially completed at a relatively constant rate and quickly tested (thus low level of *Tasks Pending Test*, item 2 in Figure 5 (note different scale)), tasks with discovered defects are sent for rework in *Tasks Pending Rework* (item 3 in Figure 5) and the rest are *Tasks Approved* (item 1 in Figure 5). The defect density in the tasks pending test (item 4 in Figure 5) starts from the defect rate (0.4) but gradually increases, as tasks with multiple defects remain in the rework cycle while the defect-free ones are approved. This leads to an escalation of the rework cycle towards the end of the project where a few tasks with multiple defects hold the project from completion. At its core, the rework escalation in this model points to a shift in the type of tasks being tested, from those tasks being tested for the first time (and containing relatively few defects), to those problematic tasks that have already gone through the rework process once or more, but still contain problems. At the end the project finishes after 8 months as a threshold for tasks approved is passed (in this case 99 percent). Interestingly, many real projects face a similar late-stage slow down, leading to the 90 percent syndrome where a project gets stuck when it seems very close to being finished (Ford and Sterman, 2003). However, these dynamics are created within the rework cycle formulation here and do not depend on the changes in quality, creation of new work, or concurrency effects that explain the 90 percent syndrome in other project models (Ford and Sterman, 2003; Taylor and Ford, 2006).

## Robustness to extreme conditions

We conducted additional simulations assessing the robustness of the current formulation to extreme values of the parameters of the model. The model generated



DOI: 10.1002/sdr

no significant surprises in this sensitivity analysis: increasing defect rate, reducing productivity, increasing test coverage, or strengthening testing precision (by reducing *Probability of Missing a Defect*) all increase the project length. Strengthening the testing precision or coverage, reducing defect rate, or increasing correction rate, would improve the delivered quality. Increasing the number of tasks covered by testing increases project duration and delivered quality because tests are now covering more tasks and thus more defects (see Figure 3), leading to higher rejection rates.

An interesting behavior arises when *Defect Rate in Rework* exceeds *Defect Correction Rate*, and testing process is perfect (i.e. *Probability of Missing a Defect* is zero and the test coverage fraction is one). Under these conditions the project will never finish because the meticulous testing process coupled with the defect-increasing rework cycle lead to a growing defect density among the tasks not approved and stops tasks from approval towards the end of the project. This behavior mode is logically plausible. In fact, previous research proposes another mechanism through which similar never-ending-project behavior can arise as more new tasks are created through rework, leading to an increasing backlog (Taylor and Ford, 2006, 2008).

## Applications

Ultimately a formulation's usefulness should be established over time in its application to diverse real-world problems. This is exactly how the classical rework cycle formulations have gained acceptance in the modeling community. While the empirical validation of the current formulation with real project data goes beyond the scope of the current paper, an initial application in modeling software projects shows promising results. This formulation was originally developed in modeling the development of a multiple-release software product (Rahmandad, 2005). Available data in that application included lines of code as a measure of tasks, and maintenance requests as a metric of defects identified through testing. Tests were separately measured and documented in that application, leading to a close correspondence between the three co-flows of tasks, defects, and tests in the application and the current formulation. The overall model included different stages of design and development, testing, launch, and maintenance of the software products and their interaction with market performance across six consecutive releases of a product. The model replicated dynamics that tie together multiple releases of a software product and the organizational capabilities in developing high-quality products. The basic findings of the case studies are documented elsewhere using a simpler and stylized model (Rahmandad and Weiss, 2009). Moreover, a software development management flight simulator has been built on the basis of the resulting detailed model and has been successfully tested by several software developers, providing additional confidence in the robustness and usefulness of the underlying formulation for the rework cycle discussed here.

### *Comparison with other formulations*

In this section we compare the rework cycle formulation we introduced with two well-established models from the literature. These models are built in parallel with the current model so that similar concepts, variables, and parameters are comparable across all three models. Parallel model development is achieved by starting from the new formulation (which we call M3 from now on), and deriving the parameters for the more



DOI: 10.1002/sdr

established formulations from M3. This process informs the relationship between parameter values of the new formulation and those of the established ones, improving the comparability of the results from different studies.

We compare the alternative models on their complexity (i.e. the number of independent stocks), structure (e.g. the different parameter settings and assumptions about the nature of work that can be changed in each formulation), and their behaviors. Behavior is measured along the dimensions of time (with metric *Finish Time*) and delivered quality (with metric *Defective Fraction in Approved Tasks*). *Finish Time*, the time it takes to approve 99 percent of the tasks in the project (or perceive them to be finished), is reported for different formulations. Given the constant resources and project size, *Finish Time* is a good metric for overall project cost as well and we do not report a separate cost metric. *Defective Fraction in Approved Tasks* is the fraction of tasks approved that include at least one defect at the time the project is finished. This metric tracks delivered quality in the traditional formulations. We calculate that in the new model by finding the fraction of tasks with one or more defects (one minus the fraction of tasks with no defects,[5] given the Poisson distribution of defects per task approved):

$$\textit{Defective Fraction in Approved Tasks} = 1 - e^{-\textit{Average Defect in Approved Task M3}} \quad (4)$$

After describing the two model formulations used from the literature for this comparison, we derive the relationship among the parameters of these three formulations. We then compare the models using different criteria, conduct model behavior comparisons, and assess the robustness of the results.

## Alternative formulations from the literature

### *Model 1*

This formulation is adopted from an early project model (Richardson and Pugh, 1981) as the simplest representation of the rework cycle which, with some modifications, has been applied extensively. Hereafter we call this model M1. Figure 6 presents an overview of this model. Tasks are completed based on current *Available Resources* and *Productivity*. The *Completion Rate M1* is also bounded by a minimum time to complete a task. A fraction of tasks are approved (either because they are done correctly, or because the testing process has 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 and sent back to the stock of tasks not completed.

It is important to note that the testing process is not explicitly captured in this model. Therefore *Rework Discovery M1* relates both to the defect generation and the quality of testing. Even with high real defect rates a lousy testing procedure can reduce the speed of discovering changes. Moreover, the formulation does not distinguish between the rework and the initial work because it sends all the discovered rework into *Tasks not Completed M1*. We therefore need to derive the parameters for *Defective Completion Probability M1* and *Time to Discover Rework M1* from the parameter values in the more detailed model. Finally, this model lumps together approved defective tasks with undiscovered defects, and both could be discovered later (contrary to the new formulation in which defects in an approved task cannot be reworked). In this formulation a project



DOI: 10.1002/sdr

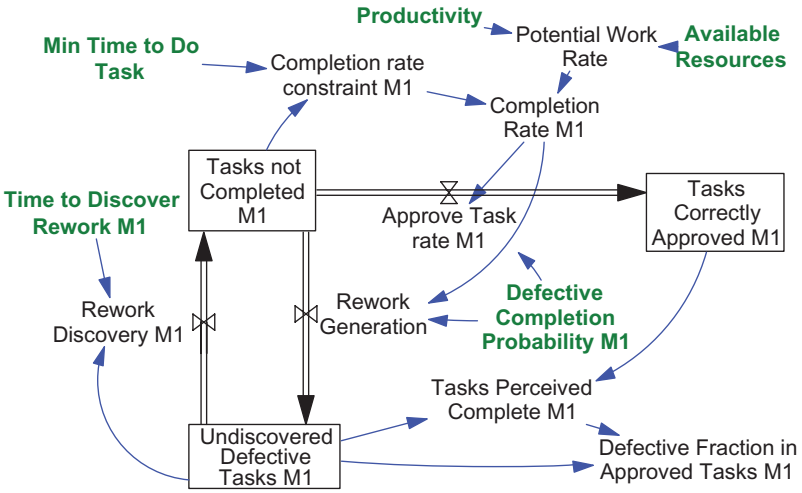


Fig. 6. The structure of Model 1. Model parameters are in bold. The complete model can be downloaded as supporting information

finishes when the sum of *Tasks Correctly Approved M1* and *Undiscovered Defective Tasks M1* reach a threshold (0.99 in this model). The fraction of tasks perceived complete that is in *Undiscovered Defective Tasks M1* stock will inform the project's delivered quality (*Defective Fraction in Approved Tasks M1*). The distinction between perceived and real progress is therefore important in this formulation and in calculating the finish time we focus on tasks *perceived* to be completed. Standard formulations are used in M1 and the full model is available as an online Appendix.

*Model 2*

Ford and Sterman (1998) introduced a more detailed formulation for the rework cycle. Similar to M3, their model includes a co-flow structure to track defective tasks along with the tasks in the rework cycle. We replicated their rework cycle formulation for the study at hand, calling it Model 2 or M2. Figure 7 provides the overview of this model. The full model is available in the online Appendix and from the original paper.

Available resources are allocated between *Completion Rate M2* and *Rework Rate M2* proportional to the desired completion rates for new tasks vs. rework. Desired rates depend on stocks of *Tasks not Completed M2* and *Tasks Pending Rework M2*, as well as time to do the tasks or the rework. *Defective Completion Probability M2* 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 *Tasks Pending Rework M2* stock and what fraction flows into *Tasks Approved M2* along with the *Defective Tasks Approved M2*. *Defective Fraction in Approved Tasks M2* is used as the delivered quality metric. For known defective tasks rework is not always successful in removing the problem. While some defective tasks are fixed in the rework process, some defects remain. The M2 formulation does not account for different levels of defectiveness, i.e.



DOI: 10.1002/sdr

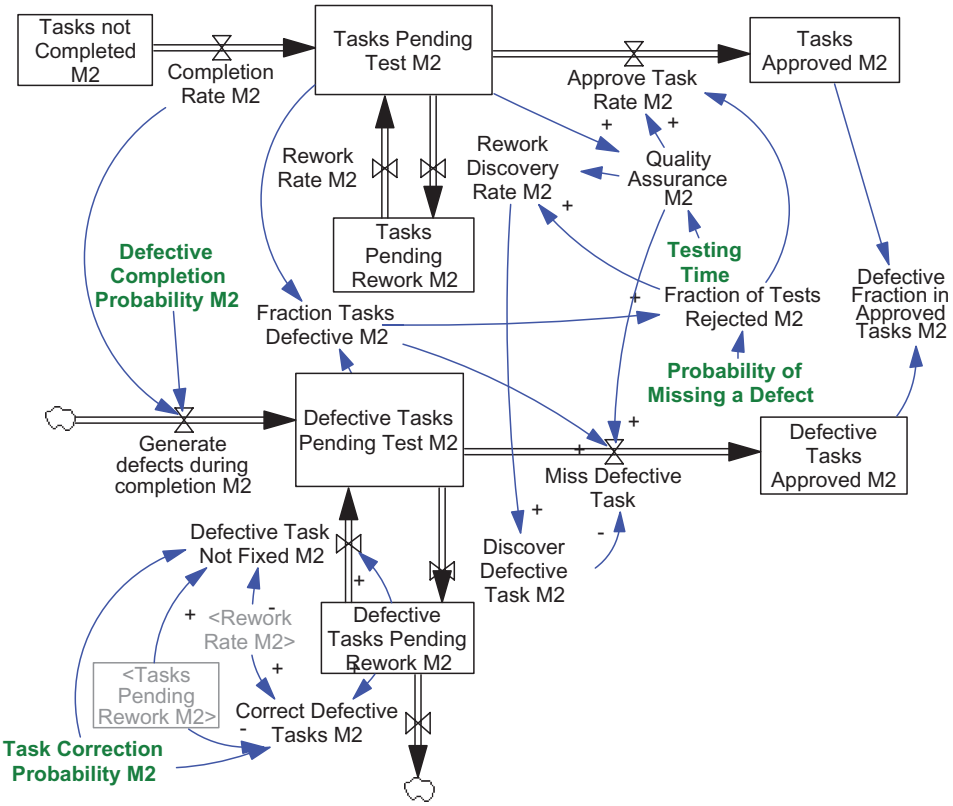


Fig. 7. Stock and flow structure of Model 2, adapted from Ford and Sterman (1998). Parameters are shown in bold and some sections of the model are removed to avoid clutter

having multiple defects per task. Therefore the relationship of parameters related to defect generation and correction probabilities in this model (e.g. *Defective Completion Probability M2*) should be established with those in M3.

## Parameter relationships

In this section we derive the parameters of M1 and M2 based on the parameters of M3. The derivation of parameters allows us to have a better understanding of the relationships between different formulations. It also enables building on parameter values from previous work to inform modeling work that uses the new formulation.

Several concepts remain unchanged across all models. Namely *Initial Tasks to Complete*, *Available Resources*, *Productivity*, *Testing Time*, and *Probability of Missing a Defect* are conceptually equivalent across M1–M3 (see Table 1). Many of these concepts are shared, without any change, across all models (thus no suffix is included in parameter names). There are also some parameters that are unique to the new equation. Specifically, *Task Covered by Test* 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



DOI: 10.1002/sdr

of tasks covered in each test. Similarly, *Test Coverage Fraction* is only used in M3, because different levels of testing comprehensiveness are not captured in M1 and M2.[6]

Different conceptualizations of defectiveness require us to find from M3 the probability of creating a defect (defective tasks) in Models 1 and 2, as well as the defect correction and discovery processes. Let us first consider *Defective Completion Probability M2*. This parameter specifies the probability that a completed task is defective (in the binary conceptualization of task quality). This probability can be related to M3 conceptualization by finding the probability that a task has one or more defects. That probability, calculated as one minus the probability of having no defects in a Poisson distribution, is

$$\textit{Defective Completion Probability M2} = 1 - e^{-\textit{Defect Rate in Completion}} \tag{5}$$

The same parameter can be used for *Defective Completion Probability M1*. A similar logic can be used for calculating the *Task 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 by assuming that a defective task is corrected during rework if (a) at least one defect is fixed in that task (one minus probability of fixing no defect), *and* (b) no defect is added to the task, given the *Defect Rate in Rework*. Specifically:

$$\textit{Task Correction Probability M2} = (1 - e^{-\textit{Defect Correction Rate}}) \times e^{-\textit{Defect Rate in Rework}} \tag{6}$$

This formulation is only an approximation, however. For example, while in M3 a project can accumulate more defects in rework if *Defect Rate in Rework* exceeds *Defect Correction Rate*, Model 2 does not include such a possibility. If tasks are always either correct or incorrect, we cannot make them more incorrect, and therefore the rework cycle ultimately finishes as long as there is a very small fraction of tasks that get corrected every cycle. This is also evident from Eq. 6, which only asymptotically reaches zero.

Finally, Model 1 does not include several of the parameters used in M2 and M3 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 represents the average time for discovery of an unknown problem. Conceptually, this parameter captures elements from both the testing speed and testing quality, so that a small value suggests a quick and accurate testing procedure. Yet M1 is not in parallel with M2 or M3 and therefore only an approximation can be obtained for this parameter. For this approximation we scale *Testing Time* based on the fraction of tested tasks (in M3) that are cycled back into the rework process (rather than being accepted). If this fraction is low, little rework can be discovered in Model 1; thus *Time to Discover Rework M1* will be higher, if that fraction is high, then rework can be discovered (faster) and thus *Time to Discover Rework M1* is larger:

$$\textit{Time to Discover Rework M1} = \textit{Testing Time}/(1 - e^{-\textit{Defect Rate in Completion} \times (1 - \textit{Probability of Missing a Defect})}) \tag{7}$$

Equations 5–7 are useful for deriving the parameters of one model based on empirically validated parameter values from another model. For example, if *Defective Completion Probability M2* is available from calibrating a model with M2 structure to a project's



DOI: 10.1002/sdr

data, one can use Eq. 5 to find an estimate for *Defect Rate in Completion* for M3, or vice versa. This is helpful for building on the cumulative knowledge in the literature on different parameters. We also use these relationships in the following comparative analysis to specify the parameters of M1 and M2 based on values of M3 parameters so that the models are compared in as close a setting as possible.

## Structural comparison of models

Table 2 provides a summary comparison of structural features of these formulations. For each model we discuss the level of complexity of the model, what it adds in terms of capabilities to the previous model, and the project characteristics best fitting to that formulation. We limit this summary to the capabilities of the models discussed here and not their potential capability, which may be achieved by altering these structures. M1 with two independent stocks is the simplest structure that captures the rework cycle. It can include feedback effects on quality, productivity, and resources, distinguishes between the undiscovered rework and the work that is pending, and thus provides a proxy for delivered quality as the project finishes when the majority of tasks are perceived to be done (some of them potentially being in an undiscovered defect category). M1 does not assume explicit testing, but rather considers a more implicit rework discovery process to be at work. Considering testing explicitly requires a stock and flow structure where correctly completed tasks and defective ones are separately followed, and where tasks completed wait in a stock for testing, before being approved or sent for rework. M2 provides one such formulation. Besides explicitly capturing testing and its parameters, this structure has the benefit of distinguishing rework from new work, which may be important if the time needed for rework is significantly different from that for new work. This formulation also allows for direct calculation of delivered quality, in terms of fraction of approved tasks that are defective. These additional capabilities, however, come at the cost of four more independent stock variables and a few more parameters. Finally, M3 uses a co-flow structure similar to M2

Table 2. Comparison of complexity (number of independent stocks), capabilities, and context of use among the different formulations. The capabilities are cumulative, so each model includes what is available in the previous models, plus the items discussed in its own row

| Model | Complexity | Capabilities (cumulative) | Context of use |
|---|---|---|---|
| M1 | 2 independent stocks | • Basic rework cycle<br>• Potential feedbacks to productivity, quality, and resources<br>• Capture a proxy of delivered quality | • Rework can commence at any point<br>• Project quality improves over time<br>• Shorter tail at project end |
| M2 | 6 independent stocks | • Capture the testing process, includes testing precision<br>• Allow for different rework and initial work sizes<br>• Calculate quality directly | • Approved tasks are not reworked<br>• Declining delivered quality is possible<br>• Testing and completion interlinked |
| M3 | 7 independent stocks | • Include multiple defects per task<br>• Include different test coverage levels<br>• Include different sizes for tests | • Approved tasks are not reworked<br>• Declining delivered quality is possible<br>• Multiple defects per task<br>• Long tail at project end |



DOI: 10.1002/sdr

but also allows for multiple defects per task. It captures the testing process more explicitly by following the stock and flow structure for tests, thus enabling differentiation among different levels of test coverage and different types (sizes) of tests. These capabilities come at the cost of one additional independent stock variable to capture tests explicitly. All models are available in supplementary material for independent simulation and analysis.[7]

At their core, these models differ in how they conceptualize defects and the rework discovery process. In M1, given enough time, all defects will be discovered; therefore the quality of tasks remaining in the rework cycle gradually improves. In contrast, in M2 and M3, once a defective task is missed in testing it will not be discovered later. Moreover, M1 and M2 see defects to have a one-to-one relationship with tasks, whereas M3 allows for different levels of defectiveness. One or the other conceptualization may be a closer fit for each real-world application.

## Behavioral comparison

In comparing the behavior of the three models we focus on understanding the similarities and differences of the behavior generated from these three formulations. Since this is a theoretical exercise with no empirical data to compare the models against, the differences across the models do not signal the superiority of one formulation or the other, but rather only highlights the differences in logical conclusions of different modeling assumptions. However, understanding these similarities and differences is helpful for selection of modeling assumptions most appropriate in specific application contexts. Figure 8 shows the behavior of the three parallel models. Items 1–3 report the number of tasks perceived complete (*Tasks Perceived Complete M1* in M1) or approved

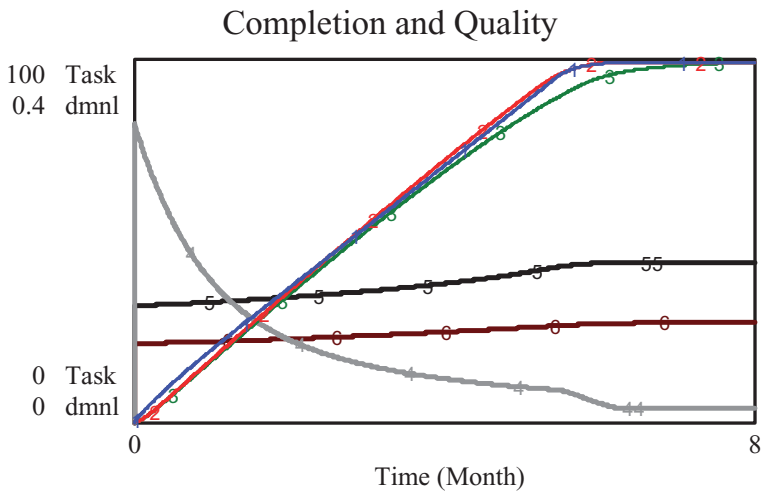


Fig. 8. Behavior of the three parallel models with the base case parameter settings. Items 1–3 report the number of tasks perceived complete (in the case of M1) or approved (in the case of M2 and M3). Items 4–6 present the *Defective Fraction in Approved Tasks* in the three models (Dimensionless, lower scale). Legend: (1) *Tasks Perceived Complete M1*; (2) *Tasks Approved M2*; (3) *Tasks Approved M3*; (4) *Defective Fraction in Approved Tasks M1*; (5) *Defective Fraction in Approved Tasks M2*; (6) *Defective Fraction in Approved Tasks M3*



DOI: 10.1002/sdr

(*Tasks Approved M2* in M2 and *Tasks Approved* in M3). These trajectories provide a quick overview of the simulated project's progress over time. Given the parallel parameterization of the models, the three act quite similarly for most of the project's life. However, towards the end, M3's progress (*Tasks Approved*) slows down due to the increasing density of defects in the remaining tasks in the rework cycle. This leads to a significant level of late rework compared to the other two formulations. As a result, M3 finishes significantly later (8.046 months) than M1 (6.272) or M2 (6.132). Models 1 and 2, with the binary conceptualization of a defect, cannot show the sharp increase in the fraction of tasks that are defective, because a defective task cannot become more defective. As a result the project ends faster in these formulations.

A more significant difference is observed in terms of delivered quality across different formulations, measured as the fraction of tasks defective among those approved/perceived complete. M1 shows a declining defective fraction as an increasing fraction of tasks in the undiscovered rework category is discovered over time. In fact, with a high enough threshold for finalizing the project, this formulation would approach perfect delivered quality as the undiscovered rework stock is drained through a first-order discovery process. On the other hand, M2 shows the highest fraction of defective approved tasks, i.e. lowest delivered quality. Only a fraction of defective tasks are caught owing to imperfect testing (*Probability of Missing a Defect* = 0.3, See Table 1), and as a result the rest of defective tasks are approved and never discovered again. As the fraction of defective tasks among those in the rework cycle increase towards the end of the project (because correct tasks are approved, leaving the more problematic ones in the cycle), the quality of delivered tasks also deteriorates at the end. The M3 formulation follows a similar logic, but the testing process catches tasks based on their number of defects. As a result, fewer defective tasks get approved. For example, a task with two defects has only a probability of 0.09 to be approved (the probability that both defects are missed = $0.3 \times 0.3$)—much less than the 0.3 probability for a defective task in M2. Similarly, more defective tasks are even less likely to be approved until they are cleared, leading to higher final delivered quality for a project modeled using M3.

Another, more subtle, mechanism is also at play here. In M3, the perfect mixing assumption dictates the Poisson distribution for the number of defects per task. However, in reality tasks with no defects are accepted in the testing process and go through without any problem. The remaining tasks therefore do not include defect-free ones, whereas the Poisson distribution in M3 assumes that such flawless tasks exist even in those pending rework. Thus, given a value for average number of defects per tasks cycled through the rework process, M3 will include more tasks with no defects and thus (to keep the average unchanged) more with multiple defects, compared to reality. In short, M3 increases the heterogeneity in the number of defects per task by assuming a Poisson distribution. The increased heterogeneity leads to a lower fraction of tasks that pass through the testing process in the next round of testing. This is because the tasks assumed flawless have no defect and they pass through, while the more defective tasks are likely not to pass the testing. An example illustrates this mechanism better. Consider two sets of two tasks with one defect per task, on average. The first set includes a task with no defects and a task with two defects (higher heterogeneity, more similar to M3). The second set includes two tasks with one defect each (potentially more similar to M2). If these two sets go through a testing process with probability of $p$ for missing a defect, then accepted tasks from the first set will have an expected number of defects



DOI: 10.1002/sdr

equal to $p^2$, while this number is equal to $2p$ for the second set. For all feasible values of $p$ (between 0 and 1), the defect density in accepted tasks will be lower for the first set, which resembles M3.

Overall, in the base case analysis the three models show significant similarity in terms of project progression. M3 shows a longer tail of rework due to increased rework activity on a few problematic tasks towards the end of the project. The delivered quality metrics show more significant variations due to differences in the assumed testing/rework discovery processes and the probability of missing defective tasks. These differences provide additional guidelines for selecting the more appropriate formulation for a given project context: where defects from any of the tasks of the project can be discovered at any point and reworked, then the declining defective fraction is likely and M1 may be more realistic. When tasks, once approved, are not typically reworked, then the defective fraction is more likely to increase towards the end of the project, making M2 and M3 more likely candidates. The last column in Table 2 specifies some signature characteristics of a project that signals a better fit for application of each formulation. For example, in the application of M3 to a software project we did observe an increasing fraction of defective tasks remaining in the rework cycle towards the end of the projects. Given the widespread use of M1 and application track record of M2, one can expect each of the three formulations to be a best fit for some projects.

### Project characteristics and alternative formulations

Different projects may differ significantly in their structures and parameter ranges. We therefore conduct a sensitivity analysis to understand the relative behavior of different models in alternative project settings and inform the selection of an appropriate formulation depending on project settings. We change the model parameters from the base case values on a large interval to see how the models compare under a wide range of assumptions. These values and results of the analysis are reported in Table 3.

The sensitivity results further establish the mechanisms that distinguish the different model formulations. For example, the major reason for the long tail of M3, compared to M1 and M2, is the late reinforcement of the rework cycle due to a few highly defective tasks. Therefore an increasing defect rate in completion significantly widens the gap between M3 and M2 as M3 lasts much longer than before (M3 ends after 20.988 vs. 8.510 for M2, in Table 3, row 7). Interestingly, in this case M1 is also lasting much longer (20.807 in Table 3) because the increased defect rate in completion reduces *Time to Discover Rework M1*. Whereas in M2 about 30 percent of defective tasks are approved due to imperfect testing (leading to much lower delivered quality for M2), multiple defects per task in M3, and lack of testing in M1, lead to longer rework and an increased finish time with less compromise on delivered quality. Similarly, increasing the *Defect Rate in Rework* (or decreasing the *Defect Correction Rate*; see rows 9 and 10) increases the finish time for M3 (to 26.745) much more than it does for M2 or M1. In this case M1 is not affected because this parameter does not exist in M1. Moreover, the impact on the quality differential is less pronounced when we increase the defect rate in rework because the delivered quality of M2 does not deteriorate sharply owing to the relative high quality of the ~90 percent of the tasks that are approved without going through rework. In M3, if defect correction becomes slower than the defect generation rate in rework and testing is not missing a large part of the defect, the project will not finish, while in M2 and M1 such projects will still finish.



DOI: 10.1002/sdr

Table 3. Detailed results of sensitivity analysis based on the change of each parameter/simulation setting listed in Table 1. Numbers are in bold when they change more than 80% from the base case

| Metrics simulation parameters | *Finish Time M1* | *Finish Time M2* | *Finish Time M3* | *Defective Fraction in Approved Tasks M1* | *Defective Fraction in Approved Tasks M2* | *Defective Fraction in Approved Tasks M3* |
|---|---|---|---|---|---|---|
| 1 Base case | 6.272 | 6.132 | 8.046 | 0.015 | 0.176 | 0.110 |
| 2 *Probability of Missing a Defect* = 0.1 | 6.254 | 7.402 | 8.926 | 0.011 | 0.076 | 0.033 |
| 3 *Probability of Missing a Defect* = 0.9 | 6.342 | 4.410 | 4.457 | **0.122** | 0.316 | **0.315** |
| 4 *Testing Time* = 0.05 | 6.223 | 5.983 | 7.645 | 0.006 | 0.176 | 0.110 |
| 5 *Testing Time* = 0.4 | 6.475 | 7.410 | 10.964 | **0.076** | 0.176 | 0.109 |
| 6 *Defect Rate in Completion* = 0.1 | 4.467 | 4.832 | 4.918 | **0.029** | 0.049 | 0.042 |
| 7 *Defect Rate in Completion* = 1.6 | **20.807** | 8.510 | **20.988** | 0.007 | **0.430** | 0.133 |
| 8 *Defect Rate in Rework* = 0.1 | 6.272 | 5.832 | 6.191 | 0.015 | 0.152 | 0.098 |
| 9 *Defect Rate in Rework* = 0.7 | 6.272 | 6.434 | **26.745** | 0.015 | 0.199 | 0.112 |
| 10 *Defect Correction Rate* = 0.5 | 6.272 | 6.467 | **26.745** | 0.015 | 0.202 | 0.112 |
| 11 *Defect Correction Rate* = 1.6 | 6.272 | 5.762 | 5.652 | 0.015 | 0.146 | 0.087 |
| 12 *Initial Tasks to Complete* = 20 | 2.639 | 2.277 | 4.012 | 0.015 | 0.176 | 0.108 |
| 13 *Initial Tasks to Complete* = 500 | **29.340** | **28.034** | **32.957** | 0.007 | 0.176 | 0.110 |
| 14 *Productivity* = 2 | **14.676** | **14.210** | **17.059** | 0.012 | 0.176 | 0.110 |
| 15 *Productivity* = 10 | 3.793 | 3.600 | 5.342 | 0.015 | 0.176 | 0.109 |
| 16 *Schedule Pressure impacts Quality* | 6.738 | 6.424 | 9.527 | 0.015 | 0.207 | 0.121 |

*Probability of Missing a Defect* has a more subtle impact on the behavior of the models: on both extremes, i.e. perfect testing and testing that catches no defects, M2 and M3 largely converge: without testing, every defective task is approved and rework is largely irrelevant. At this extreme M1 also converges to the other two because *Time to Discover Rework M1* increases significantly (see Eq. 7), leading to little discovery of defects and thus rework. However, if the probability is at 0.9, then M1 finishes somewhat *later* (see row 3) because low discovery of rework reduces tasks not completed, limiting the completion rate due to availability of tasks, and thus extending the project somewhat longer as the 99 percent completion line is reached later. When tests are perfect (*Probability of Missing a Defect* = 0), only truly defect-free tasks are accepted, and therefore the strength of the rework cycle for M2 and M3 will again be very similar and both models will take longer to finish the project (M2 ends after 8.355 and M3 ends after 9.217) with perfect delivered quality scores. Between both extremes, M2 and M3 diverge.

We also test the impact of including a reinforcing loop that is typically discussed as an important contributor to problematic project performance, i.e. the reinforcing loop created from the impact of schedule pressure on error rate. Increased schedule pressure is often observed to lead to increased error rate, thus reducing the net task approval rate and further contributing to increased pressure. This dynamic is widely recognized and analyzed in system dynamics models (e.g. AbdelHamid and Madnick, 1991; Eden *et al.*, 2000; Taylor and Ford, 2006) and therefore provides a good test case to assess the reactions of different models to different feedback effects. We model schedule pressure as the ratio of expected time remaining to complete a project (calculated based on remaining tasks and productivity) to scheduled remaining time (floored at a minimum of one month). Schedule pressure then impacts both completion and rework defect rates.



DOI: 10.1002/sdr

A simple table function relates schedule pressure to defect rate here. We assume defect rate linearly increases from its normal value to 50 percent higher as schedule pressure increases between 1 and 1.5, and saturates at this level. We find that M3 is slightly more sensitive to this feedback effect because the increased defect rate due to schedule pressure further exacerbates the intensity of the rework cycle due to a few highly defective tasks towards the end of the project. On the other hand, the quality of approved tasks in M2 is compromised most significantly due to schedule pressure feedback.

Finally, in comparing the three formulations we should note that the interdependencies of the testing process with task completion are not explored here, but may have an impact on relative behavior of different formulations.[8] Three mechanisms are relevant in this regard. First, testing often competes with completion for resources and thus can constrain available completion resources in M2 and M3, but not in the current version of M1. Second, the start of new task completion can be constrained by the amount of work currently completed or tested. Finally, testing itself can be constrained by the amount of work completed or tested. Alternative behavior modes may arise due to these interactions. For example, slow testing coupled with testing-constrained development can significantly slow down the initial rate of completion, even if the desired development resources are available, leading to an S-shaped growth in project trajectory. Detailed analysis of these interactions can show other differences across these models.

## Discussion and conclusions

In this paper we develop a new formulation for modeling the rework cycle in projects. This formulation extends the available rework cycle formulations by allowing the modeling of multiple defects per task. We show the underlying distributional assumptions that allow for the development of this formulation and test its robustness under different parameter values. We also compare this formulation with two established models of the rework cycle from the literature. We find how the parameters of these models are related to each other and how their structure and behavior compare. The following discussion summarizes our main findings and their applications for modelers. Finally we discuss the limitations of the current work and potential avenues for further research.

The basic mode of behavior for the new rework cycle model is similar to that of other models and the typical empirical patterns. It consists of the growth in the number of tasks approved as resources work on tasks, until the project ends. The final stages of the project can, however, be significantly slower when we consider the possibility that some tasks may include more than one defect. Such tasks may cycle through the rework process multiple times and delay the project. The difference in finish time between M3 and M2 or M1 may be of theoretical and practical significance. Project finish time is often a very important factor in project success and profitability, as lost revenue or penalties accumulate with delays. Moreover, previous research has attributed the 90 percent syndrome—the lengthy continuation of projects that are almost finished—to managerial responses to schedule pressure that increase the error rate (Ford and Sterman, 2003; Taylor and Ford, 2006). Our results suggest that a more basic process, the iteration of a few tasks with multiple defects, may create similar symptoms, without requiring any changes in the defect rate, or involving other feedback mechanisms. The traditional formulations used in system dynamics for modeling projects hide this effect



DOI: 10.1002/sdr

because they do not allow for multiple defects per task. Whether a long project tail is due to the feedback effects discussed in previous research or the mechanism highlighted here is an empirical question that should be analyzed for each specific project.

On the other hand, the measures of delivered quality among M1, M2 and M3 show significant differences. The different conceptualization of defect discovery and rework in M1 explain some of this difference, as all defects can be discovered at the end of the project in this formulation (see "Structural comparison of models", above). The mechanism that leads to higher delivered quality in M3 is, however, more subtle. M3 requires a project to go through more cycles of rework, owing to tasks with multiple defects that are often caught in testing. When a project includes such problematic tasks, M3 may prove to be a more appropriate formulation to capture the dynamics of the project. If acceptance of tests, however, depends not on the number of defects but on the relative quality of the task, then M2 could provide a more precise representation. With appropriate parameterization M1 also behaves reasonably close to the other two alternatives and therefore can be an appropriate choice when model simplicity is an important criterion.

Another reason for adopting M3 is the structure of available data. When data on tasks, defects, and tests come from different sources and with different units, the one-to-one relationship assumed among these in traditional formulations may not hold. For example, there is no one-to-one relationship between a line of code, a defect, and a test in software development. Estimating model parameters in such settings can potentially be done more successfully using M3 or a variation of it that has assumptions most closely resembling the data structure.

Many project studies involve theoretical and managerial insights about the impact of alternative feedback effects on modes of behavior in projects. The formulations compared in this paper provide slightly different opportunities for expansion and inclusion of such alternative feedbacks. M1 can include feedbacks to quality and productivity, as well as defect discovery rate. M2 includes additional flexibility to capture feedbacks to testing quality. Finally, M3 allows for feedback effects that manipulate test sizes (testing one or multiple tasks, e.g. unit testing vs. integration testing), test coverage, and rework quality. With the exception of a robustness test on the effect of schedule pressure on quality, we did not study any of these feedback effects in the current paper. Capturing such feedback effects is at the heart of most system dynamics studies and our formulation provides an additional building block for building such feedback-rich models in diverse applications.

Overall, selection of the appropriate rework cycle formulation depends both on the purpose of a modeling project and the project setting being modeled. Sensitivity analysis in this paper provides initial guidelines. In the absence of defects, or with very poor testing, the rework cycle is largely removed and all models behave in very similar patterns. Moreover, a very effective rework process (high defect correction rate) reduces the impact of the rework cycle and the differences across models. Under these settings picking the simplest model (M1) is often a wise choice. As defect rate and testing precision increase, or rework effectiveness reduces, the rework cycle is strengthened and the models are more likely to diverge. Under these conditions we should decide between the models depending on the nature of defects, testing process, and the availability of data. If defects are closely tied with tasks or tests may pass a task with some defects, as long as they are not very problematic then M2 is more appropriate. When tasks and



DOI: 10.1002/sdr

defects are seen as different concepts and passing tests depends on the number of defects in a task, then M3 is a better alternative.

The derivation of the new formulation also provides a blueprint for tackling other modeling challenges, where the progression of units in split branches of a stock and flow network is determined by some characteristic of those units captured in a co-flow. For example, in modeling a faculty aging chain, progression from assistant to associate, and associate to full professor depends on the number of publications of faculty in each stock (among other things). Those who are promoted have a different average publication record than those who are not. This is conceptually similar to our project model, where tasks with more defects are more likely to be sent to rework, and those with fewer defects to be approved. A similar concept can be applied to modeling the selection of credit-worthy home buyers (households and their credit being the co-flows, and the decision to lend being the split factor) or consulting promotion chains, among others.

Overall, this study introduces a new formulation that expands the toolbox for project modeling. Moreover, we conducted multiple tests to build confidence in the usefulness of the new formulation, including structure assessment, dimensional consistency, extreme condition, integration error, and several comparisons with established formulations to test for behavior reproduction and observe anomalies. However, ultimately the value of new formulations should be established in multiple real-world applications to different problems. The adoption of this formulation by modelers for their diverse applications will help with its further assessment and improvement. Future research can also compare and contrast this formulation against alternatives for calibration to empirical project data to assess similarities and differences and learn about potential estimation challenges. Comparative studies can also be conducted against more detailed, agent-based models (e.g. see Rahmandad and Sterman, 2008) to understand how well the multiple defects per task captured in this formulation tracks an agent-based model with the same conception of defects. Such comparative studies could be extended to other rework cycle formulations to shed light on the relative costs and benefits of models at different levels of aggregation to simulate project dynamics. Furthermore, M3's analytical formulation rests on the assumption of a Poisson distribution for the number of defects per task. This assumption may in reality be violated; for example, the testing process removes all the defect-free tasks from the cycle. More research is needed to assess the impact of non-Poisson distributions.

In this research we focused on a central, but narrowly defined, piece of project dynamics and compared multiple formulations. Given this focus, we excluded many factors important to project dynamics such as schedule pressure, morale, experience, communication, and other feedback effects and interactions among different phases of a project, among others. Yet the controlled experiments enabled by such a focus move us closer to a set of contingency rules of thumb for the selection of appropriate formulations for specific project modeling problems. Depending on the structure and amount of available data, the conceptualization of defects and testing process, the level of quantitative precision required, and the client's main goals, we find that all these formulations could be appropriate in some settings. Future research can offer new formulations that build on this analysis to address the shortcomings of current models. One can also study the effect of different precedence relationships and task priorities, as well as additional feedback effects, on the comparative performance of these and other formulations. Despite these shortcomings, this research makes several contributions to this



DOI: 10.1002/sdr

important area of system dynamics applications: it captures multiple defects per task, allows for variable testing types and lower-than-complete test coverage, provides an alternative hypothesis for explaining well-established project behavior modes, allows for better integration of data and model in some types of projects, and can be applied to other formulation challenges in diverse areas.

## Notes

1. Assuming a constant scope, the total number of tasks in the project, which is the sum of three stocks, is constant, and thus one of the stocks can always be calculated based on the other two, leading to two independent stocks.
2. The simulation model for this paper is publicly available from http://filebox.vt.edu/users/hazhir/www/papers/ReworkCycleFormulation/PD_M123_Final.mdl.
3. This equation is directly taken from the Poisson probability distribution, where the probability of observing $k$ events from a process with mean $m$ is
$$P(\#\text{Event} = k) = \frac{e^{-m}m^k}{k!}.$$
4. 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 a dynamic test coverage.
5. For a Poisson distribution with mean $M$ defects per task, the probability that no defect exists is $e^{-M}$.
6. M2 could also be adapted to account for this possibility.
7. All models are implemented in Vensim™.
8. We thank one of the anonymous reviewers for highlighting these possibilities.

## Acknowledgements

We would like to thank Jim Lyneis and three anonymous reviewers at the 2008 System Dynamics Conference for their valuable feedback. We also thank Thanujan Ratnarajah for his excellent research support in this project. David Ford and three anonymous reviewers provided great feedback on the previous versions of this paper. All errors are the authors' responsibility.

## Biographies

Hazhir Rahmandad is assistant professor of Industrial and Systems Engineering at Virginia Tech. He received his Ph.D. in system dynamics from the Massachusetts Institute of Technology and his undergraduate degree in Industrial Engineering from Sharif University of Technology, Iran. Hazhir's research interests include understanding the evolution of product development capability, systems engineering dynamics, organizational learning and change, health applications of system dynamics, and comparing different simulation methods.

Kun Hu is pursuing her Ph.D. degree in Grado Department of Industrial and Systems Engineering at Virginia Tech. She received her B.S. and M.E. degrees in Electrical Engineering from Xi'an University of Technology and Xi'an Jiaotong University in 2004 and 2007, respectively, and her



DOI: 10.1002/sdr

M.S. degree in Industrial and System Engineering from Virginia Tech in 2009. Her research interests include system dynamics and agent-based modeling of public health problems.

## References


AbdelHamid T, Madnick S. 1991. *Software Project Dynamics: An Integrated Approach*. Prentice-Hall: Englewood Cliffs, NJ.

Cooper KG. 1980. Naval ship production: a claim settled and a framework built. *Interfaces* **10**(6): 20–36.

Cooper KG. 1993a. The rework cycle: benchmarks for the project manager. *Project Management Journal* **24**(1): 17–21.

Cooper KG. 1993b. The rework cycle: how it really works . . . and reworks . . . . *PM Network Magazine* February: 25–28.

Cooper KG. 1993c. The rework cycle: why projects are mismanaged. *PM Nework Magazine* February: 5–7.

Eden C, Williams T, Ackermann F, Howick S. 2000. The role of feedback dynamics in disruption and delay on the nature of disruption and delay (D&D) in major projects. *Journal of the Operational Research Society* **51**(3): 291–300.

Ford DN, Sterman JD. 1998. Dynamic modeling of product development processes. *System Dynamics Review* **14**(1): 31–68.

Ford DN, Sterman JD. 2003. Overcoming the 90% syndrome: iteration management in concurrent development projects. *Concurrent Engineering: Research and Applications* **11**(3): 177–186.

Grimstad S, Jorgensen M, Molokken-Ostvold K. 2006. Software effort estimation terminology: the tower of Babel. *Information and Software Technology* **48**(4): 302–310.

Lee S, Pena-Mora F. 2007. Understanding and managing iterative error and change cycles in construction. *System Dynamics Review* **23**(1): 35–60.

Lyneis JM, Ford DN. 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 essays on modeling dynamic organizational processes*. PhD thesis, Sloan School of Management. Cambridge, Massachusetts Institute of Technology.

Rahmandad, H, Sterman J. 2008. Heterogeneity and network structure in the dynamics of diffusion: comparing agent-based and differential equation models. *Management Science* **54**(5): 998–1014.

Rahmandad H, Weiss D. 2009. Dynamics of concurrent software development. *System Dynamics Review* **25**(3): 224–249.

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

Richardson GP, Pugh AL. 1981. *Introduction to System Dynamics Modeling with DYNAMO*. MIT Press: Cambridge, MA.

Sengupta K, AbdelHamid TK. 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: New York.

Stutzke MA, Smidts CS. 2001. A stochastic model of fault introduction and removal during software development. *IEEE Transactions on Reliability* **50**(2): 184–193.

Taylor T, Ford DN. 2006. Tipping point failure and robustness in single development projects. *System Dynamics Review* **22**(1): 51–71.

Taylor TRB, Ford DN. 2008. Managing tipping point dynamics in complex construction projects. *Journal of Construction Engineering and Management ASCE* **134**(6): 421–431.




DOI: 10.1002/sdr