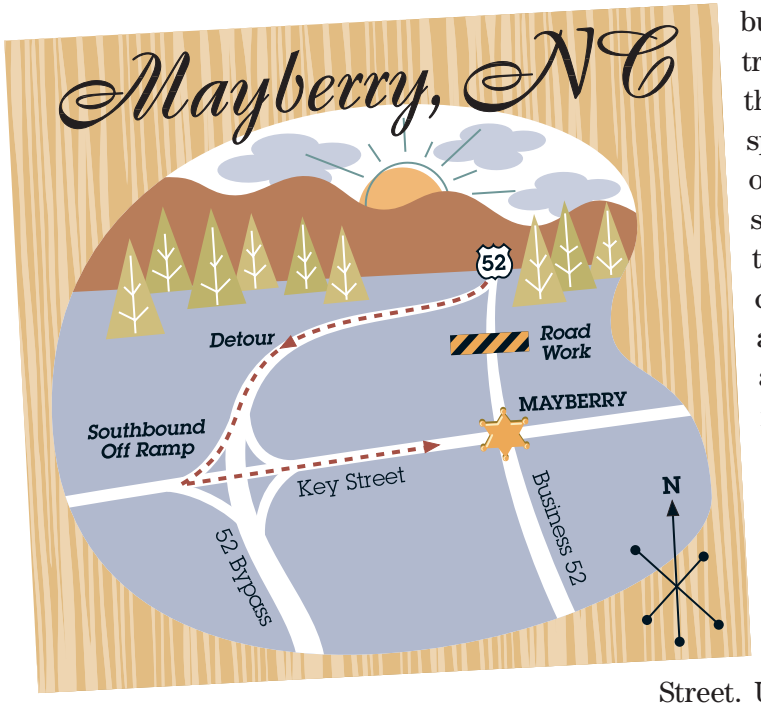
Management & Teams

Managing *in* Mayberry

An examination of three distinct leadership styles *by Don Gray and Dan Starr*

Near the Blue Ridge Mountains in North Carolina, not far from where you think it should be, there really is a town called Mayberry.

Although the main highway bypassed the town years ago, the namesake for the popular 1960s television series is still a bustling community, and a fair amount of traffic enters Mayberry's downtown from the north on the US Highway 52 business spur every morning. In town for a week of consulting work, we were able to observe the recent road construction along that route—and watched a trio of local citizens demonstrate their own unique administrative styles. Let's take a look at how these characters' traffic management closely parallels common styles of *software project* management.



MAP: ANNIE BISSETT

When road work just north of town closed Business 52, all the traffic entering town from the north had to take the 52 bypass around to the west side of town and enter the downtown on Key Street. Unfortunately, this meant traffic would have to make a left turn onto Key Street, crossing fairly busy east-west traffic (see illustration).

The town council feared that during the morning rush the traffic waiting to make the left turn onto Key Street would back up on the southbound off-ramp all the way to Highway 52 itself. This could cause a serious accident, since Highway 52 has a 65 mph speed limit. So the council decided to station one police officer and one or two rescue squad volunteers at the intersection to make sure that traffic on the ramp did not back up.

Three Approaches to Managing

Being a take-charge guy, the officer on duty (we'll call him Barney) arrived at the scene Monday and quickly sized up the situation. He decided that what was needed was a traffic light at the intersection of Key Street and the ramps. Since it would take the county months to approve a light, he decided to operate as a "human traffic light," directing traffic manually. Each direction got its turn: westbound Key (including left turns onto the southbound ramp), then eastbound Key (including right turns onto the southbound ramp), then the off-ramp (which could turn either way onto Key). Barney's plan didn't actually work all that well. Traffic stalled in both directions on Key Street. And there were a couple of close calls on the ramp; traffic backed up almost onto Highway 52 once when Barney let a few cars turn left onto Key Street. By the end of rush hour he was hot, tired, and a little discouraged, and he had written a fistful of citations to drivers for making unmentionably rude gestures at a law enforcement officer.

On Tuesday, one of the rescue squad volunteers (a helpful local woman known as Aunt Bee) said she knew how to take care of the situation. She figured that traffic could probably take care of itself as long as drivers didn't have to cross each other's paths. So she let traffic go both ways on Key Street, and let people make right turns onto and off the ramps. When somebody had to turn left, she'd stop the other lanes and let them go. Aunt Bee's approach worked better than Barney's (at least nobody made rude gestures at her), but there was still a lot more congestion than she expected, and by the end of rush hour Bee was glowing profusely.

On Wednesday Sheriff Andy showed up, bringing a lawn chair and a thermos of lemonade. He set up the lawn chair on a shady spot from which he could see the intersection and a fair way down the off-ramp, and sat down to sip lemonade. When traffic started to back up on the ramp, he got up, stopped Key Street traffic, and let the ramp empty; then he went back to his lemonade. Other than that, Andy pretty much didn't seem to *do* anything. Despite his apparent inaction, the intersection just seemed to work. People were calm and relaxed, with the drivers making right turns creating breaks for others making left turns, and everything worked a lot like it did before anyone showed up to help—just a little better.

Putting on our consultant hats, we realized we'd just witnessed three distinct styles of management—Barney's *micro*management, Aunt Bee's *motherly* management, and Andy's *masterly* management. Let's look at each of them in more detail, and see what we can apply to our own software projects.

A Question of Style

Each of our managers made different assumptions that shaped their style—in particular, assumptions about the *people* being managed, and about the *role* of the manager. These assumptions determined how they approached the critical activities of managing. In his book *Quality Software Management, Vol. 1: Systems Thinking*, Jerry Weinberg highlights five critical activities:

1. understanding **the problem to be solved,**
2. **planning** the solution approach,
3. **observing** what the people being managed are actually doing,
4. using **rules and process models** to determine what to do next, and
5. **taking action** to guide the group toward its goal.

Together these activities form a feedback system that "steers" the project team. How they are executed (i.e., what the manager defines as the problem, how the manager plans, what observations get made, which rules get followed, and how the corrective actions get taken) makes all the difference—determining just where the team will go, how the team members will feel about the software project as a whole, and ultimately how satisfactory the results will be.

Micromanagement

Barney practiced micromanagement, which is based on the assumption that the manager has to see to it that everything gets done. Most micromanagers don't deliberately meddle out of a need to be in control; they're just operating under the assumption that if they don't do it, it won't get done. Micromanagers also tend to make the assumption that those being managed will do what they're told to do; no more, no less.

These assumptions describe machines better than they do humans. Indeed, when Barney said the town needed "human traffic lights," he was describing a situation in which both the manager and those being managed were more mechanical than human. Perhaps this is why so many good programmers become micromanagers when they get their first promotion—they're just "programming" the "biorobots" who work for them!

Using Weinberg's model, we can see how Barney's assumptions defined his view of the critical management activities:

1. The **problem to be solved** was to personally make sure everything was done in an orderly fashion.
2. The **plan** that followed was for Barney to pretty much do everything himself. He would personally direct the movements of each and every vehicle. This meant that the plan had to be simple enough that he could be in control of its execution at all times.
3. Even with the simple plan, Barney was far too busy directing

traffic to **observe** much. Standing in the middle of the intersection, he wasn't in the right position to see up the ramp when traffic began to back up onto Highway 52.

4. Even if he had made better observations, his manager-centered **process model** didn't allow him to do much. The underlying assumption that he was personally responsible for each and every car going through the intersection meant that he couldn't delegate much—he couldn't count on the drivers to do anything other than what he told them to do.

5. Barney's **actions** were pretty limited; because he had to control each vehicle, he couldn't leave his spot in the middle of the intersection. In the end, he couldn't do much beyond try harder at what he was already doing—waving his arms more frantically at the folks, in the hopes that they'd get through faster.

Because the manager must make (or at least approve of) all decisions, only one thing happens at a time and everything else lines up waiting for a turn. When simplicity, centralized information, and oversight are turned from virtues into vices, it creates a choke point that affects project planning and execution:

Simplicity Since the entire project plan must be under the control of the manager at all times, the plan must be simple enough that a single person can comprehend it in its entirety. This sets an upper bound on project complexity—if the problem to be solved is beyond this bound, the manager has to simplify it somehow (e.g., letting traffic go in only one direction at a time). This serialization of activities is a common simplification in micromanaged projects as well, and it wastes both effort and time. When serialization isn't enough, the manager may start leaving "non-essential" activities out of the project plan. Micromanagers are notorious for over-simplifying, to the point where their software project plans may leave out something essential for a successful product launch.

Centralized information Since the manager is the only one who can make a decision, it's critical that he get lots of quality information about how the project is doing. Unfortunately, the only observations allowed are those that the *manager* puts in the project plan—but that manager's far too busy making each and every decision to actually observe much of anything. So in practice, micromanagers are often flying blind, making decisions on little or no actual information.

Oversight The need to get explicit approval for each action adds to the amount of time required to accomplish tasks. So micromanagement tends to be inefficient, with a lot of people waiting around for the manager to tell them what to do next. The manager-as-bottleneck is a key structural problem. The practice also leads to people problems, such as initiative squelching. The manager's assumption implies that the people being managed have nothing to contribute beyond the functions defined for them by the manager. What if the workers want to do something other than follow the rules—because they see a better way or a problem with the plan? Forget it. The micromanager will not allow it to happen. This creates short tempers and long days for those who are micromanaged.

Most people don't like this style of management. Some will respond with a sort of dead, mechanical compliance, waiting dutifully for their next set of instructions from the manager. Others may choose some form of subtle rebellion, such as "working to rule"—following the manager's instructions to the letter, no more, no less, even when those instructions are clearly a recipe for failure. And others will rebel more openly, taking advantage of the manager's continual distraction to get away with whatever they can. Alas, these responses to micromanagement tend to reinforce the micromanager's assumptions and lead to even more micromanagement. Micromanagers tend to be very busy people.

So, is micromanagement ever appropriate? Certainly, when the problem to be solved is small enough for one manager to truly comprehend the entire project plan, and the people doing the work are willing to follow each and every command of the manager. While this situation can occur now and then, it's not very common in the software world.

A common cause of micromanagement is the newly promoted, technically competent manager rushing in to help a floundering employee or rescue a particular part of a software project. This creates a co-dependent dynamic where the manager becomes the rescuer, and the employee becomes helpless. This ensures that the next time there is a problem, the manager will step in again, and so on, until something happens to break the pattern.

While micromanaged projects can (and often do) result in successful product launches, it's often more in spite of their management than because of it. There ought to be a more efficient and less aggravating way to handle the situation.

Motherly Management

Aunt Bee chose a kinder, gentler style that we call motherly managing, allowing the drivers to do some things for themselves, and helping them when she thought they needed help. But her underlying assumption was still pretty close to Barney's: the people being managed might be able to do a few routine things without being told, but all significant decisions—especially when there was some form of contention or competition—were still firmly under her control.

If the micromanager views the people being managed as machines, the motherly manager sees them more like children, able to do a few routine things but still needing protection from anything potentially dangerous. Like the micromanager, the motherly manager is not necessarily malicious or desperately in need of control. Aunt Bee had no great need to have power over the drivers; she just knew that they couldn't make major decisions without her help. She simply couldn't visualize the situation where one person could be turning left into the gap created by another turning right, because she couldn't see who was controlling the relationship, and she knew that two drivers

certainly couldn't cooperate without somebody to coordinate them.

Aunt Bee's motherly assumptions defined her view of the key management activities:

1. The **problem to be solved** was something like "take care of the people who have to cross other traffic." Like Barney, she saw the problem in personal terms; it was her problem, not the drivers' problem.

2. Because Aunt Bee saw the drivers as human beings who could do a few things for themselves, her **plan** was a bit less rigid than Barney's. She could allow at least a few routine things to happen in parallel, but under exceptional conditions she would take full control of everything, which meant reverting to serial execution.

3. Aunt Bee's more distributed plan required somewhat more sophisticated **observations** than Barney's. She had to observe those situations in which her help was needed—in particular, left turns. Notice that she *wasn't* observing whether people were having trouble making left turns; her underlying assumption said that a left turn signal was a request for help. Like Barney, she spent her time in the middle of the intersection, a point from which she couldn't see up the ramp very well.

4. Because of her motherly assumption that the people being managed couldn't handle any form of contention or conflict, Aunt Bee's **process models** dictated that she must personally resolve these things. So her response to just about any out-of-the-ordinary condition was to stop traffic and go back to taking turns.

5. Like Barney, Aunt Bee was working from a very limited set of **actions,** in part restricted by her need to be in the position of control at the center of the intersection. If those actions didn't work, about all she could do was more of what she was already doing.

Like micromanagement, motherly management can work when its underlying assumptions are true and the problem and solution aren't too complex. Trouble is, most software development shops aren't day care centers, and most development is non-routine and requires that a lot of conflicts be resolved. Interfaces, partitioning, decomposition, protocols—these are all "left turns" in the view of a motherly manager, who must personally make sure that everybody plays well together. This creates a structural problem similar to micromanagement. Similar, but also different. Since some work can take place independently under motherly management, the manager is less of a choke point than in micromanagement.

But because the process is still highly manager-centric, the actual amount of work that can be done in parallel is often less than expected. We end up with a process that's very nearly effective: almost parallel, relatively observant, and coming awfully close to giving workers independent responsibility:

Parallel (almost) Only pre-defined "routine" things can take place in parallel. As long as traffic went straight ahead or turned right, Aunt Bee's plan seemed to work. But she couldn't predict how many people would want to turn left. When lots of people started turning left, her plan fell apart. In the same way, the actual performance of a motherly-managed software project depends heavily on just how much of the development is really "routine," with no need for interactions or conflict resolution. If there are a lot more "exceptions" than expected, a lot of developers working in parallel according to the project plan may be sitting on their hands waiting for the manager to make a decision. This can make a project plan that was *parallel* in theory become *serial* in practice.

Myopic Motherly managers make more observations than micromanagers, but they still confine those observations to specific conditions noted in the project plan. If the conditions defined by the manager are in fact not the key exceptions that need to be managed, the motherly manager will be spending time and energy observing the wrong thing, while missing the observations that are really necessary for project success.

Nannying Motherly management can be less oppressive than micromanagement for the people being managed, because the "mother" allows her "children" to do a few things on their own. The individual developers can go ahead as long as they aren't going against the flow or getting into conflicts. But at the first indication that something non-standard is going on, the whole process stops until the manager decides what to do. The manager must handle all the decisions that really matter—and this squelches the individual contribution to solving the overall problem just about as effectively as micromanagement. There is a great deal of variation here—a manager who views the employees as teenagers is less openly controlling than one who views them as toddlers. Still, most of the people who work in the software business have college degrees, and we wonder if we're making the best use of their expensive educations when we manage them as though they were children.

While micro and motherly management both pay lip service to the stated goal, the assumptions underlying these two styles concentrate on the *means* rather than on the *end.* The assumptions also severely limit both the things the manager can observe and the actions the manager can take in response. The manager's limited observations make it hard to measure actual progress toward the real goal. This type of management is by nature both inefficient and unreliable—inefficient because the manager can't see opportunities inherent in the situation, let alone take advantage of them; unreliable because the manager is often flying blind toward an unknown destination.

If we are going to find a style that's more efficient and effective than *micro* and *motherly,* we must start by changing our underlying assumptions. Barney sees the people being managed as machines to be programmed; Bee sees them as children to be helped. Now let's see what

happens when Andy views them as adult human beings.

Masterly Management

Andy took an approach that at first didn't look like "management" at all. He just sat in his chair, sipping lemonade and watching traffic on the Highway 52 off-ramp. When it started backing up badly, he strolled out into the intersection, stopped traffic on Key Street, and let the off-ramp clear; then he went back to his lemonade. He seemed to be "working" a lot less than Barney or Aunt Bee, yet traffic flowed smoothly. We refer to Andy's style as *masterly* management—because of our three traffic controllers, only he was truly the master of the situation.

The keys to Andy's management style were his underlying assumptions: that drivers are adults, that most of the time they can take care of themselves, and that his role as a manager is to support these competent adults so they can do the real work of getting themselves safely through the intersection. Andy felt secure enough about his own competence and the drivers' know-how that he could remove himself from the center of the job.

Because Andy did not place himself at the center of the management task, he could be much more flexible and effective at the key management activities:

1. Andy saw the **problem to be solved** as moving traffic efficiently and safely through the intersection. He also realized that most of the time this intersection didn't need any help; people made turns here every day without any supervision. What made this a unique problem that might require some management intervention? The detour increased traffic on the Highway 52 off-ramp, and that might, on occasion, cause traffic on the ramp to back up onto the highway and cause a safety hazard. Notice the difference—while Barney and Aunt Bee defined the problem in terms of what they had to do, Andy defined the problem in terms of results, independent of who actually "did the work." By doing this, Andy positioned himself to observe and "steer" the system that did work, rather than as the person doing the work.

2. With his understanding of the real problem to be solved, Andy was able to construct an effective **plan** for its solution. The drivers could be responsible for getting themselves through the intersection. He and his "management team" would monitor the off-ramp and make sure that it could be emptied when (and if) it backed up far enough to pose a safety hazard. While Barney might accuse Andy of not having much of a plan, the fact is that Andy's simple-looking plan actually allowed some very complex things to happen. Because he didn't attempt to control low-level actions by the drivers, Andy's plan delegated management work to individual drivers. This allowed them to operate in parallel, which they did—drivers waiting to turn left off the ramp took advantage of gaps in traffic created by drivers turning right.

3. Now that he had both a problem statement and a plan, Andy could identify which **observations** he needed to make. To keep traffic from backing up onto Highway 52, he had to watch the ramp—not the intersection. So he positioned himself off to the side, where he could see the ramp. This is another critical difference in Andy's style. Standing in the middle of the intersection, Barney and Aunt Bee were taking in a great deal of information—most of it irrelevant to solving the real problem. They weren't in the right place to make the observations that really matter. Of course, Andy didn't ignore what was happening in the intersection—but he didn't make the intersection his primary focus.

4. Andy's management style used two **process models.** First, if traffic's backing up on the off-ramp, stop traffic on Key Street and allow the ramp to drain. Second, if something blocks the intersection, get it out of the way immediately. The rest of the time, Andy's process model says "Let the drivers take care of themselves."

   Both of these models are more subtle than they look. The first model allows Andy to do some fine-tuning as the morning progresses. How far up the ramp is "too far" for traffic to back up? At first he took a conservative approach, draining the ramp when it was backed up about halfway to the highway. Later, after observing how quickly Key Street traffic could be stopped to drain the ramp, he changed his definition of "too far" to something more like three-quarters of the way up the ramp. This meant even fewer interventions were needed, because often traffic would back up to the halfway point and then drain back down by itself.

   The second model contains a flexible definition of just what triggers action. Andy's looking for a symptom, which could have a variety of root causes. If something blocks the intersection (e.g., a driver too timid to turn left), Andy's model will handle it.

5. Finally, Andy took a lot less "overt" **action** than either Barney or Aunt Bee. Most of the time it appeared that he was doing nothing at all. Yet, when action was required, he knew what action was appropriate and effective. But it would be wrong to say that Andy's actions were simpler than Barney's or Aunt Bee's. In fact, his infrequent interventions required *more* skill. After all, Barney and Aunt Bee were already standing in the middle of the intersection, and had the drivers' complete attention. Andy had to enter an intersection full of moving vehicles, get the drivers' attention, temporarily interrupt their self-management, get the drivers to carry out his instructions, and finally re-establish the self-managing system. This is a task requiring some skill.

Like the other two styles we've discussed, masterly management works

when its underlying assumptions are valid. In software development, where the people being managed are skilled, competent, educated adults, these assumptions are usually *true*. Masterly management, therefore, addresses the structural and behavioral problems we saw with micro and motherly management:

- The delegation inherent in the plan means that most contentions and minor conflicts get solved without the manager's intervention, so most of the time the people aren't waiting for the manager's attention. When a problem does require the manager's attention, that problem doesn't have to wait in line behind a bunch of minor conflicts.
- This support for parallel activities means that masterly management can work with projects that are just too complicated to be understood in all their detail by a single manager—and most software projects would fall into that category.
- Because the people being managed are also delegated a self-management job, they are able to contribute observations that a micro or motherly manager is likely to miss.
- Masterly management involves managing the *project* rather than the *individuals*. Most of the time, the people doing the work are free to pick their own methods within some basic guidelines (for instance, driving on the correct side of the road, or using the corporate standard tool set). This allows creative energy that might otherwise be spent on finding ways to "beat the system" to instead go toward creating profitable products.

In short, a masterly manager like Andy observes and steers a system. If the problem is well understood, the plan is appropriate, and the people doing the work are competent, the controller often doesn't need to do much. Unlike micro and motherly managers, masterly managers spend most of their time in observation and thought rather than in frantic activity. But don't be fooled—when Andy was sitting in his chair sipping lemonade, he was more effectively in control of the situation than either Barney or Aunt Bee.

If masterly management is so good, why don't we see it more often? Because in some ways it's unsettling, especially for the manager:

Looks can be deceiving Masterly managed projects often give a certain appearance of chaos. When Andy managed the intersection, traffic was turning every which way, which was disturbing compared to the neat and orderly behavior when Barney was in charge. However, more traffic moved through the intersection, and did so more safely, under Andy's chaotic-looking management style. Many software projects already look like chaos. Will going to masterly management make them more so? We doubt it; we suspect that much of the apparent chaos in software development comes from resistance to micro and motherly management.

Power is as power does Masterly management requires a different mindset. Most people associate the word *manager* with the word *power*. Yet moving from micromanagement to masterly management involves giving up much of the apparent power and authority of the managerial position, and giving it to the people being managed. The masterly manager has more real power, according to writer Barry Oshry (quoted in Weinberg's book *Becoming a Technical Leader*), if we define power as the ability "to act in ways which enhance the capacity of our systems to thrive and develop in their environment."

Measuring what counts In some organizations (particularly those where micromanagement is the rule), a masterly manager may have a hard time getting promoted. After all, you won't be doing much visible *managing* compared to the micro and motherly managers around you, and it will be easy for the micromanager who makes promotion decisions to conclude that the project succeeded in spite of your "inaction," not because of it.

But masterly management also has rewards. Masterly managers often don't have to work as frantically as micro and motherly managers. As a masterly manager, you're less likely to find yourself in the office at three in the morning, trying to resolve yet another trivial issue. And you'll get the satisfaction of knowing that you're truly an effective leader when the project team says, "We did this ourselves."

Micro, Motherly, or Masterly Management

The best way to determine your management style is to ask questions and observe what is happening.

- Do the people reporting to you scatter like leaves in the wind when you show up? Do you feel like they are performing to the letter of the law and not the spirit? Do you jump in and start coding when there is a problem? If so, you're probably micromanaging.
- Do you organize workflow for a minimum of interaction so things go smoothly in the team? Do you step in and try to make everything all right for everybody? In crunch mode, do you revert to micromanaging? Your heart may be in the right place, but you may be in motherly managing mode.
- Do you spend a fair amount of time observing what is happening, thinking about the impact the events will have on your team and project, and planning what to do? If so, you may be masterly managing.

If you would like to change your management style, there are some important questions to think about. First, how did you come to have your current management style? For most of us, the way we manage is influenced by the people who've managed *us*, and by the environment in which we manage. Acknowledging these influences, and the constraints of your current work situation, may help you determine whether it's time for new models. It's important, too, to examine how you *feel* about your style. If you're happy with the status quo, change may not be necessary. But if you feel overworked, and seem to be constantly fighting fires, then maybe a change is in order.

And finally, what would you *like* to have happen? We saw that Barney, Bee, and Andy's views of the "problem at hand" shaped their unique responses, and the same is true for you. Once you know what you would like to have happen, you can create and implement the plans that will allow you to achieve your goals and keep

your traffic running smoothly. STQE

Don Gray (don@SystemsSmiths.com) *has been working with manufacturers forging better communications, teams, and projects for fifteen years, between whitewater kayaking trips. Dan Starr* (dcstarr@ntsource.com), *with Lucent Technologies–Bell Labs in Naperville, Illinois, has been active in the design, architecture, and human processes related to developing communications software and systems for twenty-five years. When he's not working or riding his motorcycle, he's writing a novel about time travel and hairspray.*