# How to Save a Failing IT Project

John Levy, Ph.D.





Welcome to the webinar How to Save a Failing IT Project. I'm John Levy and I'll be guiding you through this journey into what you can do about a project that's not succeeding.

. I hope you're ready to respond to the questions I'll put out to you as we go along.

Just to review how you can ask questions during the webinar … look for …….

OK? Let's get started.

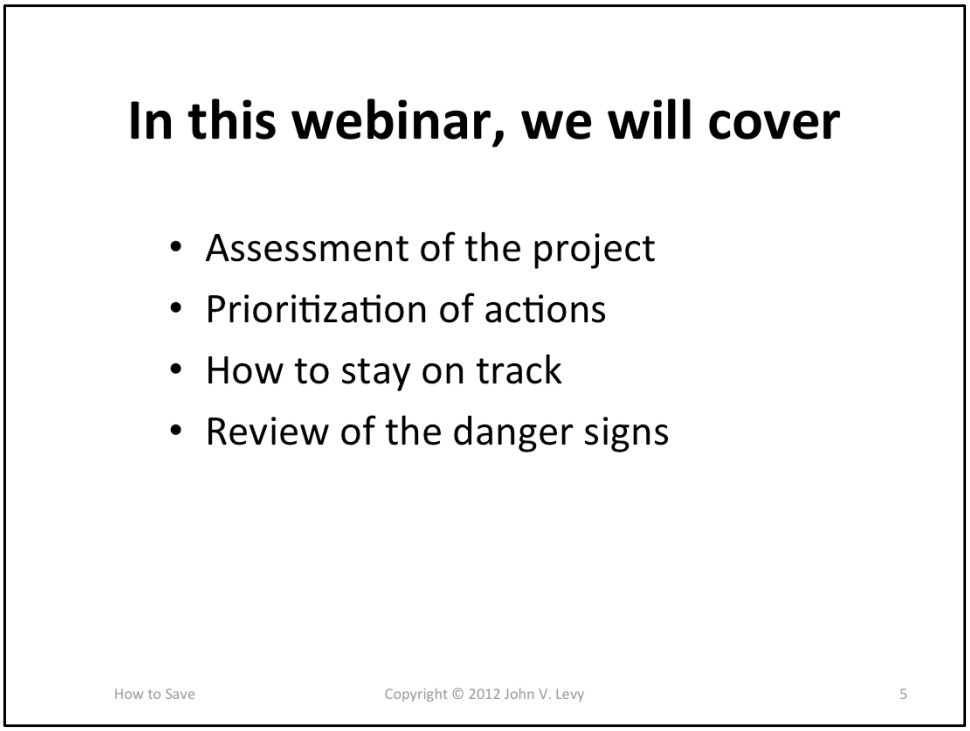
## In the last webinar, we talked about

- Why do IT projects fail?
- What indicators are most important?
- What do the indicators mean?
- What should I do about them?





What we addressed in the previous webinar were these questions. Why projects fail, how you can notice that they're heading towards failure (that is, the indicators); and then what the indicators really mean about the projects. Finally, we'll talk about what you can do about the situation that each of these indicators points to.




Now I'm assuming that you HAVE a project that is failing – or at least that you're worried that the project you're working on COULD fail. After all, we all worry about things going wrong on our projects, so it is reasonable to imagine the worst.

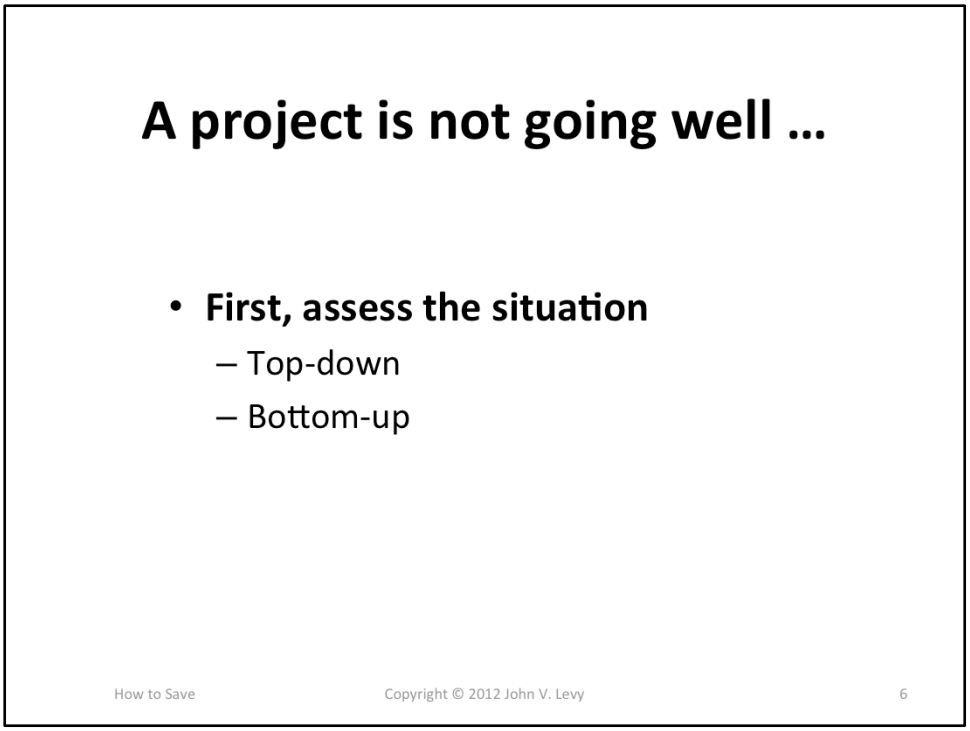
What we'll be addressing today are these items.

Assessment

Making an action list and prioritizing the actions

How to follow up and keep things on track

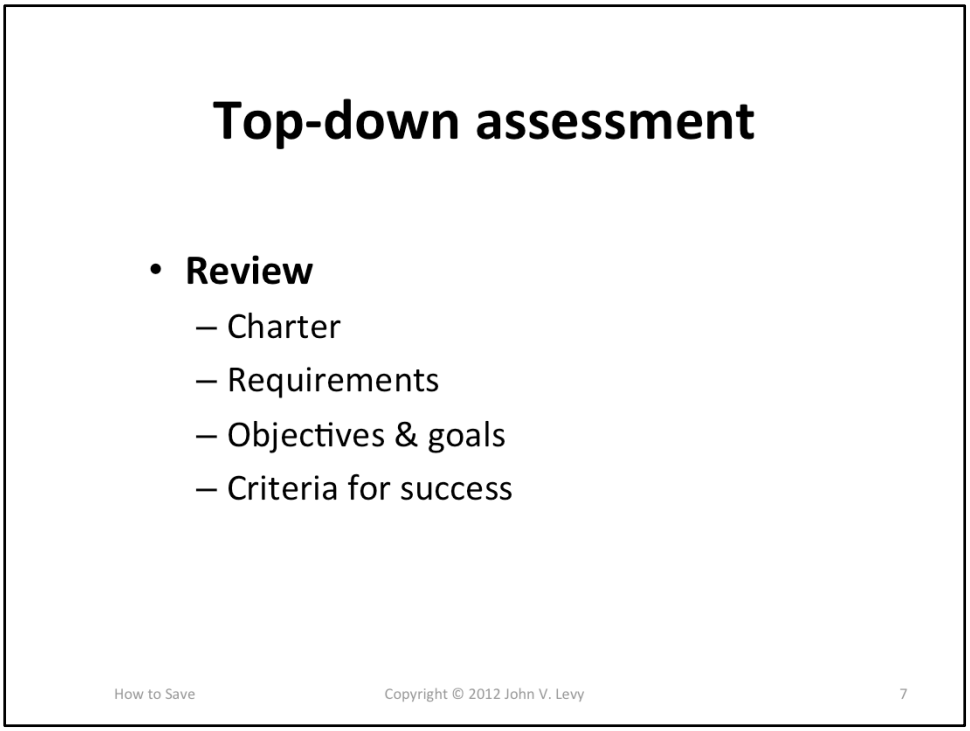
And we'll finish by going back to the 10 danger signs that were discussed in the previous webinar, so we can see how they relate to what we cover in this webinar.



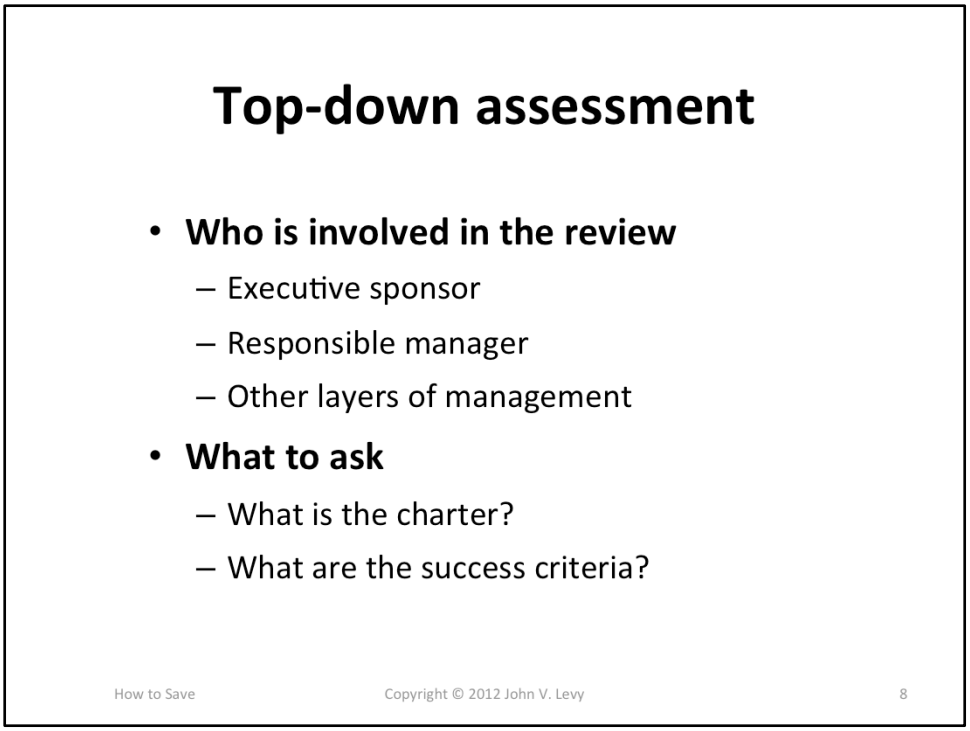

OK, so you have a project that is not cruising along. Before you step in to take action, you need to know what's going on.

You may think you already know what's going on – after all, you have seen late delivery of results, for example – but it's always best to get a clear view of all aspects of the project. So let's go over how to assess a project, pretending that you're unfamiliar with it. This is the way you would evaluate a project if you were a consultant brought in to look at it and help save it.

We'll do two kinds of assessment, one of them "top-down", which means from the upper management down to the team; and the other "bottom-up" meaning from the team up and outward to others.



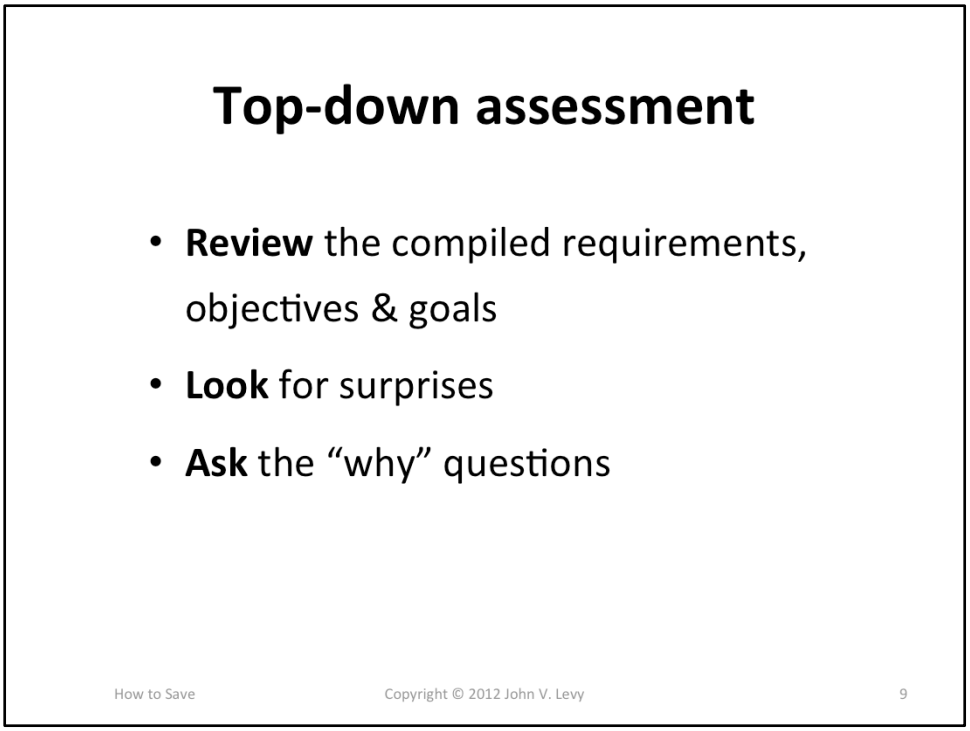

The first step in a top-down assessment is to get the charter & success criteria. What is this project supposed to accomplish? How will the results be evaluated for the business? If there is no written charter or list of criteria for success, the first thing you should do is create them. **It will not make sense to assess the project before you know what it is supposed to do.** Find someone who has a charter or an idea of the desired outcome, and write it down. If no one has the charter, **write one on your own** and review it with the sponsors of the project. Refine it until they agree that it states what they expect or desire from the project. Next, find the project's requirements list. These should be written down, and you should expect that they have changed since the project started. So if you are given a list of requirements that were written at the outset of the project, ask for the updates. If there are no written updates, start taking notes on what's been changed. Make your own updated list and refine it as you talk to the sponsor, the managers and the team. Pay attention to the fact **that there may be CONFLICTING requirements** in the list. Continue on to OBJECTIVES and GOALS, if these are different from the charter and success criteria. Gather as many different views on objectives & goals as you can, and compare them. Finally, review the success criteria, if you have them. If not, then again you may want to ask about how a project succeeds or what criteria are used to determine success in this environment and organization. Compile the list, and think about whether they conflict with each other; and whether they relate to the charter. You may need to go back through the whole sequence again to clarify things.




As you do the top-down assessment, start with the executive sponsor (if there is one). If there is no executive sponsor, find the manager who **authorized and is paying for** the project

Review the charter & success criteria with next-level management, one layer at a time (VP, Director, Manager, Supervisor, team leader). The goal is to clarify what the charter & success criteria mean to each person. Add clarifying notes.

While reviewing charter & success criteria with management, you should also gather requirements, objectives & goals, in any form that you can get them. Be sure you understand the **context** of each requirement, objective and goal, and add notes as needed to clarify them.

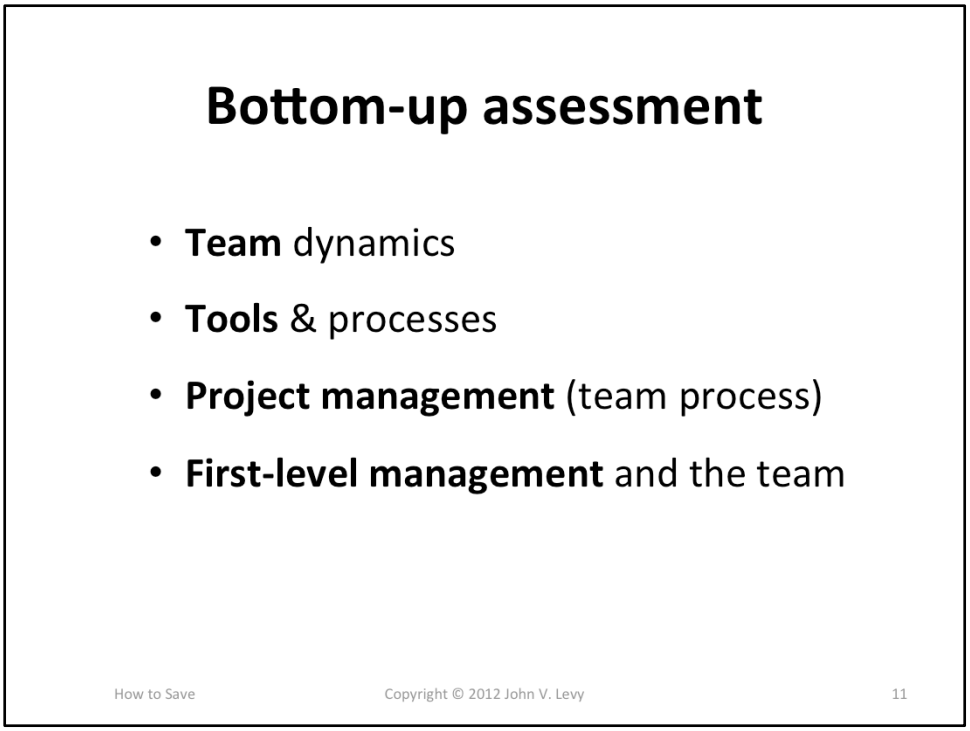



After you have gathered all of the requirements and objectives for the project, you'll want to schedule **a half-day review meeting** for requirements, objectives & goals with the development or integration team.

At the meeting, make notes about which items were a surprise to the team, **why** they were a surprise, and what the team suggests or plans to do about it. Then add to the list all team-generated requirements, objectives & goals, and make notes on the context or "impact" as viewed by the team.

You get context by asking "**why is this goal part of the project**?" This question will often elicit new requirements or success criteria.

When you have finished all of this, you can move on to the bottom-up assessment.




OK, let's move on the the bottom-up assessment.

You have started off with a top-down assessment that resulted in a list of requirements and other inputs to the project. Now you want to inspect what's going on and see how it relates to what is on your lists.

In particular, here are four areas to consider: team dynamics, the tools and processes used by the team, how the team manages the project (which is what I mean by **team process**), and how the team interacts with its manager or managers.

## Bottom-up assessment

- **Team dynamics**
  - Watch them work
  - How do they deal with problems?
  - Any conflicts? How resolved?

As Yogi Berra said, "You can observe a lot by watching."

Watch the team while it is working. Once you've seen a lot of teams working, you will be able to **sense** how well the team is working together. You'll see whether there are **conflicts** between particular team members, and whether the team deals well with **technical hitches** during the development work.

Pay attention to how the team views the leader and the management. Talk with team members individually, and also as a group.

Ask open-ended questions of team members individually about
how the work is going;
what the hang-ups are in getting things done; and
whether there are members of the team who get in the way of progress.

Notice who leads things REALLY; often, a team will depend on one or two people who are not officially leading the team.

## Bottom-up assessment

- **Tools & process**
  - Know the tools?
  - Modern tools?
  - Enough automation?





Check the level of sophistication of the tools being used by the team. See if they're easy to use.

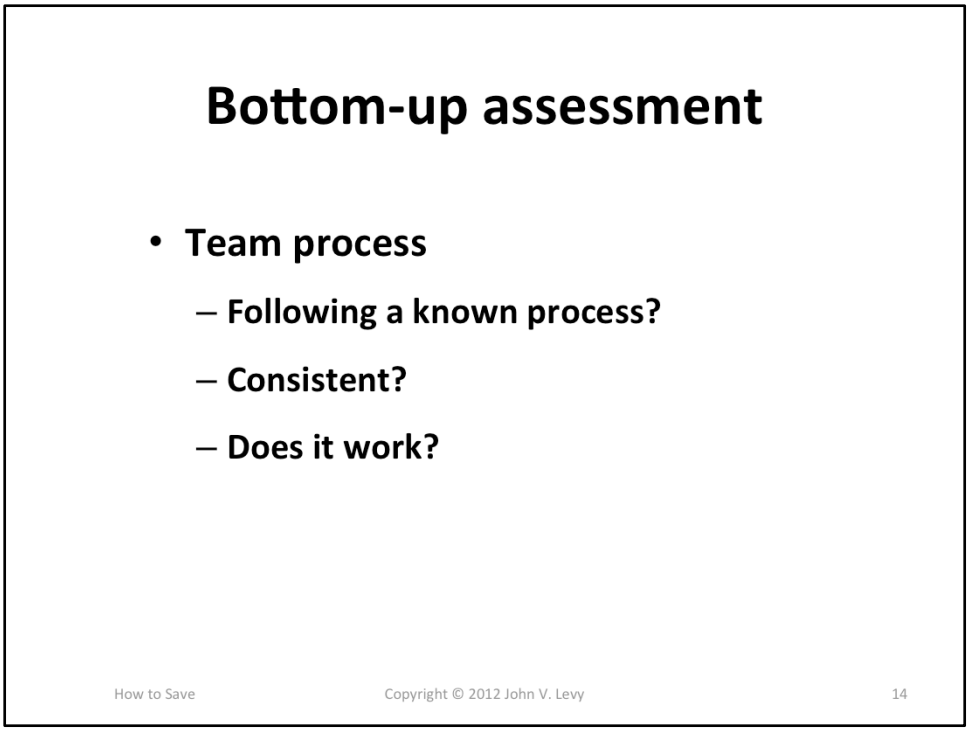
The best development teams use current, up-to-date Integrated Development Environments (IDEs) and has automation for performing tests. Check to see if everyone on the team knows how to use the tools.

Does everyone use the same tools? Sometimes, individuals will use their own customized environments. This isn't a problem unless it causes incompatibilities in the code, or if it means that some team members can't read the code of other team members.

Automation should make it easy to submit and test new code. Reporting on the results of testing should be automatic, too.

Ask open-ended questions of team members individually about how the work is going; what the hang-ups are; and whether there are members of the team who get in the way of progress.

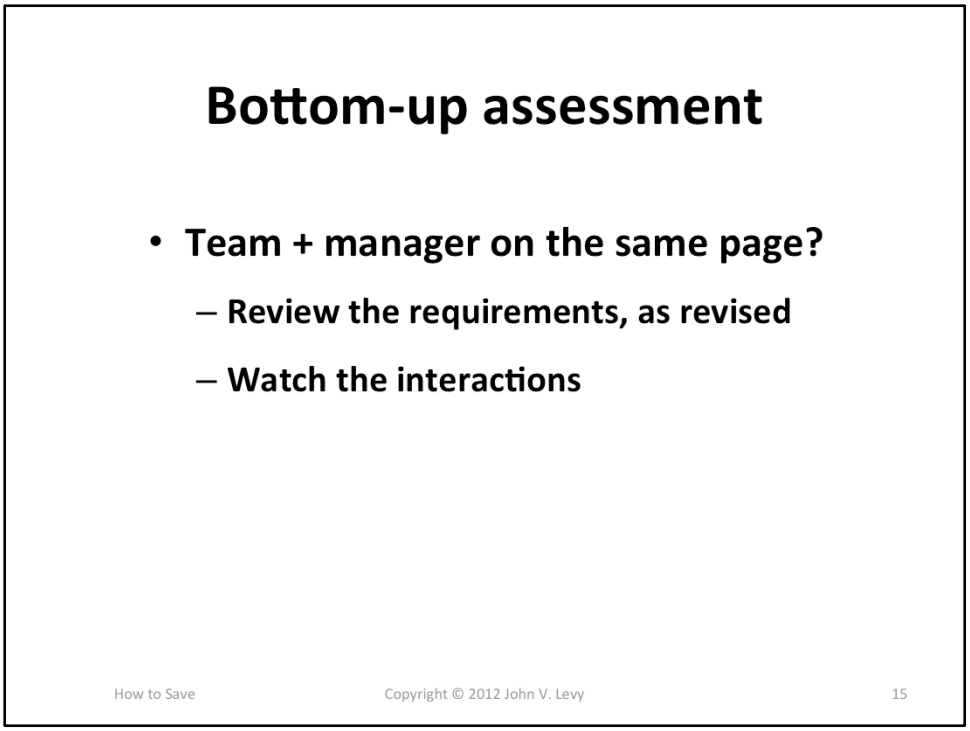
Find out how long it takes from code-submission to integration and test. Is it measured in minutes or in hours? Observe the process of code-submission and system build or system integration. Note whether there are re-starts during the process (starting over because something went wrong).




Your next question has to do with the process the team is following. If it's an Agile team, it should be examining its own process at the end of every sprint or iteration, and making small changes in order to improve the process. In every team there should be a known, consistent process to follow.

To see how well the process is working, have a look at the **bug log**. Note how many priority 1 bugs are outstanding. Ask about the **history** of the bug backlog: is it increasing, decreasing or remaining about the same? Ask how many bugs are created each time there is new code to fix prior bugs (on average).

Observe whether the team uses a **Kanban** board, **sticky-notes** with tasks, or other observable **status** information that everyone can see. If the team is using a form of Agile development, ask what the **velocity** of the team is, and how it has changed over time. The team should be happy to show you how well they are performing, particularly if they are improving their performance over time.




The final step of a bottom-up assessment is to review how the first-level manager and the team interact. Is this manager the same person as the sponsor? The same as the project leader? There is no one best way here, but you want to know what the relationships are.

Observe the team's interactions with other managers, and with outside suppliers of components, and with peer groups. You're looking for any indications of dysfunction, such as an attitude of superiority – or inferiority – between the team and others.

Finally, review the now-revised & annotated requirements, criteria for success, requirements, objectives & goals with the **sponsor** and other levels of management. Ask about the manager's view of team function.

You can now report on your assessment, and give feedback on what you found to the manager or to whoever asked for the assessment.

## Create an action list

- Publish the revised requirements, objectives, goals & success criteria
- Review checklist for actions to take

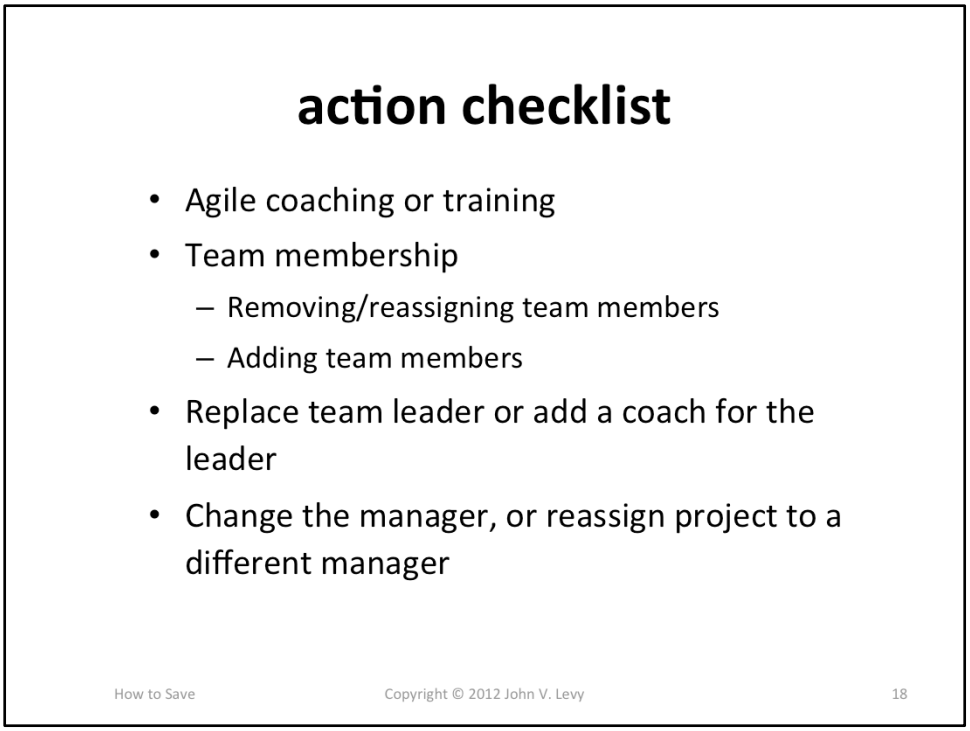




Now that you have finished an assessment, it's time to take action.

The list of possible actions is endless, so all we can do is examine some of the actions you might take. I think you'll find what you need on this list, because you can use the list that's coming to trigger your thinking for other useful actions.

So once you have created a revised list of requirements, you should publish them to the team and to all stakeholders. If you can, have someone take responsibility for keeping the requirement list up to date.




OK, let's move on to the action checklist. Here are some of the actions you can take when a team is underperforming or there are conflicts among the team or between the team and the managers.

Since I believe strongly in Agile principles, the first suggestion is to get coaching or training in Agile methods, and to launch or improve Agile processes in your organization. There are very good coaches out there, and if you need recommendations, feel free to contact me.

Within the team, you may have to move people around, including outplacement of people who don't work well in THIS team. Often, someone who is having problems in a team can find a group to work with where they can be productive, so don't give up on the individual when you are improving the team by outplacement.

Also you may need new team members to fill in technical or process areas that are weak now in the team. And you may need a different team leader. As an alternative, consider getting a temporary coach for the team leader, so the leader can improve while still leading the team.

If the problem is the interactions between the team and the manager, consider moving the project to a different manager, or hire a different manager for this project.

## action checklist

- Add automation tools
- Reduce scope of project
- Break project into separate sub-projects
- Move development team into a private space





Now let's consider actions that might improve the project speed and team functioning.

If the process is not automated enough, add automation tools.

Often a project bogs down because it is too large. If this is the case, reduce its scope; or break up the project into separable parts, and then charter each one as a project.

If you're running an Agile team and the team is trying to work from cubicles that are near other non-team people, consider moving the team into a space where they are more isolated from others, but close together as a team. This rearrangement can work wonders on team morale and efficiency.

## action checklist

- Reinitiate definition and charter of the project – change requirements, objectives & goals
- Adjust the way the development team interacts with support groups





You started the assessment with finding the charter and requirements of the project. If the project is not progressing well, consider restarting the process. Redefine the project to include only the essential things that will make an immediate difference to the business. Find simple but appropriate criteria for success.

Interaction with other groups can also be a sore point in some projects. If it is in your project, consider asking for intervention from upper management, or from HR, in smoothing out the connection with a support group or a group on which the team – and the project – depends.

## Executing the actions

- Form a super-team to oversee changes in the project
  - Needed if there are more than one or two changes





So those are some of the checklist items to consider in fixing a failing project. You should compile your own list from your experience and your observations. Now the question is how to implement the changes.

I recommend you recruit team members, possibly including a business sponsor, in forming what I call a "super-team" to lead the changes. This accomplishes two things. First, it puts responsibility for change on members of the team and the management who are already engaged in the project. Second, it allows you, whether you are manager or consultant, to step back and let the people who know the environment best to start working on managing the changes.

Of course, if there are very few changes to be made, you can just oversee them yourself. But often, fixing a failing project takes longer-term supervision and feedback. So make the super-team, if you create one, responsible for reporting to a manager on their progress on a regular schedule.

## Executing the actions

- Get commitment from line manager
- Prioritize actions for line management
- Select actions for the super-team to oversee
- Commit to a completion date for each action





There may also be changes that are needed in the management or in the way the managers run the project.

If there is a manager change to be made, do that first. Then, if there is a super-team, have the manager commit to supporting the super-team in making the listed changes. Some of the changes may require manager actions, too, so get buy-in from the manager to be sure that the manager will take responsibility for doing the actions.

Finally, list out the actions that the super-team is to take, and put a due date on each one. Get agreement and commitment, and off you go.

## Executing the actions

- Line manager should lead – take the first action
- Super-team takes responsibility for the most important items





If there is something the manager has to do to clear the way for other actions, the manager should "go first" – take on the changes and show support for the project by being prompt and direct.

Then the super-team can get going; and the manager should give feedback – hopefully positive – to the super-team as they complete actions.

## Executing the actions

- Super-team may assign others to do actions
- But super-team remains responsible for completion





You can allow the super-team to delegate things, but make it clear that they will be held responsible for completion of each item on their action list.

## Executing the actions

- Get mentoring/coaching for the super-team
- Set regular review dates
- Set criteria for success





As the actions are done, you should see improvement. If things seem to be going slowly in implementing the actions, consider getting a coach or a mentor for the super-team. Sometimes, like managers, super-teams need leadership that doesn't come naturally.

Review the progress, both in the action list and also in the project, regularly. Personally, I think that monthly reviews are not often enough, but you need to choose an interval that works for you. As with Agile iterations, I think that 2 weeks is a good interval, and 3 weeks is an acceptable interval.

Keep you eye on the criteria for success, and be sure to provide feedback to the team and the super-team when they show progress.

## Executing the actions

- Follow up checklist
  - Watch the communications
  - Track completions
  - Probe for resistance or deflection
  - Review against original plan





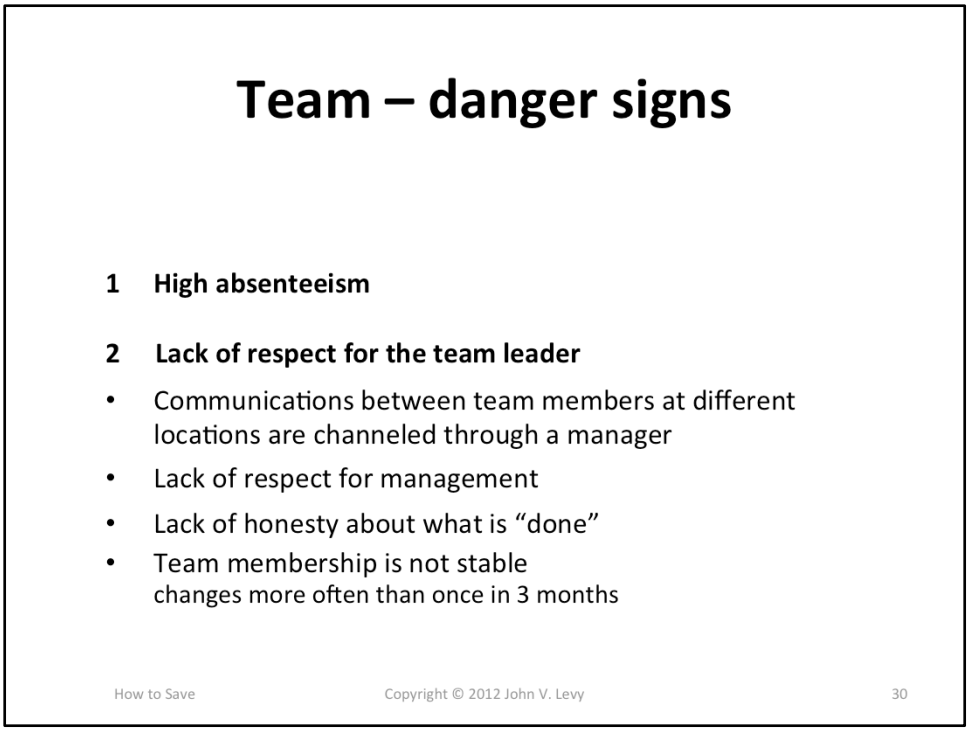
As with any process or project, you need to watch what goes on for a while to know how it's going. Pay attention to how the team and the super-team are communicating with each others and with managers. Listen for signs of resistance to the implementation, and if needed go back to observe or interview people to find out what's going on.

You started with a plan that went poorly. You assessed what was going on and made a new plan for actions and implementation. Now you need to keep reviewing how the project is going. Don't wait for a major indication of trouble to step back in. Regular reviews will keep you on track or let you adjust for smaller problems.


OK, I promised to review the danger signs from the previous webinar. If you've already heard these, you can stay with me and see if you have a new perspective on them, or you're welcome to go now. I'll post the slides and the text to my website by tomorrow.

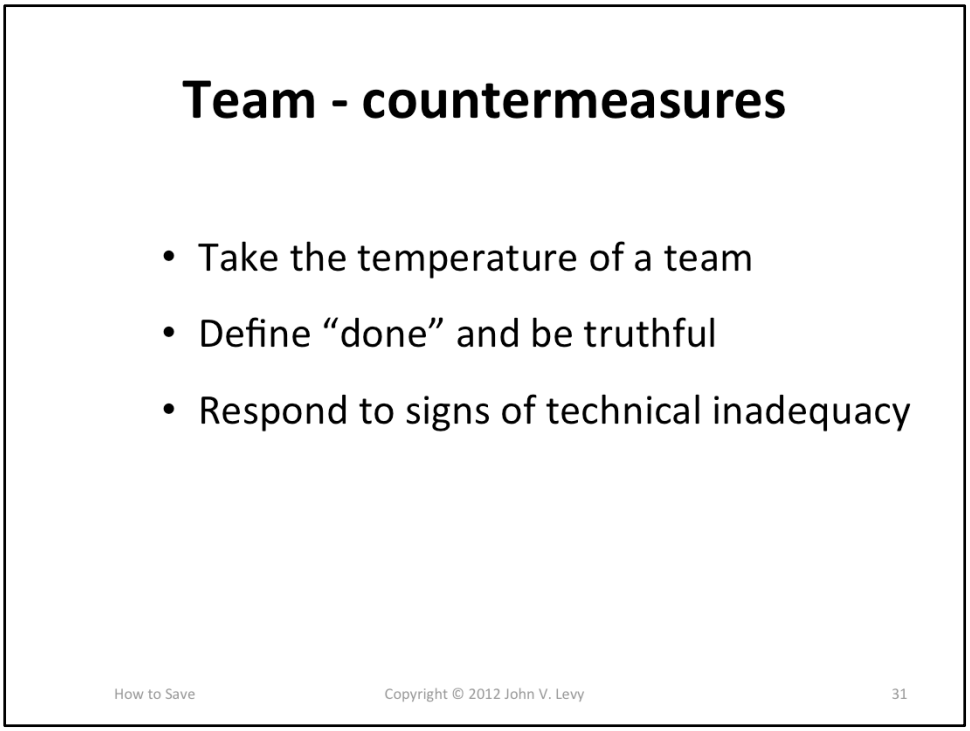
I've divided up the signs into 5 different categories. For each category I'll point out two KEY signs – signs that I think are the most significant within the category or domain.
Then I'll review the countermeasures for these signs – what you can do about them.

OK? Let's start on the signs for Teams.




**High absenteeism** is one of the key indicators. Usually it means that the team members find working in this team to be stressful and unfulfilling. There are hundreds of reasons for this kind of dissatisfaction, but you can be sure that when team members often fail to come to work, the situation is critical.

**Lack of respect for the team leader** is another key indicator. If the team doesn't respect the leader, the leader can't lead.




Some things you can do to help the dynamics of a team include the ones shown here. **Taking the temperature of the team** just means you spend time **listening** (yes, listening) to team members. Ask questions that are open-ended, like, "how do you feel about the project" and "how well do you think the team is working together?" **If the team is having trouble finishing up each increment of work**, you may have a problem with the definition of when the work is "done." In the Agile method projects, defining exactly what "done" means is a critical step. For example, "done" means that the code is written (if there is code), it has been reviewed by another team member, checked-in to the system, had its unit testing completed with no errors, has been compiled into a test system and run against the "smoke test" or basic functional testing. Anything less than that is NOT considered "done," and the item cannot be checked off until all of those steps are actually finished.

Once you establish a clear set of criteria for "done", then you can report on the real status of the project to your own boss (assuming you're the project manager). From then on, there should be no shading of the truth about what is done and what is not done.

Finally, **you may have some team members who are struggling** with the technology or with the processes of development. Ask team members (remember that you're listening to them now) about whether there are others who may need help. Usually, you can trust that team members know which of the others are doing well and which are not.

## Progress – danger signs

**3 Backlog of bug fixes grows linearly with time**

**4 Team cannot demonstrate a working prototype**

- Same bugs show up multiple times
- Regular small slippage of schedule
- Lines of code generated (KLOC) is used as the primary measure of progress

The first one relates to software bug fixes. When you're developing code, there are always bugs that need to be fixed. Usually, the team will put each bug description on a list, prioritize them, and then work on them from the list. The list is the backlog – bugs that haven't been fixed yet. If you ask "how many bugs are on the list?" this week, and then you ask the same question next week, you can compare the answers and see a trend. If week after week the number of bugs on the list keeps growing, your project is in trouble. The team keeps adding more code to the project before they fix the bugs in the previous code. Or else the fixes they are putting in are causing more bugs. Either way, the indication is that they are not catching up with the bugs. You have two choices: either stop all work on new code and focus everyone on the team on bug-fixing; or insist that code not be integrated in the system until after it has passed a set of tests to show that it is good enough. You may have to do both of these alternatives to get your project back on track. The second sign is one that happens when you ask for a demonstration of the partially-completed project. If the team can't put together a working prototype, this is a sign that there may be NO code that actually works correctly. In other words, a demonstration is the only proof that things are working. In an Agile project, the team demonstrates working code every 2 or 3 weeks (on a regular schedule that has been determined at the beginning). Any feature or function that is not demonstrated is considered NOT DONE. I highly recommend adopting an Agile approach to development if you've had this kind of problem with your project.

## Progress – countermeasures

- Make progress visible
- Deal with schedule slippage
- Stop the growth of work backlog





Here are a few of the countermeasure you can take to help make sure your teams make progress.

First of all, have a central place – preferably on the wall – where the team can see their own progress. Show the "DONE" pieces and the "being worked on" pieces clearly, who is working on them, and, if you have them, the time estimates that were made for each piece. A team that can see its own progress, step by step, tends to want to keep on finishing things.

If you have schedule slippages, you can do the things we talked about – stop other work while the team fixes all the bugs; and re-emphasize the need to have tests ready to evaluate each new part of the system as it is produced. Add automation to the testing tools to make this easier. Then, if necessary, re-estimate the entire project and get it reset with the management.

This will also stop the growing backlog. Then, of course, continue to monitor the backlog so it doesn't happen again.

## Processes – danger signs

**5 Testing/QA is the bottleneck for progress**

**6 System build / integration fails to complete more than 5% of the time**

- Bug fixing is done by a person other than the original author
- System integration (or system build) takes more than 4 hours

How to Save  34

**Testing or QA is a bottleneck for getting code released.** As with the system integration question in the previous slide, testing itself may have problems that cause delays. If your QA people are not in the same room with the developers/coders – or even not on the same continent – you may be waiting for code to be transmitted, received, integrated, tested, and results of tests communicated back. If this causes serious delays, you will see a backlog for testing. Of course you want thorough testing, but if the capacity of QA is too low, the backlog will delay the project, and will also tempt people to circumvent QA. This will cause other serious problems for the project.

Get good testing tools lined up, and automate the testing as much as possible. Then, if you still have a backlog, it's time to consider reallocating resources from coding to testing.

**System build doesn't always complete successfully.** Here is another red flag. If the build process breaks down and doesn't complete a significant percentage of the time, then you either have a broken build process, or you have inputs to the build that are not properly controlled. Find out which one it is, and then fix it, because a reliable, speedy system build should be a very high priority for your projects. Make sure you have the capacity to do AT LEAST a daily build; preferably an hourly build, or even CONTINUOUS build.

## Processes – countermeasures

- Have enough automation
- Shorten the feedback cycle from QA work





If Testing/QA is the bottleneck for progress, get better tools or reallocate resources to make QA NOT a bottleneck.

If System build / integration fails to complete more than 5% of the time, look at the build process and/or tools and fix them. If it's caused by badly-formatted code files or other inputs, get the team to fix the requirements and formats so it won't happen.

## Management – danger signs

**7 Team has worked more than 60 hours per week for 3 weeks in a row or more**

**8 Upper management (CIO, VP, Director) openly disparages the team**

- **Two schedules**, one for internal team use, the other for management reporting
- Team provided **an unrealistic schedule** in order to get the project approved
- **Two sets of requirements docs**
  or two sets of project plans
- **Responsible manager** (or Product Owner) **does not attend milestone review**
  or end-of-sprint demo & review





OVERWORKED TEAM. If the team is consistently spending more than 60 hours per week working, you have a high risk of burnout and also of getting LESS done than if you let the team work a reasonable – and SUSTAINABLE – schedule. Sure, there are times when you need a burst of activity to meet a deadline, but after a week or two at most, that sort of burst should be finished. The team should return to a regular schedule.

MANAGEMENT badmouthing the project – this is always a bad sign, because if the team hears that a manager has said things about the project that cast doubt on its value, they will again feel less motivated and less willing to give their best effort. Even worse, if you combine these two factors – overwork AND disparagement – you'll get the worst possible result that can only be overcome by an outstanding team motivating themselves to carry on. You don't want to rely on that for too long.

## Management – countermeasures

- Use sponsorship
- Investigate your reward systems
- Enough control? Or too much control?
- Undo team burnout





SPONSORSHIP is a key idea – have a senior executive who is the sponsor for the project. This executive attends reviews and follows the progress of the project with interest. The executive is someone who can break logjams and get resources for the team needs if necessary, and who will help negotiate relationships with other departments that the project may depend on.

REWARD SYSTEMS – are about what measures you're using to determine performance of the people on the project. Are you measuring their hours? Their lines of code? Their bug rate? What ever measures you use, ask yourself whether you really want or need that number to be high (or low). Make sure that someone who gets a high score on your metrics is really someone who is contributing to the project's success. If you find that the correlation is not so good, start looking for a different measure.

CONTROL – is a delicate thing. If you take too much control, you will make the team lose initiative. If you have too little control, then you may not get anything you want out of the team at the end.

BURNOUT – is what happens when the team is overworked. If you can, measure the useful output of the team, and notice when the output starts dropping off (bug rate, for example, may be a function of how many hours per week the team works). Also watch for other signs in the individuals. Encourage them to go home, to rest, to eat regular meals.

## Complexity – key signs

**9 The core team depends on more than two other teams for essential parts of the system**

**10 The intended product/release depends on a subsystem which is not yet completely defined**

- More than 150 people are involved in the project
- The core team has more than 12 members
- There are more than 4 primary (API or protocol) interfaces
- Requirements or success criteria keep changing





First, dependencies on other systems or subsystems. A subsystem can include a service like a database server or a communication link; or a tool, like a compiler or an XML parser. If there are three or more subsystems that your system depends on , you may spend all of your time chasing the latest changes in those subsystems. Consider, instead, implementing a simpler subsystem just for this project, if that will do the job.

And Second, when you depend on a subsystem that is not yet defined, you're asking for trouble. Your project may be held up indefinitely while you wait for the subsystem to complete its definition or implementation. You may be able to mock up a local version of the subsystem, but then you risk not being compatible with what is released later by the subsystem designers.

## Complexity – countermeasures

- When is there too little time?
- When are there too many interfaces?
- How changes add complexity





Complexity is one of those abstract ideas that can become very real when you're doing implementation on a project. Assessing the complexity may be part of your job as project manager or leader, so consider some of these ideas in making your plans.

First, extend your time estimates for integration and testing whenever the project involves more complexity. For every subsystem interface you have to deal with, you may add 5% or some other increment of testing time in your plans. If the overall project does not seem to have enough testing and integration time at the end, when all the pieces must come together, then you'll get pressed to eliminate some of the testing. This can lead to disaster, so start off with enough time estimated for integration at the end.

The same thing that applies for interfaces also applies for changing requirements, evolution of the system as it is deployed, and other additions to complexity. Whenever the scope of the project is changed significantly, be sure to renegotiate the timeline and schedule.

## Summary

Assessment

Action List

Super-team / manager + team leader

Monitor progress

Get help when you need it





Well, we've covered a lot, so I would like now to summarize where we've been.

If you have a failing project, you need to start with an assessment. Assessment can be done top-down or bottom-up. I recommend doing both. When you finish, you should have a good idea of the charter and the requirements for the project.

Next you need to choose actions to take to fix what's not working. I've offered you a list of possible actions as a suggested way to start your thinking about project success. Choose your own list, or get some outside help to construct an action plan.

Then you're ready for implementation of the action plan. Create a super-team if that seems to fit, or get the manager and the team leader to commit to implementing the needed changes. Monitor the progress regularly and you should be on the road to fixing your project.

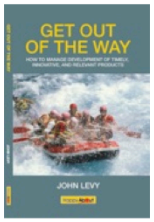
Finally, getting help when you need it is always a good idea. If you need a consultant who can help your project get through a difficult spot, feel free to contact me. If I'm not the right consultant for you or your project, I'll refer you to others who may be better matched to your needs.

# References

*White paper*: **9 Mistakes that Lead to IT Project Failure**

http://johnlevyconsulting.com

*Book:* **Get Out of the Way**




http://johnlevyconsulting.com/john-levy-book/

*Agile Management* **blog**

http://johnlevyconsulting.com/blog/





Here are some references you may want to take advantage of.

I'm offering a free white paper titled "9 Mistakes that lead to IT project failure". If you go to my website, you'll find the signup for this paper right on the home page. This paper is aimed at executives, but you may learn from it at any level of project management.

When you request the white paper, you'll also be subscribed to my every-2-weeks articles, which I call "Agile Management". You can of course unsubscribe at any time.

And if you're interested in my book on managing high-tech teams and people, have a look at "Get Out of the Way". It's primarily aimed at technologists who have become managers, but there are tips in there that you will enjoy reading related to project management, including a lot of stories about projects I've been involved in over the years.

## Next webinar

"**Why Agile Won't Fix All of your Problems**"

January 15, 2013 at 10:00 AM

http://johnlevyconsulting.com





I recommend you join me in January for another webinar titled "Why Agile Won't Fix All of your Problems." In this webinar, we'll review what Agile methods are good for (and I do believe in using Agile methods), and why many people who expect Agile to fix everything are disappointed.

I invite you to join me again in January. Just visit the web page on this slide, where you can click on the link to sign up.