chapter 4

The Rework Cycle

I haven't failed, I've found 10,000 ways that don't work.
—Thomas Alva Edison[31]

The only man who never makes a mistake is the man who never does anything.
—Theodore Roosevelt

I. What Is The Rework Cycle?

The Rework Cycle, as defined in the discipline of System Dynamics[32], is a key concept that must be considered in developing an accurate representation of the processes involved in engineering and managing complex system development programs. It was first developed explicitly by Cooper (1980)[33] and has subsequently been adapted by many other authors.[34] Explaining the Rework Cycle, Richardson and Pugh note:

> Not all work done in the course of a large project is flawless. Some fraction of it is less than satisfactory and must be redone. Unsatisfactory work is not discovered right away, however. For some time it passes as real progress, until the need for reworking the tasks involved shows up. Hence, we are led to suggest that the project workforce accumulates two kinds of progress: real and illusory. We shall term the latter undiscovered rework. Together, cumulative real progress and undiscovered rework constitute what is

[31] The author also found some sources that attribute this to Benjamin Franklin.
[32] System dynamics was pioneered by Jay Forrester. His first book on the subject is *Industrial Dynamics*, Portland, OR: Productivity Press, 1961.
[33] Cooper, Kenneth G. Naval Ship Production: A Claim Settled and a Framework Built, *Interfaces*, 10:6, December 1980.
[34] Ford and Sterman note several papers: Cooper, 1993a, b, c, 1994; Seville and Kim, 1993; Jessen, 1990; Kim, 1988; Pugh and Richardson, 1981; Abdel-Hamid, 1984; Ford and Sterman, 1997; Ford, 1995.

> perceived within the project as actual cumulative progress. . . .[35]

Understanding the Rework Cycle is key to understanding the dynamics of complex system development. In the context of simulating the dynamics of a real software development program, Cooper comments:

> . . . we had to simulate the performance of actual projects as they really did occur — not just how they were planned to go, or how they should have gone. . . . To do so we had to explicitly address their substantial rework, as well as its causes, detection, and execution.[36]

The basic rework cycle is depicted in Figure 4.1. It begins with a quantification of the work that must be performed, much like developing a cost estimate based upon an understanding of the scope of tasks involved. All of the planned tasks are in a bucket, as it were, waiting to be performed. The work generation activities begin to expend energy and resources to perform the tasks in the "Work to Do" bucket. Once the work is performed it is allocated to the "Work Done" bucket. But herein lies the problem: some of the tasks that are thought to be done, are not actually done. Some fraction of the work thought to be done must be redone. That portion of the work goes, by definition, into the undiscovered rework bucket.[37]

The quality of the work performed is a key factor that drives the amount of rework in any given System Development activity. Quality, in this context, is a measure of that portion of work performed that is actually completed, as opposed to that which is thought to have been completed. Because the quality of the work is not perfect, some of the work that is thought to be done is not actually done. That work is allocated to the rework bucket.

There is a further difficulty as well. Exactly which tasks are done and which need to be redone? In other words, some of the rework is known: that is, some tasks are known to be incomplete or otherwise problematic. However, as Richardson and Pugh note in the above quote, there will likely be tasks that certainly require rework but which have not yet been discovered. The rework bucket contains both known and undiscovered rework.

Known rework can be intelligently managed because it is known. However, undiscovered rework injects risk into the program precisely because it is not known. It is not a trivial task to manage something that is not known. What is the potential cost, schedule, and technical impact? How extensive

[35] George P. Richardson and Alexander L. Pugh, *Introduction to System Dynamics Modeling*, Portland, OR: Productivity Press, 1981, pp. 56-57.

[36] Cooper, Kenneth G. and Thomas W. Mullen, Swords and Plowshares: The Rework Cycles of Defense and Commercial Software Development Projects, *American Programmer*, May 1993, p. 42.

[37] The buckets labeled "Undisc RW" in Figures 4.1 and 4.2 contain both known and undiscovered rework. There is no need to model these separately since the tasks in each bucket must be redone and are thus handled in the same way in the model.

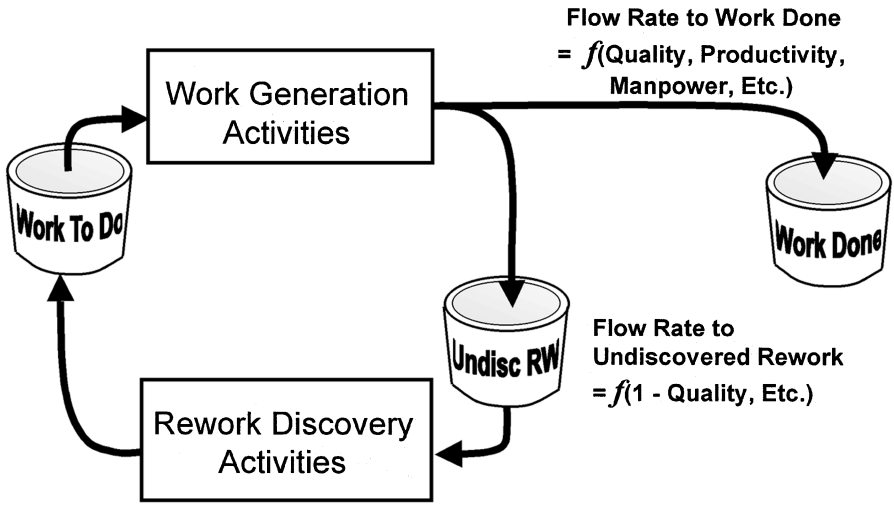

Figure 4.1 The Rework Cycle.

is it? What is the risk to the company and to the customer? Many more issues could be raised, but the key point here is that undiscovered rework is a potentially dangerous thing for all stakeholders. The implication to the system development process is this:

Key Point

> Specific activities must be included in the SDF that are aimed at exposing undiscovered rework in order to mitigate its potentially adverse effects.

Figure 4.1 illustrates a single rework cycle or "phase." The term "phase" in this context refers to a complete rework cycle, unlike its previous usage where it describes different brackets of time along a program timeline. Figure 4.2 depicts a system made up of multiple Rework Cycles, or multiple phases. For simplicity, the activities that compose the Requirements Development activity are coalesced into one phase; similarly, those composing the synthesis activity. Thus, as illustrated here, the system-level portion of the development comprises two Rework Cycle phases. The subsystem tier, which comprises similar Requirements Development and Synthesis activities is depicted with a single Rework Cycle phase. This illustration shows that the fidelity of the output of Requirements Development is a function of the quality of the work performed in its set of activities. This output becomes the input to the system-level Synthesis activity. The quality of the output of the Synthesis activity is limited by the quality of the input data.

The objective here is to illustrate how poor quality ripples through the entire system development in a non-linear fashion. For the purposes of this

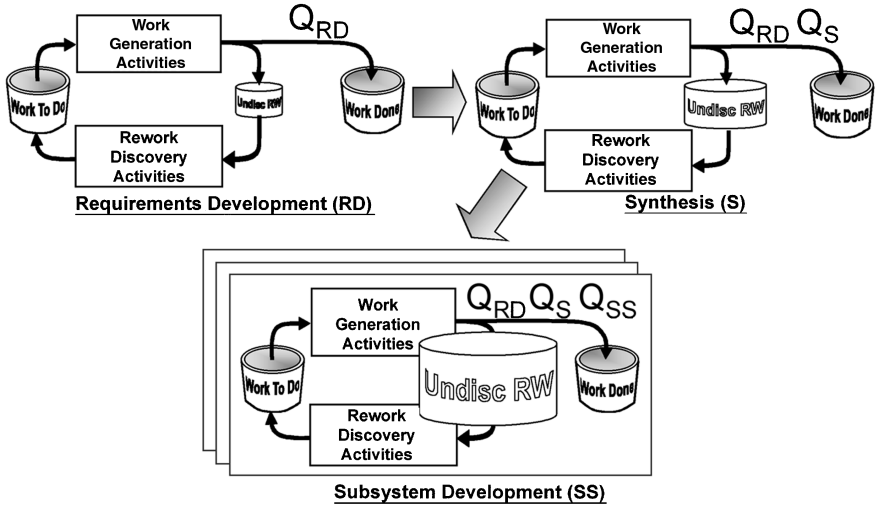

Figure 4.2 The Rework Cycle in Multiple Phases.

illustration, only three phases are shown for an entire System Development program. The rationale for the three phases is that requirements drive the Synthesis activity and the Synthesis activity drives the lower-level development activities. In reality, each sub-activity could be modeled as a rework cycle, each with its own quality level. In a real program there would likely be many more levels in the hierarchy.

It is apparent that even small lapses in quality at upper levels in the hierarchy can have a devastating effect on the quality achievable at lower levels in the hierarchy. Because of feedback effects, this influences quality at upper levels as well. A vicious cycle begins to emerge.

Key Point

It is clear from this discussion that improving the quality of work performed should be a major priority in sound program management. The amount of work performed that is actually complete is directly related to the quality level of the work. As the quality level goes down, the amount of rework that will be required necessarily increases at a non-linear rate.

Key Point

This discussion highlights a key difference between complex system development and relatively simple product development. In complex systems, rework at lower levels grows exponentially. The impact of this effect is not as significant in relatively simple products.

II. A Simple System Dynamics Model

In order to illustrate the impact of various parameters on the success of a system development program, a simplified System Dynamics model will be employed. The model includes three phases as mentioned in the discussion concerning Figure 4.2. Some of the key parameters include: quality of the work performed, productivity of the workers performing the task, and the level of effort applied to discovering rework as a percentage of the total staff level. Certainly, more detail could be added with profit, but for the purpose of this discussion, this model will be adequate. When the output is examined, a focus on the absolute numbers is not the primary concern; rather, the central issue has to do with the trends they indicate.

In this context, productivity is simply the rate at which the work is being accomplished. It is a measure of how efficiently allotted time is being used. Quality, in this context, refers to the amount of work actually completed as a fraction of the amount of work thought to be complete. As discussed above, not all the work performed is actually complete. Some fraction becomes either discovered or undiscovered rework.

In the System Dynamics model three variables for quality are employed. The first is "nominal quality," $Q_{nominal}$, which represents the quality of the work as it is being performed by the people or machines. In the model, this number does not change. It is assumed that a fixed percentage of the work performed goes to the rework bucket. The second variable for quality is called "average quality." Once the program starts, it is calculated. Average quality, Q_{avg}, is defined as:

$$Q_{avg} = \frac{Work\ Done}{Work\ Done + Rework} \tag{4.1}$$

As the program progresses, Q_{avg} approaches unity because the work done bucket is filling up and rework is being discovered and thereby eliminated from the rework bucket. The third variable is quality, Q. It is a function of the nominal quality of the phase in question times the average quality of the input data. Thus, each phase starts at a quality level, Q, equaling its nominal quality times the quality of the input data. This is illustrated in Figure 4.3 which is an output of the System Dynamics model. The horizontal axis represents time, while the vertical axis represents the quality level on a scale of zero to one.

The initial quality level, $Q_{nominal}$, for each phase was defined as 50%. The quality of the first phase remains constant because, in this simplified model, it is not dependent upon downstream phases for its input.[38] However, the quality level of the output of the first phase is the average quality level or $Q_{Phase\ 1\ avg}$. The quality level of the Phase 2 activity is calculated as follows:

[38] It is recognized that this is a simplification. In real life there would be feedback from the downstream activities. Such feedback is developed in Chapter 5.

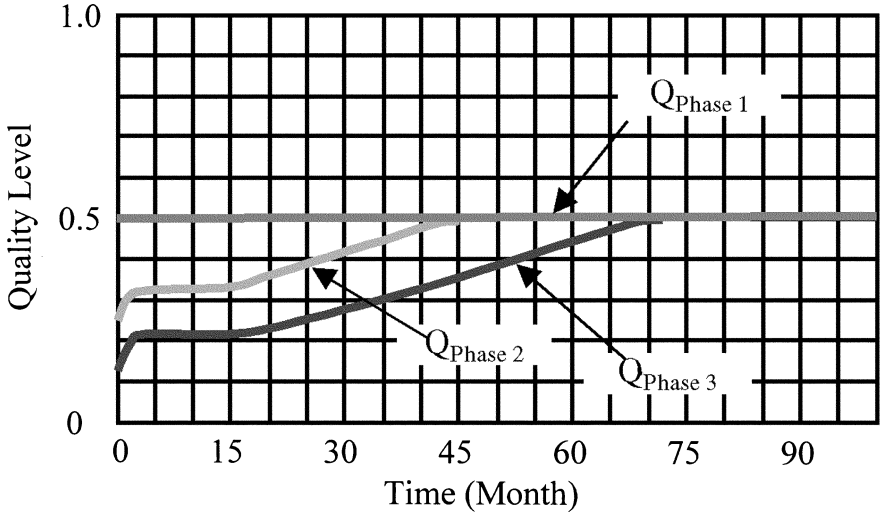

Figure 4.3 Dynamics of Quality over Time.

$$Q_{Phase\ 2} = Q_{nominal} Q_{Phase\ 1\ avg} \tag{4.2}$$

This is because the Phase 2 activity is dependent upon the output of Phase 1 for its input. The instantaneous quality of the output of Phase 1 is its average quality at the particular instant. Therefore, as shown in Figure 4.3, the initial quality level for the Phase 2 activity is 25% (50% × 50%). It rises to the nominal value over time as rework is discovered in Phase 1 and converted to work complete, thus raising the quality level of the input to Phase 2. Likewise, for Phase 3 the initial value is 12.5% (50% × 50% × 50%).

Key Point

The obvious assumption here is that the Phase 1 activity continues to discover rework in order to convert it to work complete. If this does not happen, the quality level of the input to Phase 2 does not improve and the quality level of the Phase 2 work stagnates at 25%. This represents a classic "Pay me now or pay me later" scenario. Poor quality work that is not corrected early will cause many orders of magnitude more serious problems in the future. In terms of the model, if the average quality of Phase 1 is not improved, the model will not converge — it will not be able to finish because not all the work is complete. How much more on a real program?

Key Point

> This discussion highlights an issue regarding the optimal timing of the release of output data from one activity as input data to subsequent activities. Should the data be input to the next activities as soon as it is available or should the rework activities be given time to improve the data? The answer to this question involves an assessment of the quality of the output data. The better the data, the sooner it makes sense to pass it downstream. The poorer the quality of the data, the more rework it will generate in subsequent activities and thus increase overall program costs (Garbage in → Garbage out).

Returning to Figure 4.3, it can be seen that as more phases are added, or the more tiers in a complex hierarchy, the more rapidly quality deteriorates at lower levels. It can also be seen that the schedule is pushed farther to the right. This is apparent since the horizontal axis represents time and the phase is not completed until the quality level reaches its nominal value. The quality level of Phase 3 at any given instant in time is calculated as follows:

$$Q_{Phase\ 3} = Q_{nominal} Q_{Phase\ 1\ avg} Q_{Phase\ 2\ avg} \tag{4.3}$$

where the values of $Q_{nominal}$, $Q_{Phase\ 1}$, and $Q_{Phase\ 2}$ are the instantaneous values at the time $Q_{Phase\ 3}$ is calculated. If k equals the number of phases in the hierarchy, then a pattern emerges whereby:

$$Q_{Phase\ n+1} = Q_{nominal} \left(\prod_{n=0}^{k} Q_{Phase\ n\ avg} \right) \tag{4.4}$$[39]

If all the phases have the same quality level, as it is assumed in this model, then:

$$Q_{Phase\ n+1} = Q_{nominal} Q_{avg}^{\ n} \tag{4.5}$$

Of course, Equation 4.5 is only true for the initial value of quality since after time equals zero, each Q_{avg} changes at a different rate.

Exponential Rework Growth — The aforementioned System Dynamics Model is used here to briefly examine the effects of rework as it ripples

[39] Equation 4.4 assumes that for $n = 0$, $Q_{Phase\ n\ avg} = 1$, such that $Q_{Phase\ n+1} = Q_{Phase\ 1} = Q_{nominal}$

through this simple three-phase hierarchy.[40] Figure 4.4 depicts the amount of rework generated by each phase as a function of time, with a nominal quality value of 50%. For Figures 4.4 through 4.6 the horizontal axis represents time, and the vertical axis represents the number of tasks waiting to be performed. It is clear from the figure that rework is growing in a non-linear fashion. It is also clear from this view that the schedule is being pushed to the right in a significant way. This is indicated because each phase is completed when all of the tasks have been completed; in order for all the tasks to be completed, all of the rework must be emptied from the rework bucket. Thus when the rework bucket remains empty, the phase is completed.

Note that as the results of the simulations performed are presented, both cost and schedule impacts are identified. The schedule impacts are observed directly from the figures since time is the unit of measure along the horizontal axis. Cost is a function of the number of tasks that must be performed. Therefore, an increase in the number of tasks that must be performed indicates a commensurate increase in cost. Such is the case with rework. Rework represents tasks that were performed at least once, but for various reasons, must be redone.

Note in the following discussion that the effort applied to generating work, or the Potential Work Rate (units = tasks per month), is fixed by the number of staff (units = persons) available and their respective productivity (units = tasks per month per person). For all simulations the number of staff generating work is fixed at five people. The potential work rate for Rework Discovery activities is determined by multiplying the Potential Work Rate for work generation activities by the fixed percentage identified in each simulation. For example, if the effort applied to Rework Discovery activities is set at 20% of the effort applied to generating work, then the effort applied to discovering rework would be 20% times five people times the productivity level.

Figure 4.5 provides a similar view of the program, but with a nominal quality level of 70%. The contrast between these two emphasizes the need to minimize rework generated. Rework still cascades non-linearly through the hierarchy. However, its magnitude is reduced significantly as quality is improved.

Figure 4.6 illustrates rework growth at a quality level of 90%. Notice that the scale of the vertical axis has been changed relative to Figures 4.4 and 4.5. Both Figures 4.4 and 4.5 have a vertical axis scale of zero to 40. Figure 4.6 has a vertical axis scale of zero to 4, indicating that the rework generated with a quality level of 90% is negligible as compared to quality levels of 70% and 50%. Notice also the schedules of the three figures. Completion of the phase is indicated when the bucket containing the remaining undiscovered rework tasks reaches zero. With a quality level of 90% the program finishes

[40] See Appendix F for a description of the model itself. The effects of schedule pressure, staffing, etc. have not been modeled.

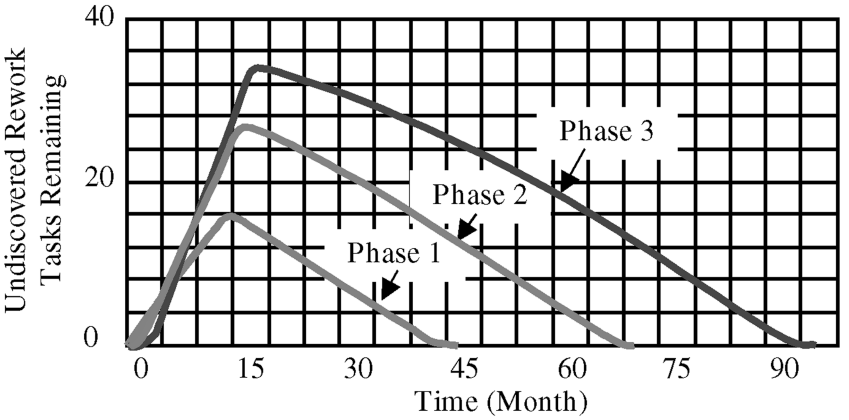

Figure 4.4 Rework Growth as a Function of Number of Phases (Quality = 50%).

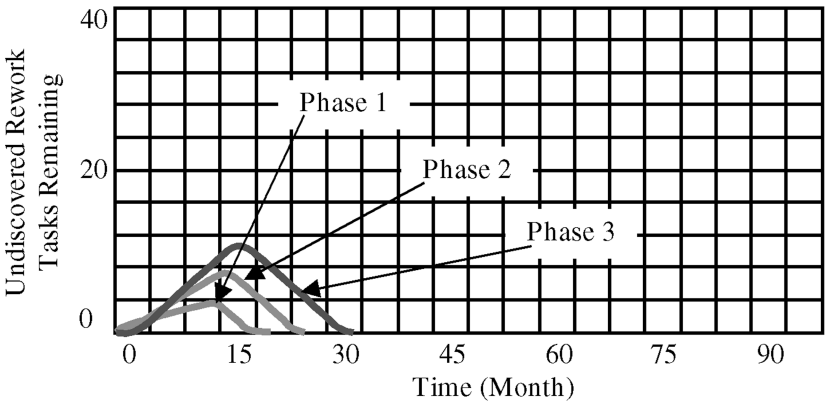

Figure 4.5 Rework Growth as a Function of Number of Phases (Quality = 70%).

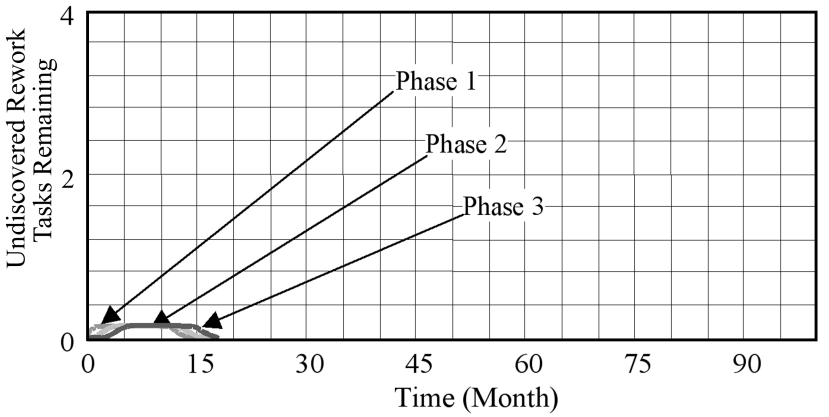

Figure 4.6 Rework Growth as a Function of Number of Phases (Quality = 90%).

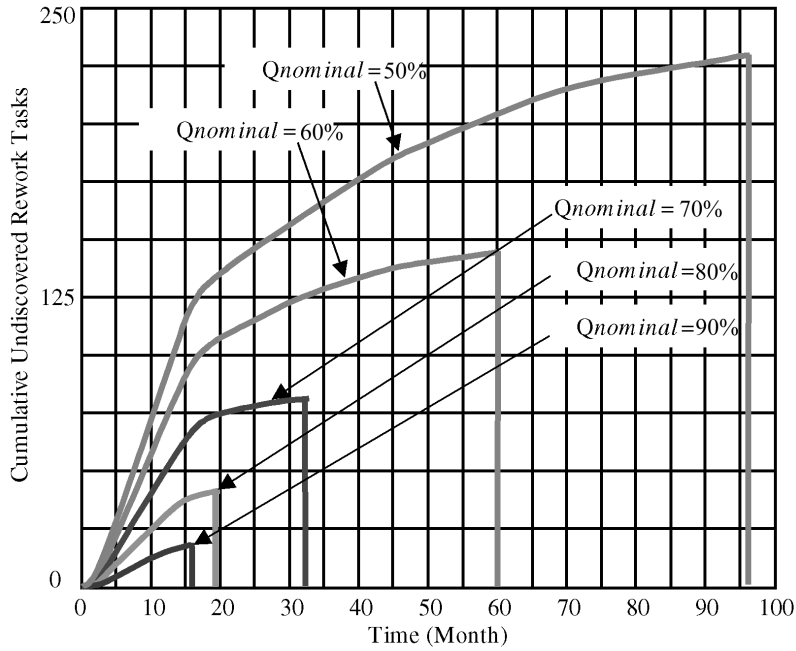

Figure 4.7 Cumulative Rework Generated as a Function of Quality Level.

in about 17 months. With a quality level of 70% the program finishes in about 32 months. At a quality level of 50% the program finishes in about 96 months.

The absolute numbers here are not important. This is a relatively simple model and a real program is far more complex. Nevertheless, the trends the numbers indicate are worthy of note. It is obvious from this analysis that the number of tasks that need to be reworked is significantly reduced by increasing the level of the quality of work performed. It naturally follows that, with less work needing to be redone, the program can be completed much faster and therefore in a much more cost-effective manner.

Figure 4.7 provides a comparison of how the cumulative amount of rework generated grows non-linearly as the quality level deteriorates. The vertical axis represents the cumulative number of Rework Discovery tasks generated. The same point made in Figures 4.4, 4.5, and 4.6 above is reiterated here: Rework resulting from poor quality adds significant cost to the program in terms of both dollars and schedule slippage.

Key Point

> Rework can also induce technical risk to the program because the results of the corrected work could involve changing technical parameters that invalidate certain aspects of the design.

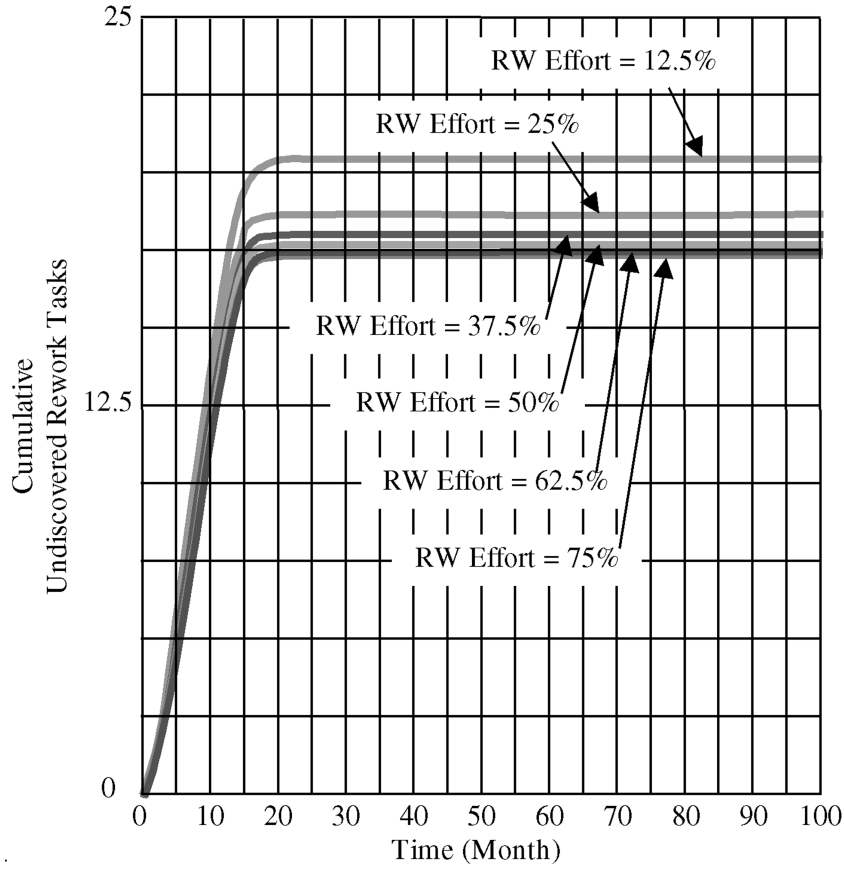

Figure 4.8 Rework Generated as a Function of Rework Discovery Effort — Quality = 90%.

The SDF outlined in this book identifies specific tasks that ought to be performed specifically for the purpose of discovering rework as early in the process as possible. This is necessary in order to mitigate its potential negative consequences.

How Intense Should Rework Discovery Be? — As mentioned above, in this simulation, the amount of effort applied to discovering rework is defined as a percentage of the effort applied in generating work complete. Figures 4.8, 4.9, and 4.10 illustrate the impact of varying the level of effort applied to discovery of rework. For each simulation the productivity level is set to 90%. The horizontal axis in each figure represents time, while the vertical represents the total number of rework discovery tasks generated (more is worse).

In Figures 4.8 through 4.10 it is the knee in the curve that indicates at what point Rework Discovery activities level off. This is the point at which the phase concluded. In order to find the optimal percentage of effort that

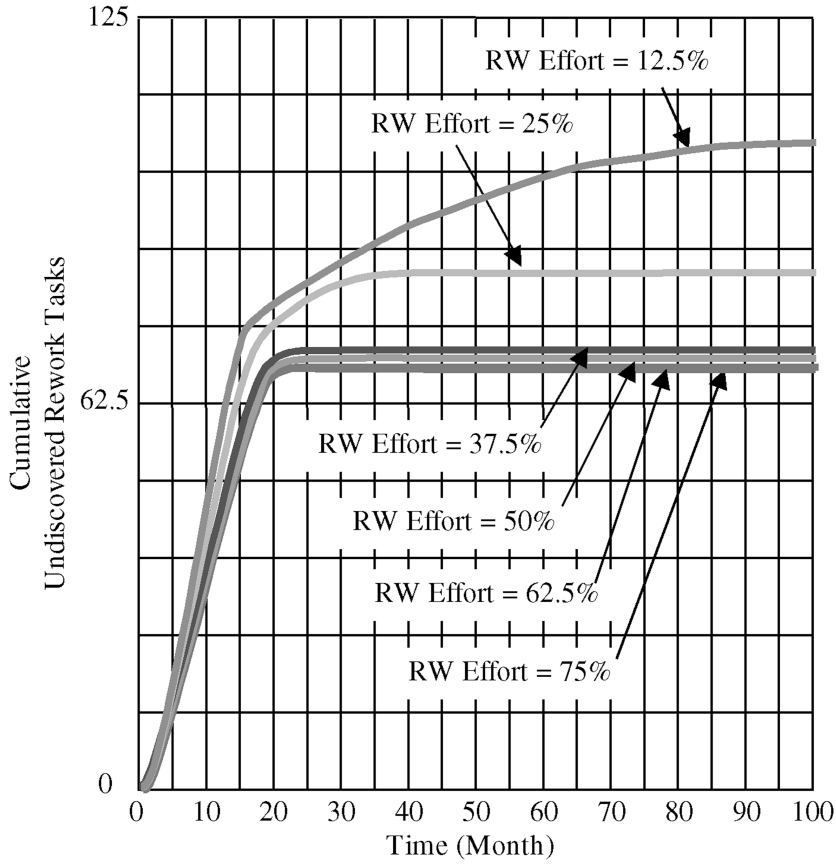

Figure 4.9 Rework Generated as a Function of Rework Discovery Effort — Quality = 70%.

should be directed toward discovering rework, the curve with the lowest cumulative amount of rework must be identified. For the first simulation, the nominal quality parameter is set to 90%. At this quality level, there is minimal difference between applying 25% effort toward discovering rework and applying higher percentages. However, as quality deteriorates, more effort must be applied to discover rework in order to minimize its impact to cost and schedule.

Figure 4.9 shows the impact of operating at a quality level of 70%. At such a quality level, about 38% of the effort applied should be directed toward rework discovery. More than that would not be cost effective. Less than 38% would result in unnecessarily high cost and schedule impacts.

Figure 4.10 shows the results of running the simulation at a quality level of 50%. In this scenario, the effort applied to discovery of rework should be on the order of 50% or more.

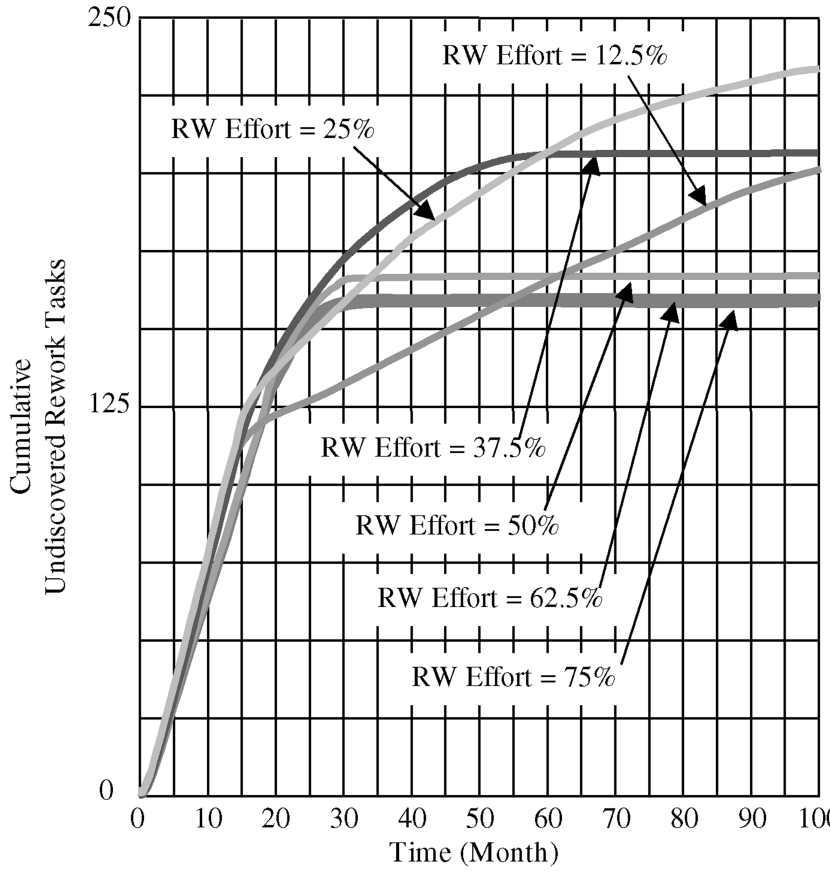

Figure 4.10 Rework Generated as a Function of Rework Discovery Effort — Quality = 50%.

Looking at Figures 4.8 through 4.10, notice how the total rework generated grows at a non-linear rate as quality deteriorates from 90 to 70% and finally to 50%. At a quality level of 90% (Figure 4.8) the total rework ranges from about 17 to 21 tasks, depending upon how much effort is applied to finding it. At a quality level of 70% (Figure 4.9) the rework grows to a minimum of about 60 tasks. At a quality level of 50% (Figure 4.10) the rework grows to a minimum of about 160 tasks. In terms of schedule, at 90% the program finishes within about 20 months; at 70% it finishes in about 25 months; and at 50% it completes in about 35 months when the rework discovery effort is run at the optimum level. Notwithstanding the above, the main point of Figures 4.8 through 4.10 is not to determine the optimal amount of rework discovery effort that must be applied in those specific situations.

Key Points

- Failure to acknowledge the existence of rework and to address it properly can be detrimental to a development program.
- Poor quality results in unnecessarily high cost and schedule resources spent on fixing problems that may well have been avoided.
- The lower the quality level, the higher the percentage of resources must be applied to discover the resulting rework in order to minimize its adverse effects on cost and schedule. This relationship between quality level and resulting rework is exponential.

III. Rework Mitigation

The preceding discussion has shown how even small degradations in the quality of work performed can undermine the ability of a program to operate within budgeted cost and schedule constraints. Undiscovered rework injects cost, schedule, and technical risk into any development program because the nature of the problems is not known. Undiscovered rework, by definition, is unwittingly accepted within the program as work complete. Such data is used to make design decisions, make-buy decisions, critical trade-offs, cost estimates, schedule estimates, etc. This undiscovered rework will make itself known at some point in the program: during subsequent design phases, during manufacturing, during integration and test, during deployment, or during operations. It is obvious that the later in the program it is discovered, the more difficult and expensive it will be to rectify. In addition, there may be hazards to humans and the environment hidden in the design.

Key Point

The main point here is that improving the quality of work performed must be a priority if a program is to be run at minimal risk. As a corollary, effort must be made to discover undiscovered rework in order to minimize its cost, schedule, and technical effects on the program.

What are some of the causes of rework during the development process? The following nonexhaustive list identifies some of the sources.

- Imposed requirements incomplete, ambiguous, contradictory, unverifiable
- Undiscovered mission requirements

- Planned technology not available in time
- Initial design "granularity" insufficient to detect fundamental issues
- Heritage hardware and/or software not robust enough for new context
- Interfaces not compatible
- Initial technical, cost, and schedule allocations not sufficient
- Inadequate control and flow of requirements and information
- Incomplete or too top-level performance analyses
- Testability and/or producibility difficult and expensive
- Failure to adequately test and/or simulate design
- Inadequate test planning

These are not uncommon causes of rework. This short, nonexhaustive list suggests that activities ought to be included in the SDF that are specifically aimed at discovering these causes of rework. In Adamsen (1995), a linear, sequential SDF is developed. As Table 4.1 indicates, the activities identified, which are typical of most system engineering processes, fall naturally into two categories: those activities focused on generating work and those focused on discovering rework. It is interesting that one half of those activities focus on the discovery of rework.

Key Point

> Sometimes the activities described in the table as focused on rework discovery are viewed as "non-value-added" activities. The preceding discussion shows that such a view is misguided. There is much "value-added" to finding rework early, which will serve to ensure that the program proceeds with minimal technical, cost, schedule, and safety risk to the program.

Table 4.1 Focus of SDF Activities Defined in Adamsen (1995)

Requirements Development		Design and Analysis				Verification	
Activity	Main Focus	Activity	Main Focus	Activity	Main Focus	Activity	Main Focus
Requirements Analysis	Discover Rework	Identify/Modify Design	Work	Analyze Performance	Discover Rework	Analysis (may be same as Synthesis) Test	Discover Rework
Mission Analysis	Work	Allocation	Work	Assess Producibility, Testability	Discover Rework		Discover Rework
Requirements Verification Check	Discover Rework	Functional Decomposition	Work	Optimize	Work	Plan System Test	Discover Rework
Functional Analysis	Work and Discover Rework	Design Integration	Work				