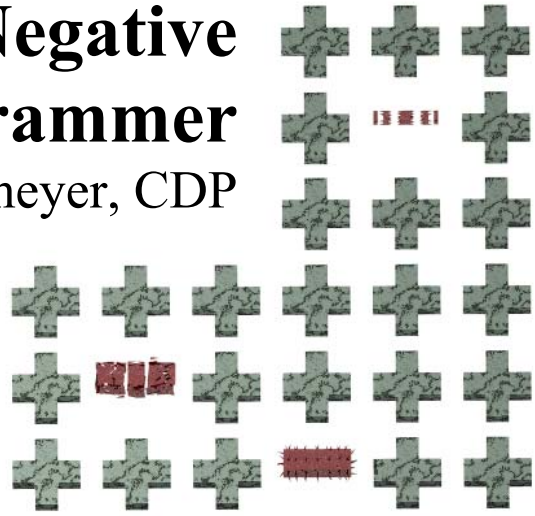
# The Net Negative Producing Programmer

G. Gordon Schulmeyer, CDP



## Introduction

We've known since the early sixties, but have never come to grips with the implications that there are net negative producing programmers (NNPPs) on almost all projects, who insert enough spoilage to exceed the value of their production. So, it is important to make the bold statement: *Taking a poor performer off the team can often be more productive than adding a good one*. [6, p. 208] Although important, it is difficult to deal with the NNPP. Most development managers do not handle negative aspects of their programming staff well. This paper discusses how to recognize NNPPs, and remedial actions necessary for project success.

Researchers have found between a low of 5 to 1 to a high of 100 to 1 ratios in programmer performance. This means that programmers at the same level, with similar backgrounds and comparable salaries, might take 1 to 100 weeks to complete the same tasks. [21, p. 8]

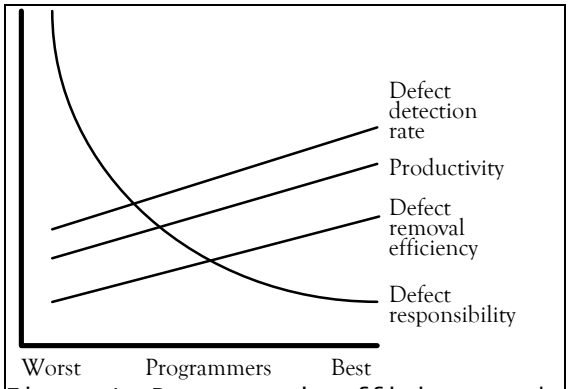



Figure 1: Programmer's efficiency and e
(adapted from DeMarco [6])

The ratio of programmer performance that repeatedly appeared in the studies investigated by Bill Curtis in the July/August 1990 issue of *American Programmer* was 22 to 1. This was both for source lines of code produced and for debugging times - which includes both defect detection rate and defect removal efficiency. [5, pp. 4 - 6] The NNPP also produces a higher instance of defects in the work product. Figure 1 shows the consequences of the NNPPs.

This negative production does not merely apply to extreme cases. In a team of ten, expect as many as three people to have a defect rate high enough to make them NNPPs. With a normal distribution of skills, the probability that there is not even one NNPP out of ten is virtually nil. If you are unfortunate enough to work on a high-defect project (density of from thirty to sixty defects per thousand lines of executable code), then fully half of your team may be NNPPs. [6, p. 208]

John Gardner, in *No Easy Victories*, said, "An excellent plumber is infinitely more admirable than an incompetent philosopher." He went on to observe that if society scorns excellence in plumbing because it is a humble activity and accepts shoddiness in philosophy because it is exalted, "neither its pipes nor its theories will hold water." [14, p. 13] In the context under discussion, substitute "programmer" for "philosopher" and draw the conclusion that either the computer program will not run or that the NNPP will be so slow as to thwart project success.

## Recognizing NNPPs

In Table 1 it is pointed out that there are significant differences between manufacturing and software development. It is a consequence of these differences that sets the programmer apart and provides difficulties in the recognition of the NNPP.

A model of programmer behavior would help us to recognize NNPPs. Such models of programmer behavior must be able to account for at least these basic programmer tasks:

* *composition:* writing a program
* *comprehension:* understanding a given problem
* *debugging:* finding errors in a given program
* *modification:* altering a given program to fit a new task
* *learning:* acquiring new programming skills and knowledge

| | Manufacturing | Software Development |
|---|---|---|
| Input quality | Uniform | Highly variable |
| Input/product characteristics | Uniform | Unique, different |
| Process characteristics | Repetitive | Creative |
| Control action | Well-defined | Ambiguous |
| Output quality | Measurable | Not mesurable |
| Interplay of measurement and control | None | Significant |

Table 1: Software development differences (taken from Christenson[4])

In addition, the model must be able to describe these tasks in terms of: the *cognitive structure* that the programmer possesses or comes to possess in memory, and the *cognitive processes* involved in using this knowledge or in adding to it. [21, p. 46] Certainly, the programmer tasks of composition, debugging, modification, and learning may be measured (more about this below) fairly easily to determine programmer differences. The *cognitive structure* (comprehension) is not measurable as such, but manifests itself in the *cognitive processes* (composition, debugging, modification, and learning).

## Metrics

Perhaps surprisingly, the people orientation of companies has a tough side. *In Search of Excellence* tells us that the excellent companies are measurement-happy and performance-oriented, but this toughness is borne of mutually high expectations and peer reviews. [17, p. 240]

Most functions associated with software development can undertake a measurement program. However, we find that little measurement has been done by the programming community. For whatever reason, programmers and their managers seem not to want to record information concerning software defects. Two possible reasons that come to mind are lack of time and a disinclination to ponder the less attractive aspects of what to do about NNPPs. [8, p. 355]

No matter the disinclination, to recognize NNPPs we must measure. Of specific measurement interest are how programmers balance productivity versus defect generation, and what is the programmer's defect detection rate and defect removal efficiency.

For productivity versus defect generation a look at the following metrics at the individual level would help; see table 2.

| Productivity | Defect Generation |
|---|---|
| Document pages per hour | Defects per 100 pages |
| Lines of PDL* per hour | Defects per 1000 lines of PDL |
| LOC† per hour | Defects per 1000 LOC |
| * Program design language | |
| † Lines of code | |

Table 2: Productivity vs. defect generation

Whereas, the difficulty of capturing defect detection rates and defect removal efficiency during debugging and testing is much greater than the productivity versus defect generation. Possible remedies would be for a software configuration control board to track software problem reports by individual names uncovering defects and by dating the problem hand-off to an individual and dating the resolution of that problem. Be very careful that the vast differences of the seriousness of problems reported on software problem reports are not ignored.

Determining the NNPP through these metrics, though difficult, is doable. But, questions of legality and ethics must be considered. The delicacy of measuring one-person projects is especially sensitive in Europe, where some countries prohibit the measurement of an individual worker's performance either because of national law as in Sweden, or because the software staffs are unionized and such measurements may violate union agreements, as in Germany. [13, p. 30, 31]

Ethically, it may be inappropriate to weed out NNPPs based on metrics because:

- We have not measured long enough to be certain of the accuracy of our measurements.
- We do not know which of the measures or what combination of metrics correlate best with the behavior we want to encourage.

- We feel that such a use would lead to distortion of the data, and we depend on the measurements to give us insight into the entire development process. [10, p. 84]

The basic fear is that we will pick the wrong thing to measure or that whatever we pick cannot be measured consistently or accurately. [10, p. 83]

The legal and ethical fears are such that it is believed that there must be a published protocol regarding the use of the metric data. The purpose of the protocol is to allay the legitimate worries about misuse of the data. The underlying philosophy of the protocol must be: data collected on an individual basis can be used only for the benefit of the individual. Upon finding that a given project worker is an NNPP, the measurer can share the data with that NNPP and with no one else: "Fred, you're rather defect prone. Over four months of coding, you introduced 79 defects that cost altogether more than $53,000 to remove. The average worker introduced less than $2,600 worth of defects while doing approximately the same work." Once the data is trusted, the NNPP will use it to maximize their contribution to the effort. It is believed that one's own self-esteem is a far greater motivation than any that software management could apply. [6, p. 211]

## Reasons For NNPPs

Our understanding of NNPPs would be further enhanced if we explore some of the driving reasons for their performance. A fairly recent study of 250 heterogenous subjects examined four of the most frequently named causes of poor performance: lack of job satisfaction, lack of identification with and involvement in the organization for which the programmer works, little belief that professional behavior will result in reward, and the programmer's own lack of a sense of professionalism. [7, p. 46]

A negative attitude toward productivity can be expected to cause an individual to be less productive than if he or she embraces a positive attitude. Pressuring for productive behavior may merely cause the person to feel stress from being "coerced" and thus increase the negative attitude. [2, p. 15]

We may not be aware that the group is working toward a different set of objectives than, say, their manager thinks they are. Consequently, unless we take precautions to see that *all* the objectives are communicated, and remain so, we should not be surprised when the program does not meet schedules or runs inefficiently, or uses too much storage.

There is, however, a certain danger in communicating objectives: objectives can change estimates! Experimentally, after it was found that instructions for objectives effect performance, the estimates given by each programmer were checked. The programmers had estimated number of batch runs and number of elapsed days to complete each project, and the comparison of their actual and estimated runs and days is shown in Table 3. Those who were instructed to finish as fast as possible were motivated to be far more conservative in their estimates of time to completion, and they actually performed much better than their estimates and much better than the other group - even though the other group had been much more optimistic in their estimating.

| | Runs | | Days | | |
|---|---|---|---|---|---|
| | Estimated | Actual | Estimated | Actual | % Late |
| Efficient Proramming | 22 | 69 | 48 | 76 | 75 |
| Fast Programming | 39 | 29 | 68 | 65 | 25 |

Table 3: Effect of environment on estimating (taken from Weinberg [23])

Software managers should take a long, hard look at this table before they set down the goals for their next project. If a goal is set explicitly, there are two effects: programmers work toward the goal at the possible expense of another goal, and programmers will be far more conservative (or accurate) in estimating how well they will meet the goal. Estimates on goals not emphasized will probably be completely unreliable, both because they are not made carefully and because they are not important enough to resist being sacrificed to other goals. [23, pp. 130, 131]

## Remedial Actions

Now that we have seen how to recognize NNPPs through metrics and uncovered some reasons for that poor performance, we need to focus on some of the remedial actions possible for NNPPs. No longer are steps to ignore, deny, or blame others acceptable, we must move to the top steps of problem solving shown in Figure 2 by assuming responsibility and finding some solutions. This aphorism in *The One Minute Manager* should provide guidance to the software manager's rehabilitation of the NNPP: "Everyone is a potential winner; some people are disguised as losers; do not let appearances fool you. [3, p. 71]

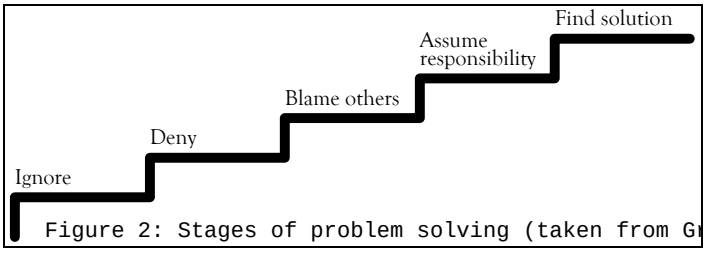


Figure 2: Stages of problem solving (taken from G

The model of programmer behavior discussed above provides some insight into potential remedial actions. The *cognitive processes* of composition, debugging, modification, and learning when measured, give positive identification of the NNPP which puts "strength" in the software manager's hands to deal with the issues.

When one or more of your people goes off course or does a bad job, you have got to let them know it and take immediate remedial action. One way of avoiding big problems is to subdivide the job sufficiently and have enough checkpoints so that missing one is a signal, not a catastrophe. [16, p. 109] Some other remedial actions covered below are counsel, reassign, or discharge.

# Counseling

The Hawthorne experiment researchers suggested that a supervisor could best counsel an employee if they followed these five rules:

1 Listen patiently to what the employee has to say before making your comments

2 Refrain from criticizing or offering hasty advice

3 Never argue, while counseling

4 Give your undivided attention to the employee while the employee is talking

5 Look beyond the mere words of what the employee says - listen to perceive something deeper than what appears on the surface. [1, p. 282]

A significant aspect of counseling the NNPP is the correct use of performance appraisal, or better, performance management. Some new insights into software performance management were provided by Susan Webber in the July/August 1990 *American Programmer*. A key element is appropriate consequences of the programmer's actions. Although appropriate consequences are a critical ingredient in effective performance management, five additional factors are involved: pinpoint behaviors and results, measurement, feedback, goals and subgoals, and positive reinforcement. [22, pp. 67, 68] Often, the poor performance of the NNPP is a result of one or more of these major timewasters: management by crisis; attempting too much; personal disorganization; lack of self-discipline; inability to say no; procrastination; leaving tasks unfinished; socializing; or incomplete information [15, p. v, vi] If pointed out quickly, this behavior could be made to stop, thus turning around the poor performance of NNPPs.

System test groups have the objectives of writing and executing effective test suites and are reinforced for finding defects. Alternatively, design groups have the objectives of writing code that can be executed without defects and for effectively correcting defects that are found by the systems groups. Be careful, as noted above, that the objectives are as desired and unambiguous.

One key result metric for a design group is the backlog of problems it has to resolve. Performance management has been successfully applied to defect backlog. As a result of one application of performance management, a 21 percent improvement in productivity in this area was attained. In addition, rather than the typical complaints usually made when individual performance is measured, the programmers made positive comments about the performance management and its implementation. The experience of these programmers with performance management has resulted ia an association of metrics with positive consequences. [23, pp. 70, 71]

## Reassigning

The problems of the NNPP are cleared up quite satisfactorily by a scrupulous adherence to the principle of trying alternatives before resorting to discharge. This is especially true when the difficulty was due to carelessness or complacency on the part of the NNPP (an unmistakable warning will often suffice to snap them out of their lethargy), or to incompatibility between the supervisor and the employee (in which case a simple reassignment sometimes works wonders). [9, p. 117]

No one need be embarrassed to have been reassigned tasks that make optimal use of their strengths. For instance, a poor coder is given debugging duties; or a poor tester is given coding or documentation duties. Even NNPPs who are temporarily chastened to learn that all their testing and diagnostic skill barely made up for a high defect insertion rate, will know (once assigned properly) that their prior low value to the project was mostly due to *management failures*, rather than their failure. [20, p. 205]

Reassignments are easier for Japanese managers, who are willing to go along with a change if it contributes to any of the following goals: making the job easier or safer, making the job more productive, improving product quality, and saving time and cost. This is in sharp contrast to the Western manager's almost exclusive concern with the cost of the reassignment

and its economic payback. [12, p. 114] Typically, because economic payback is high with these NNPP reassignments, they readily fulfill the Western manager's criterion.

Job satisfaction may be enhanced at all programmer skill levels by using both performance management as covered above and job enrichment techniques. [19, p. 16] This job enrichment for a NNPP moves away from reassignment as such, but allows creativity on the part of the software manager to focus on specific aspects of the NNPP's assignment.

## Dismissal

Because termination is very costly for the organization and potentially a matter for legal action by the NNPP, it must be the last resort. Table 4 provides the software manager with a way to measure the influences of various alternatives to termination.

If it is clear after counseling and reassignment, then the organization may be best served by firing the NNPP.

| Instructions: insert<br>+ for positive influences<br>- for negative influences | Probability of Success | Time and Resources Required | Endorsement by Superior | Temporary/ Permanent Results | Reflection on Managerial Capabilities |
|---|---|---|---|---|---|
| Rehabilitate the subordinate | | | | | |
| Change how you feel about NNPPs | | | | | |
| Restructure the job | | | | | |
| Restructure the situation | | | | | |
| Demote NNPPs | | | | | |
| Transfer NNPPs | | | | | |

Table 4: Influences of alternatives to termination
(adapted from Roseman [18])

## Conclusion

It is likely that your project has NNPPs. This paper was written to help you recognize the NNPP and the various remedial actions which are suggested to improve the situation.

## Bibliography


1 Bittel, Lester R. *What Every Supervisor Should Know* (2nd Ed.) (New York: McGraw Hill, 1968)

2 Blake, Robert R. & Mouton, Jane S. *Productivity: The Human Side* (New York: AMACOM, 1981)

3 Blanchard, Kenneth & Johnson, Spencer *The One Minute Manager* (New York: Berkley Books, 1981)

4 Christenson, D. A. et al, "Statistical Methods Applied to Software", collected in Schulmeyer, G. Gordon & McManus, James I. *Total Quality Management for Software* (New York: Van Nostrand Reinhold, 1992)

5 Curtis, Bill, "Managing the Real Leverage in Software Productivity and Quality", *American Programmer* July/August 1990

6 DeMarco, Tom *Controlling Software Projects: Management, Measurement & Estimation* (New York: Yourdon Press, 1982)

7 Dunn, Robert H. *Software Quality: Concepts and Plans* (Englewood Cliffs: Prentice Hall, 1990)

8 Dunn, Robert H., "The Quest For Software Reliability", collected in Schulmeyer, G. Gordon & McManus, James I. *Handbook of Software Quality Assurance* (New York: Van Nostrand Reinhold, 1987)

9 Gellerman, Saul W. *The Management of Human Relations* (New York: Holt, Reinhart and Winston, 1966)

10 Grady, Robert B. & Caswell, Deborah L. *Software Metrics: Establishing a Company-Wide Program* (Englewood Cliffs: Prentice Hall, 1987)

11 Grove, Andrew S. *High Output Management* (New York: Random House, 1983)

12 Imai, Masaaki *Kaizen: THe Key to Japan's Competitive Success* (New York: McGraw Hill, 1986)

13 Jones, Capers *Applied Software Measurement: Assuring Productivity and Quality* (New York: McGraw Hill, 1991)

14 Longstreet, William, "Executive Exchange - Seize the opportunity for Qualitivity," *Industry Week* January 25, 1982

15 Mackenzie, Alec *The Time Trap* (New York: AMACOM, 1990)

16 Metzger, Philip W. *Managing a Programming Project* (Englewood Cliffs: Prentice Hall, 1973)

17 Peters, Thomas J. & Waterman, Jr., Robert H. *In Search of Excellence: Lessons from America's Best-Run Companies* (New York: Harper & Row, 1982)

18 Roseman, Edward *Confronting Nonpromotability: How to Manage a Stalled Career* (New York: AMACOM, 1977)

19 Schulmeyer, G. Gordon *Computer Concepts for Managers* (New York: Van Nostrand Reinhold, 1985)

20 Schulmeyer, G. Gordon *Zero Defect Software* (New York: McGraw Hill, 1990

21 Shneiderman, Ben *Software Psychology: Human Factors in Computer and Information Systems* (Cambridge, MA: Winthrop, 1980)

22 Webber, Susan, "Performance Management: A New Approach to Software Engineering Management", *American Programmer* July/August 1990

23 Weinberg, Gerald M. *The Psychology of Computer Programming* (New York: Van Nostrand Reinhold, 1971)