Is Your Project Out of Control?

A Manager's Guide to Identification and Correction of Projects Gone Awry

Microsoft®

Chris Williams (email: clwill)
Director of Product Development

Is Your Project Out of Control?

A Manager's Guide to Identification
and Correction of Projects Gone Awry

About this Document

This guide is intended to help senior level project managers, especially those with limited experience in managing software development projects, recognize the symptoms of a project that is in trouble. The key goal is to identify the problem. In many cases the corrective measures are obvious; in others, some possible solutions are presented.

This document is the result of a number of interviews with a broad spectrum of experienced project managers at Microsoft. I wish to acknowledge the wisdom and assistance of: Mike Blaylock, Adam Bosworth, David Cole, Jon De Vaan, John Fine, Denis Gilbert, John Ludwig, Steve Madigan, Paul Maritz, Merle Metcalfe, Dave Moore, Bob Muglia, Chris Peters, Jeanne Sheldon, Roger Sherman, Anthony Short, Brian Valentine, and Dave Weil.

What is control?

It would seem that to know when you are "out of control," you must understand what "in control" means. Yet this is both difficult to answer and not necessarily relevant. The point is not to measure the project, the people, or the management against some yardstick. Rather, it is to recognize when the team has made the transition from a healthy natural chaos to a point where progress is hindered by a lack of control.

As Chris Peters noted, "The default state of a project is out of control." The natural tendency is toward unhealthy chaos. Only the management of the team, with the help of the team itself, can provide the control necessary to make the project a success. This makes it difficult if you are vigilantly watching for some transition from "in control" to "out of control," because some teams are never in control in the first place.

So this document focuses on recognizing the unhealthy signs of a project that needs corrective action. It's much more a listing of symptoms than a cookbook for project management, much less a how-to than a diagnostic tool.

This balance of this document is divided into the major areas where projects can get out of control. They are:

- Focus
- Product
- Team
- Communication
- Process
- Schedule

Each section describes the key aspects that make the category important, gives assistance in identifying problems within that category, and occasionally offers some ideas to deal with these problems.

Finally, the document ends with an appendix that divides this list of problems into a timetable. This can be used as a checklist for managers at various points in the project, helping to identify that state of "out of control."

Focus

Probably the most important, and the most difficult to judge, characteristic of a well functioning team is the focus and vision to create an outstanding product. Without this focus, the team cannot execute on the underlying tasks that make up the project.

Lack of a shared, crisp vision

The one success factor mentioned by every single interviewee was the need for the team to have a clear vision for the project. This was mentioned in several ways, but the metric was always to pick a few individuals from a variety of areas on the team and ask them each to very briefly outline the vision for the project. If they cannot immediately identify, in just a few sentences, the key goals and the key customers for the project, the project is in trouble. The response to "what are we building?" should be a prompt "the fastest spreadsheet in the world," or something to that effect.

The breakdown may be because the project does not have a crisp vision, the vision has not been well communicated to the team, or the team does not agree with or believe in the vision. In any case, this is a fundamental flaw that will prove fatal to the project.

Without a vision for the project, the team will be unable to make difficult feature or bug tradeoffs, will be frustrated by communication problems in the team, and will inevitably make decisions that are misdirected because they have no foundation.

Lack of focus on the customer

Another way to look at the problem of focus is to ask, "who will pay good money for this product?" If the customer is not readily identifiable and is not a key part of the goals and vision for the product, this is another major sign that the product is in trouble. When difficult decisions arise, make sure that a customer advocate is identified in the group, and that they are heard.

The corollary to this is to be sure that you have identified who is *not* the customer. It is often too easy for someone to say, "anyone would want this," which is basically the same as saying that they have not really thought about who the customer is. Customers are not homogeneous; some group will definitely be outside your market. It's vital to identify them too.

Focus on extracurricular activities

One trap that is very easy to fall into, especially here at Microsoft, is the tendency to focus on things that are not directly related to shipping a product. Examples range from the apparently important to the obviously inane, from an inordinate focus on preparing for conferences or project reviews to angst over the color of the T-shirt.

If you find the answer to the question "what's up?" is a harried "preparing for a BillG review," you should investigate. If it comes for a long time (longer than a week) and from a large number of people (more than two or three), you have a problem. If the team is getting T-shirts for almost completing milestone 2, you should be concerned. If the end goal in sight is not a product for your customers, but a conference for your peers, you should consider some actions to refocus the team.

Certainly some creative extracurricular activities are vital to maintaining team morale and they can often help to focus the team. But if they seem the persistent focus of the group, if they occupy more than just a small window of time, or if they are the subject of controversy in the group, they are counter-productive.

Product

It's often easy to look at the product itself and judge whether the project has a good chance of success. This is where an objective third-party view, that of another experienced project manager, can help the most.

Difficulty making the transition from technology to product

Version 1.0 of any product is more inclined to be out of control, or at least appear that way, because the target can move or be unclear. Much of the difficulty can happen in the transition from technology to product. This transition, where a great piece of technology gets solidified into something a customer would pay money for, is, from all experiences here, a very difficult one.

The key flag for a product that is not successfully making this transition is a team that is in love with the technology. They are so enamored of it that they can't articulately identify who the customer is, why the customer would want it, and how their technology fits into a business model. They often redirect questions in this area to demonstrations of the technology, or to detailed discussions of how it works.

People in love with the technology also often have a difficult time discussing version 2.0 or 3.0 of the product. They've become so engrossed in what it is today that they haven't thought about how it will be a business in the long run.

Biting off too much

If you review the project with an experienced project manager and their response is that this looks very hard, this is a danger sign. Every project shouldn't be easy, but then again, they should be doable.

There is a tendency, especially again with version 1.0 projects, to feel compelled to make a big hit with the product. This can lead to over-ambitious projects that are destined to have difficult births. Often this is caused by fear of a competitor. We worry that we'll make an inadequate response to their latest release, and that we'll be a failure in the marketplace.

But one of Microsoft's key strengths is our persistence and willingness to stick with products, even those that start with humble beginnings (Windows, Word for Windows, Windows NT). This should provide comfort and cushion for projects that need to limit their round one goals to a more achievable set.

The solution is to make a project plan that covers several manageable releases. Plan version 1.0 to build a solid foundation and establish a position in the market. Leave for version 2.0 or 3.0 those features that will make the project late, or that require time in the marketplace to jell.

This is a tough decision. Everyone wants to make a splash. But experienced managers will all tell you that, after five years, they would rather be working on version 3.0 of a product than working on the fifth year of version 1.0.

Too many dependencies

Dependencies on other projects are an unavoidable fact of life. And as we, as a company, move to a more component architecture, the number of dependencies for any one project is increasing. This is clearly a good goal as we can leverage our size and allow groups to specialize in areas of their expertise.

Years of experience at Microsoft have taught the most veteran Program Managers to avoid dependencies at all costs. This is clearly an exaggeration and can lead to rampant duplication of effort. But the lesson is clear: a project with key functionality being delivered by another group needs to manage that dependency very closely. And a project with most of its functionality in the form of dependencies is a project likely to be in trouble.

No, or very little, specs

This is becoming rarer at Microsoft, but the tendency to "fly by the seat of your pants" is a danger sign. No project should exist without some common point of reference, and a spec is a very good example of one. Especially at the beginning of a project, before coding has begun, the spec is the only deliverable, and the only real sign of progress in the group. Once well into a project, the product becomes a more visible indicator, and should be trusted far more than any document, but at the beginning some documentation is a must.

Extremely detailed spec

The converse to having no specs is having an encyclopedic spec. Communicating the project to the team and its dependents is key, but an extremely detailed spec can be a problem for a couple of reasons.

An overly detailed spec is a warning sign because it can, itself, become the deliverable. The team can get overly focused on updating it, on filling in every little hole, or on producing regular updates. It is an indicator that the team has lost the focus on the end product, and has forgotten that the key deliverable is a product for our customers.

Another way that too much spec can be less than a good thing is that it breeds overconfidence. By having an apparently bullet-proof spec, a team can fail to prepare for the inevitable calamity or change in direction. They will schedule things with inadequate buffer, they will be resistant to changing it for all but the most dire of causes, and will lose sensitivity to the market they are trying to serve.

So how much is too much? This obviously varies with the product, but the warning signs are fairly easy to spot. Is the spec in numbered volumes? Has the spec been read, really read, by anyone other than the author? Is the spec the review objective for the program manager, as opposed to a good feature in the product? Is updating the spec someone's full-time job? These are not necessarily fatal, but often indicate a problem.

The vision in the wind

Often used as the excuse for a project's problems, but only occasionally correct, is the project whose vision changes every few months. This has been the downfall of more than one project, and is usually blamed on "management." The reality is more often that the project never had a clear vision in the first place, or the vision failed to take into account some crucial elements (the customer, for example).

Detecting this problem is rarely difficult, teams will usually howl in pain with each change of the vision. Clearly changes in the individual features or minor project goals are commonplace. The time to worry is when the fundamental direction of the project changes but the shipment deadline doesn't.

Not managing product performance and size

Features can be debatable: which should be in, which should be out, it's always a question. But the areas of product functionality that are constant and most frequently overlooked are performance and size. Failure to pay heed to these issues will always come back to haunt the team, and can be an early warning sign to a product doomed to be out of control.

The product team must clearly specify performance and size goals for every significant part of the product. If they are not managing these issues by monitoring their progress toward these goals with their daily builds, they are going to be surprised at the end of the project. If they have no time in the schedule to fix performance and size problems, they will likely slip. If they have no people to worry about this as their primary job function, they are likely to ship a pig of a product. In any of these cases, they are not in control.

Team

The caliber of the team was often called out by the interviewees as a key success factor. While this is relatively obvious, the signs of a problem in this area are less so.

No shipping experience among the senior team leaders

Shipping software, especially here at Microsoft, is a complicated art. It requires a combination of skills and experience that are fundamentally impossible to attain elsewhere. The skills range from technical to political, and the experience can range from the smallest products to major releases. But there is no replacement for people who've lived through it.

If the team does not have people who have shipped software *here at Microsoft* in at least half of the key positions (development manager, test manager, group program manager, user education manager, and/or product or business unit manager), they are likely to have a very difficult time.

Lack of technical skills to produce the product

This is rarely a problem here, since we tend to be so technologically focused, but an obvious source of problems is a team that does not have, or has lost, the technologists to produce the product. Key warning signs include black box technologies ("we don't want to change that code, no one knows how it works") or overly sensitive technologies ("we'll do anything to avoid touching that code, it's very delicate").

No owners

Each core technology should have an individual who can always be turned to in that area. They should "own" the technology or feature, and their ownership should be well known by the whole team, and all dependents.

They don't have to be a lead or manager, they are often just a star member of the team who understands the details and communicates them well. They usually feel a strong vested interest in its success, and are often the cheerleader and staunchest supporter of at least the feature, often the product as a whole. They wear the label as "the person to go to" on any issue relating to that area.

Inadequate staffing

An obvious warning sign of the health of a team is open or inadequate headcount in key areas. A team with key positions (the business/product unit manager, development manager, test manager, group program manager, or UE manager) left unfilled for a long period of time will find it impossible to set and maintain direction. A team with more than one of these roles unfilled, even for a short period of time, is in jeopardy.

A team with no testers, or even one with substantially fewer testers than developers, is likely to have a difficult time producing a quality product. A team with developers writing specs as their key job, or with program managers writing code as theirs, is understaffed or misdirected in a serious way. Even a team with a high percentage of temporaries (greater than 20%) is putting a great deal of the intellectual horsepower in a volatile resource. All of these require reallocation of resources to succeed.

Hostages

Much like a team without key people, a team where the key contributors want to leave, but are being held against their will, is destined for disaster. It's often better to just admit the problem and let them go, at least they won't drag the rest of the team down into their bad morale pit.

Communication

Communication, both inside and outside the team, is obviously key. During a post-mortem people often say "there was a lack of communication." It's hard to know what that really means. Here are some clues.

The "reality distortion field"

A key communication problem within the team, one that seems far too prevalent here at Microsoft, is fondly known as the "reality distortion field." This is when a team, engrossed in its own magnificence, convinces itself that impossible dates can be met, that enormously complex technical problems are nothing to worry about, and that naysayers just "don't have the religion."

This occurs in varying degrees which makes it often difficult to detect. It shows its true colors when challenges to the schedule, business model, or technology are not carefully considered but rather are discarded as the ranting of a non-believer. Often the retort is "you just don't understand." Obviously the appropriate reaction is to build understanding, but if this is not forthcoming, there may be a need for some cold, hard reality. A thorough project review with a skeptical audience can do wonders toward dissipating the field.

Lack of communication among dependents

The most important part of working with groups that you are dependent on, or who are dependent on you, is communication. It's easy to understand the importance of getting the programmatic interfaces correct, or to see why planning the dates is vital, but none of this will actually happen without a huge investment in communication.

Discovering whether you have a communication problem with your dependents is actually quite easy. Among the key questions to ask:

- When is the next drop? The answer should be unequivocal and the date should be before your next drop.
- Who in our group attends their staff meetings? The answer should be at least one of your program managers, maybe a developer or two as well.
- How many bugs do we have on their bug list? The answer isn't important, but someone on your team should know it.
- How do they prioritize our bugs? The answer should be "like their own."

In any case, your project is in jeopardy if you don't manage these communication issues very well.

Inordinate emphasis on secrecy

One of the clues to a team that is in trouble is a team that's trying to hide things. Clearly some details of the technology can or should remain confidential, but a team that is overly protective of their specs or schedule is hiding something. And it's not likely to be something good.

Ask for a look at the spec. You don't have to read it (but you probably should), it's just a test to see if they'll give it to you. Ask to go over the schedule. If they let you, it's probably OK. If they don't you should be alarmed. If you're not on the team, maybe there are good reasons. But it's time to really be alarmed if you're on the same team and you can't get access to things. Something is wrong.

Status reports that blame others

If every month the status report looks bleaker and bleaker, yet the problems are always someone else's fault, there's a problem. There's little question that a team can be derailed by the failings of their dependencies, but if it happens continuously month-after-month, there is a problem. They should be making alternate plans, falling back on their contingent strategies, doing something. But throwing up their hands and blaming their woes on others is a clear sign of a team that is out of control.

Process

Much of the wisdom relating to the development process is well known, but it deserves repeating. Often the difference between in and out of control is in the details.

Not using milestones and milestone reviews

Virtually every team at Microsoft divides their projects into milestones and uses these milestones to track the progress of their development. However, many teams also cheat on their milestones, which of course only causes problems down the line.

There are several common milestone cheats:

- Fudging the end date ("we didn't really say the 3rd, we said 'August'")
- Declaring victory ("well most of the team made it")
- Not having definitive quality objectives ("so what if we have 1,500 bugs, the code is complete")
- Moving things that didn't make it into the next milestone without careful evaluation, and without a complete reschedule of the rest of the project.

Teams also commonly fail to come to closure on milestones by not holding milestone reviews. These reviews, also referred to as milestone post-mortems, are extremely important sources of corrective action for the next milestone. Failure to hold them, or to reset the schedule during them, is a serious warning sign.

Lack of regular (daily) builds

Time has shown the daily build of the product to be one of the most important diagnostic tools for the health of the product. Having a daily build does not ensure the health of the project, but few well-run projects survive without one.

Key to the daily build are the daily sniff tests (or smoke tests). Stolen from the electronics industry where people would plug in a board and see what smoked (or smelled bad), sniff tests are a vital part of the build.

These tests ensure the basic stability of the product by checking daily that nothing has been checked in that jeopardizes the basic product. If something is uncovered, it is repaired immediately. This provides a foundation for the new work to be added each day, and ensures that the project doesn't stray too far from a stable working version, requiring days of corrective measures.

Sniff tests should also be built to be run by the developers before checking in code. This prevents breaking the daily build and also encourages developers to take more responsibility for the stability of the product.

The importance of a regular (preferably daily) build cannot be underestimated. A good way of thinking about this is to consider how much time it would take to recover from a fatal flaw in a check-in. If you build weekly, and someone checks in something stupid on Monday that you don't find until you build on Friday, a week of work might have to be repaired before health is restored. You've lost two weeks. If you built daily, it would be two days.

It's also important to start daily builds very early in the process, preferably from day one on new versions of existing projects. Think of it as getting the project to a known state and ensuring that it stays there.

Not dogfooding early and often

Eating your own dog food, that is making your team run, live, and breathe their own product, has been shown to be a vital method of ensuring the health of a project. Some people claim that this is only viable with apps people use in their daily life (such as the operating system, or a compiler), but the reality is that you can make daily and frequent use of the product a key goal for every team member.

You can achieve dog food in a variety of ways. Have a database? Use it to track your bugs. Have a spreadsheet? Use it for your schedule. Have a game? Make sure people are using it daily, taking it home, and sharing with their friends.

Dogfooding is so important that some teams can actually judge the ship date from the day they first start seriously doing it. For example, operating systems, some feel, are a year from release when dogfooding begins. Whatever your metric, if you're not doing it, you're in trouble.

Not tracking some metrics

It's easy to get carried away with tracking bug counts, lines of code, person-hours per bug fixed, and on and on, but that's not a big risk here at Microsoft. We tend to be so antibureaucracy, that we shun this kind of process analysis. That's too bad, because some attention to metrics can help a great deal.

The team should pick some metrics to track and should widely publish them regularly, at least monthly, preferably much more frequently. This can be bug arrival rates, bug closure rates, whatever, but tracking the process is essential. If the team is "flying blind" they'll never get the plane off the ground.

The key to metrics is to track something meaningful. To do this, you should:

- Decide the overall goal that you want to accomplish. This could be as straightforward as "cutting priority 1 bugs," or as complex as "increasing customer satisfaction."
- Determine a metric that will display whether you are meeting your objective. Define what it means if the number goes up, and if the number goes down, and what you might do about it.
- Track it, report it on a regular basis, and review your performance using it.
- Change how you are working if you aren't meeting your objective. If you can't envision this happening, you're tracking the wrong metric.

The pitfall here is to track metrics for their own sake, or to track things that don't optimize your team's behavior or measure your progress towards shipping. Tracking the wrong thing can be worse than tracking nothing, because you may lure yourself into a false sense of accomplishment.

Missed transitions of responsibility

There is a "center of gravity" for projects that can be observed fairly easily, and the PM team should be following that center as it moves over time. This center begins in program management as the team designs the project, and specs it. It then moves to development as they code the project, and then to test as they help to polish it for shipment.

The PMs on the team should be making the transitions with the center. As the focus moves to the coding team, PM should be working to close down issues, triaging bugs, and responding to difficulties the development team has in implementing the spec. As the focus changes to test, PM should focus on triaging bugs and preparing the product for shipment.

If you find the PMs worrying about new features after code complete, or planning the next version instead of closing down bugs, they are not helping to ship the product. Either they need to be part of a separate team focused on the next release, or they should be more closely working on the current release.

Missing the finer points of process control

Few teams truly handle all aspects of their process well. This is not a fatal flaw, but it can be used as an indicator of the maturity of the development team. There are many fine points that should not be overlooked. Some examples, not that your team *must* follow, but rather are used by many teams:

- After code complete, all check-ins must have a raid number.

- After some point in the process, every check-in must be reviewed by at least one other person (some teams make the requirement that it be a lead).
- No bug is dismissed as "non-reproducible" without consent of the test manager/lead.
- No bugs are dismissed en masse, every bugs gets a fair and honest hearing. Teams that blow off huge numbers of bugs often regret it, or read about it in InfoWorld.
- After some point, all feature/interface changes must be approved by a "change control group."
- If the number of bugs exceeds the number that the development team can fix in a reasonable period (e.g. ten days), work on new features stops and the development and test teams work on fixing and closing bugs

These are just examples, but your team should have a list of these kinds of rules, and should be following them.

Schedule

Non-"bottom-up" schedule

One of the more reliable predictors of the health of a project can be to determine where the schedule came from. If the schedule came from anywhere other than the people who will actually do the work (with appropriate buffering applied), it is wrong. Period.

Common places for a bad schedule to come from include:

- Marketing often determines when a product "has to ship." While this is valuable input into the product development process, date-driven milestones are always the source of problems. If coupled with explicit features, as it usually is, it is a recipe for disaster. The only three variables are time, resources, and product. Teams here have relatively finite resources, you cannot restrict all three variables of the equation.
- Management occasionally overrules the team and decides what the schedule "should be." While management is another valuable source of input into the process, it cannot successfully dictate the schedule. Management *can* greatly assist by helping to adjust resources, priorities, and other aspects of the process, but it should not force a schedule on the team.
- Program Management is chartered with removing obstacles to shipment, and is given responsibility for much of the shipping process, so they sometimes mistake their role for that of schedule-setter. Unfortunately, as with other people who are not actually performing the task, their estimation skills usually fall short.
- Even if the schedule was prepared by the responsible managers and leads, it's suspect. The schedule must represent the commitments of the people who are going to do the actual work.

Regardless of the source, if it's not the team (the developers, testers, writers, etc.), the schedule is wrong. Not suspect, not inaccurate, it's wrong.

How do you determine the source of the schedule? Ask several line team members what they think of the schedule, one-on-one. If they agree with it, and have honestly bought into it, they most likely had a hand in creating it. If they scoff, roll their eyes, or refuse to discuss it, the schedule is invalid.

Magic dates

A warning sign of a bogus schedule is magic dates. Such dates tend be in two forms, the “when do you need it” form, and the “would you look at that” form.

The “when do you need it” form of magic date is the schedule date that miraculously meets the date management said they needed it. To be fair, the team could have carefully manipulated the resources, features, and teamwork to make the date work out. But even the best laid plans result in a schedule that only closely approximates the actual “needed by” date. More than likely what has happened is that the team has felt the need to acquiesce to what they feel is a management-driven date, and thrown their estimates away.

The “would you look at that” form of magic date is the schedule for several disparate components that all miraculously converge to be completed on some date. Realistically even under the best of circumstances, convergence of a number of components will happen at only roughly the same date. If you find yourself looking at a schedule where they all magically make the same date, you should be very suspicious.

Schedule not detailed enough

If the schedule is in increments longer than a week (five work days), per person, the schedule is not detailed enough. Letting people run without a schedule checkpoint weekly is risking their being off by as much as a week. Compound that by all the people on the project and the schedule is far too fuzzy.

This is such a key issue that some people insist on even smaller increments (2-3 days). Whatever the granularity, the key is to ensure that the schedule is accurate to the amount of time you’re willing to slip.

Schedule churn

Somewhat the converse of the not detailed enough schedule is a team that is overly fixated on the schedule. Clearly some attention to the schedule is very important. But too much of anything is a dangerous thing.

The team with this problem tends to release new versions of the schedule very frequently (two or more times per week). The schedule tends to be too detailed, listing people’s tasks by the hour, or some other fine granularity. They tend to make adjustments to the entire project schedule that are in very small increments, such as days. In all, this indicates a lack of focus on the product, and too much focus on the process.

The other serious danger of schedule churn is “crying wolf.” Frequent date changes get the team, the managers, the customers even, all believing that you are out of control. They won’t believe any date you give, and they are justified.

Lots of items assigned to TBH or TBD

This is one of the key warning signs of a schedule that is fictitious. TBH (to be hired) doesn't happen overnight, and unless magic happens, they will be hired and up-to-speed too late to be useful. TBD (to be determined) is a sign of one of several problems: inadequate resources, inadequate planning, too much product, or not enough willpower to cut features. In all cases this is a dangerous warning sign.

How do you know when this is a problem? Are key features or bugs assigned to TBH/TBD? (One of the more common areas that gets this treatment is Setup, the first thing every customer sees.) Is more than a tiny percentage (less than about 1%) of the project unassigned? Is TBH/TBD used for more than a month as placeholders? These are signs that the project may be in trouble.

Two sets of books

Some managers feel the need to lie to their teams about the “real” date. They have in their head the date they expect the work to be done, they may even tell their boss this date, but they tell the team some earlier date. They excuse this blatant abuse of trust by insisting that it makes the team work harder. “They wouldn't come in on weekends or really bust it if I told them the real date.”

This is simply horrible management. You can't trust the team with the truth, but you're willing to trust the team with the product. Makes no sense.

It has been proven time and again that teams can and do respond well to liberal doses of the truth. Explain the details of the schedule, define the pressures on the team, involve them in the decisions, and track the progress daily with them. Every team dealt with in this way responds like the professionals they are.

Buffer misuse

Buffers are there for bugs, or for performance and size problems (which are just a class of bug), not for completing features that are running late. Using up the buffer for feature work, and then saying “we made the milestone” is just lying to yourself (see the discussion of milestones above). When will the bugs in those features be fixed?

The team should readjust the schedule when new items are added or items take more time than allotted, not just eat up the buffer. Again, the buffer is for bugs and issues, not “everything else.”

Failure to schedule “busy work”

Forgetting to schedule things like holidays, vacations, etc. can be a real problem. Even important tasks like doc reviews, code reviews, and group meetings can get left off the schedule. People will often pass over these things as “just part of the buffer,” or something to that effect. There will be enough things that just “come up” that you should schedule anything you can anticipate.

"We'll make it up"

Commonly heard at or near the completion of a milestone. Usually preceded by "that's OK…" "Making it up" has never happened in recorded history (at least not when "it" is anything substantial). There's no reason to believe your team will be the first.

Spend the time to completely reevaluate the schedule, from the bottom up, and come up with a whole new schedule. No other remedy is safe.

"Fix – if time"

A parallel problem to "we'll make it up" is bugs assigned to "fix – if time." The bugs will just sit on the list, there will never be time, and they will eat away at you, adding to that massive bug count that is making the whole team sick to their stomach. Be honest with yourself, decide to fix it or decide to postpone it. Make a decision.

Sometimes this is due to a team not having decided what the quality bar is and they can't decide whether this is something that needs to be fixed. Setting this bar early helps provide a consistent tone to the whole project, and increases the chances of success.

Summary

Few teams will exhibit all of these symptoms, and the exhibition of one or two does not necessarily spell doom for the project. The way to use this list is to be as objective as possible and use it as a diagnostic tool to determine the health of your team.

If you feel that the team is out of control, you have many options. You can and should take prompt action, either on your own or with assistance. You can enlist the help of your manager, the division VP, other experienced managers at Microsoft, or the Directors in the Product Development Resources group (email PDRDIR). The key thing is to do something. Remember, entropy favors the state of "out of control."

Out of Control Checklist

This list is the same as the list above, but presented in chronological order.

Spec Complete Milestone

Especially important when coding is about to begin are:

- Lack of a shared, crisp vision
- Lack of focus on the customer
- Difficulty making the transition from technology to product
- Biting off too much
- Too many dependencies
- No, or very little, specs
- Extremely detailed spec
- Not managing product performance and size
- No shipping experience among the senior team leaders
- Lack of technical skills to produce the product
- Inadequate staffing
- The “reality distortion field”
- Not using milestones and milestone reviews
- Non-“bottom-up” schedule
- Magic dates
- Schedule not detailed enough
- Lots of items assigned to TBH or TBD
- Failure to schedule “busy work”

Coding or Code Complete Milestone

During coding, at coding milestones, or at code complete watch for:

- Focus on extracurricular activities
- The vision in the wind
- Biting off too much
- Too many dependencies
- Not managing product performance and size
- No owners
- Hostages
- Lack of communication among dependents
- Inordinate emphasis on secrecy

- Status reports that blame others
- Not using milestones and milestone reviews
- Lack of regular (daily) builds
- Not dogfooding early and often
- Not tracking some metrics
- Missing the finer points of process control
- Schedule churn
- Lots of items assigned to TBH or TBD
- Two sets of books
- Buffer misuse
- “We’ll make it up”
- “Fix – if time”

Beta or Release Candidate Milestone

And in the final stages, be especially sensitive to:

- Lack of focus on the customer
- Focus on extracurricular activities
- Not managing product performance and size
- Hostages
- Lack of communication among dependents
- Not tracking some metrics
- Missing the finer points of process control
- “Fix – if time”