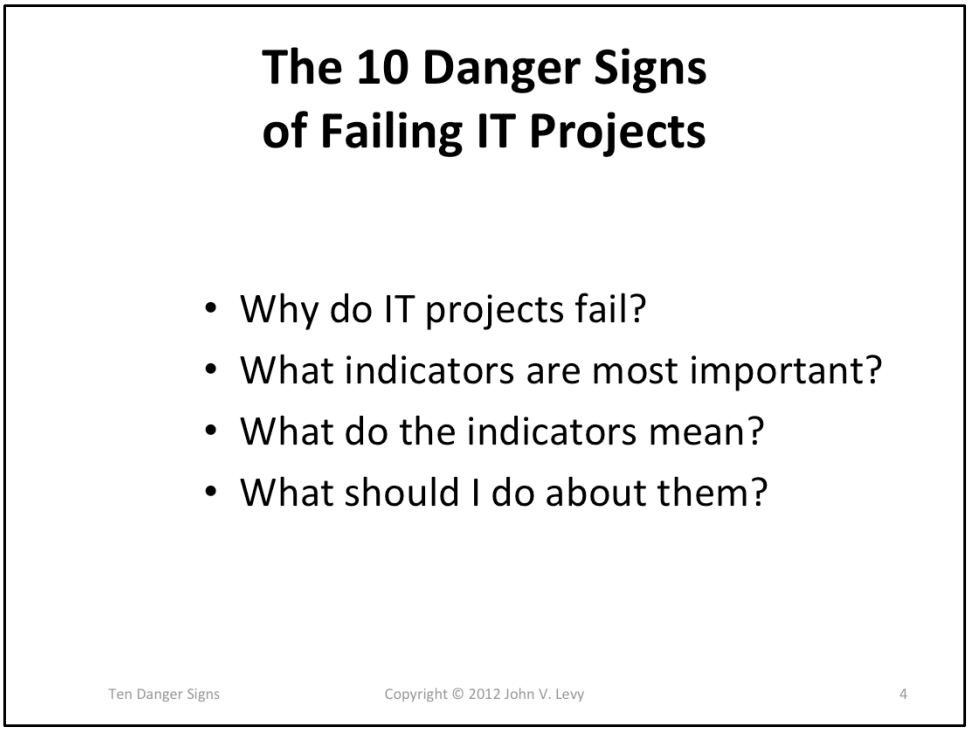
The 10 Danger Signs of Failing IT Projects

John Levy, Ph.D.

Welcome to the webinar on the 10 danger signs of failing IT projects. I'm John Levy and I'll be guiding you through this journey into how projects can go wrong and what you can do about it. I hope you're ready to respond to the questions I'll put out to you as we go along. That will make it easier for me to find out what you think and will also give you a chance to express yourself as we go along.

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

OK? Let's get started.



So let's move on to what you can expect from this webinar.

I assume that most of you are interested in this webinar because you're a sponsor or a manager who is responsible for delivering on an IT project.

A **sponsor** is an executive or senior manager who typically manages the portion of the business most affected by the project. For example, a VP of Marketing who is sponsoring a customer analytics project, or a VP of Sales who is sponsoring a CRM project.

A **manager** who is responsible for delivering the project may be (a) a project manager in the business or in IT, or (b) a line IT manager who owns the development/ delivery team.

What we'll be addressing today are 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.

Introduction

- Roles
  - Executive sponsor
  - Business sponsor ("Product Owner" in Scrum)
  - Program manager
  - Project manager
  - IT project manager
  - IT team leader
  - IT team member
  - IT team process coach

Let's review the typical roles in an IT project:

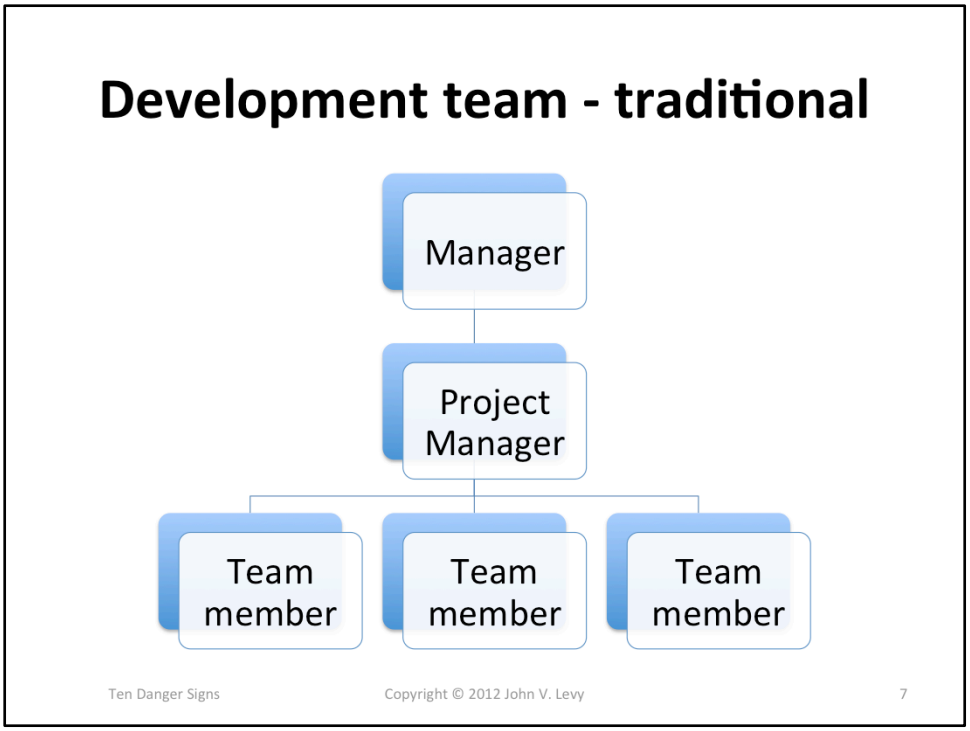
Development or integration **team members** who are doing the technical work, including development, QA/testing, integration;

Operations **team members** who are doing infrastructure management, integration, deployment and maintenance.

Of course, there may be DevOps teams who do a combination of the two.

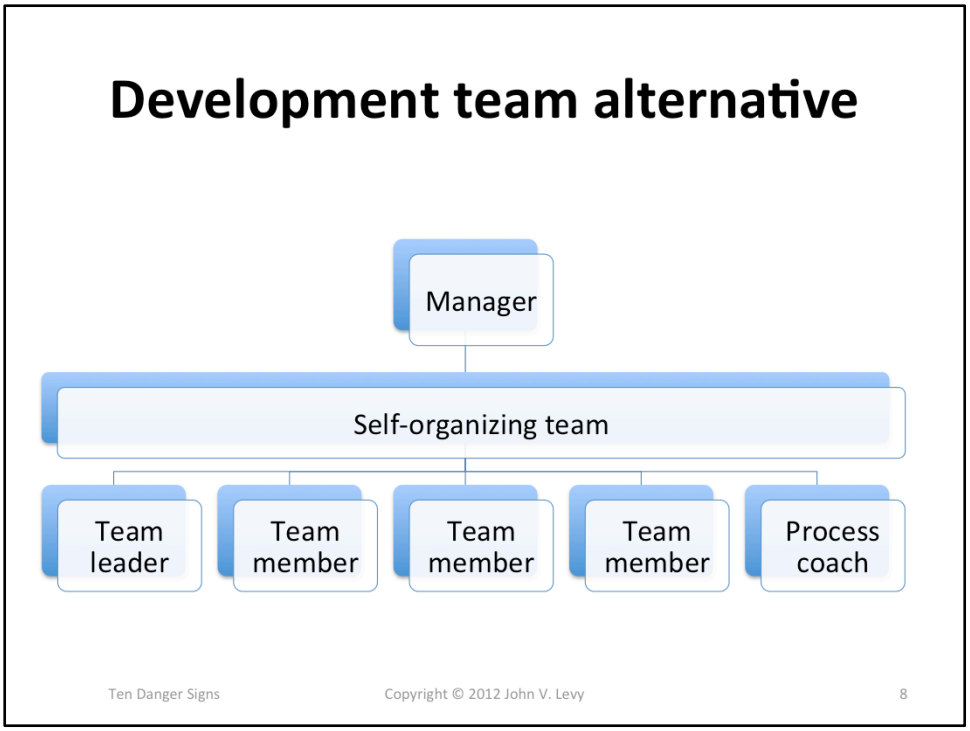
Project manager in IT

And then there are the various other managers and leaders who are listed here. And you may have a process coach – someone to help keep the development process on track as things go along.



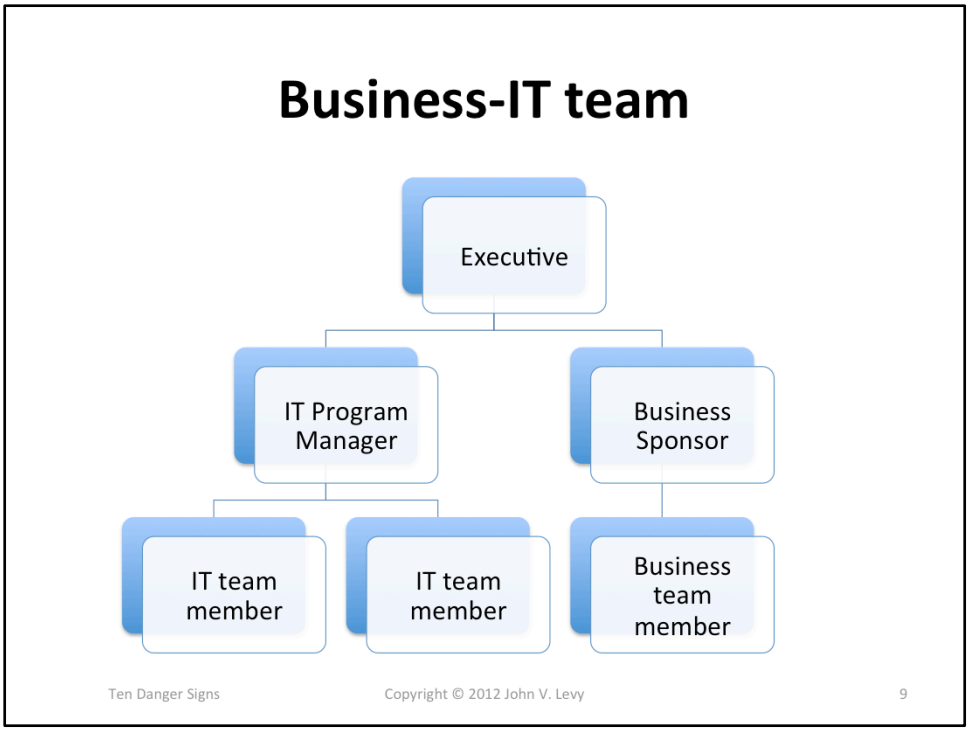
OK, now let's review how development teams are organized. I'm including this review just to get fixed in our minds the kinds of structures that are used in developing things, including software or IT capabilities in general.

First, here is the structure in a traditional project team where there is a single project manager and a team of developers. Above the project manager, typically, is a line manager. Alternatively, the project manager may have a more complex reporting relationship, such as having a line manager for the function the project is related to plus another manager for the internal customer of the project – the manager whose business will be affected by having the project completed.



Here is an example of the structure for an Agile project – In this case, the manager is over a team, and the team may not have a formal project manager. The key is "self-organizing team," one of the ideas that has come from the Agile software framework. A self-organizing team selects its own way of organization, including how to divide up the work, who to designate to represent the team for various situations when the team has to interact with others outside the team.

I won't go further into Agile teams now, but I recommend you have a look at Agile methods when you have a chance. Agile methods are having a positive impact on development projects.



Here is another way to organize a development team when the project is an IT development project. Here we see that there is a role called "Business Sponsor" – that's the person who is paying for the project, or at least is the designated representative of the Executive whose budget is paying for the project. Then there's a Business team member shown here, who may be a Business Analysts or other role who has good knowledge of what the business expects to get out of the project.

Then on the IT side, there is an IT Program or Project manager and a number of IT team members.

On the Business side, there may also be a Business Project manager. If there is, then the Business Project manager and the IT Program manager usually work closely together – if all things go well in the organization.

It's time now to get into the danger signs.

I've divided up the signs into 5 different categories. Here are the names of those categories. For each of these categories, I'll give you an initial list of signs that can crop up during a project and talk briefly about what each of them could mean for the project. Then I'll point out two KEY signs – signs that I think are the most significant within the category or domain. We'll talk about those two signs in terms of what they mean for the project.

Then later on in the webinar, we'll return to the key signs and talk about the countermeasures – what you can do about them.

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

Team functioning is key to successful projects. Yet team health is a subtle thing. You can't judge a team by its physical surroundings, nor by its style. Remember, there can be a wide variety of cultural norms for teams. But you can watch for indicators that a team is filled with conflict or is confused or is not sufficiently skilled to do the work required.

Here are four signs that indicate possible troubles in a project.

Communications channeled through a manager – not a good set-up, since it indicates a hierarchical management that does not know how to work with, trust and encourage teams. But it is not a project-killer. In some cultures, it can be made to work, but success relies on the manager in the chain being very quick and responsive.

Lack of respect for management – often occurs in high-tech teams, if only because management usually is much less skilled in technology. In some teams, lampooning upper management is a sport openly engaged in, and it can serve to bind the team together against a common "enemy", namely the management. But if the management is truly "clueless," then the disrespect can indicate deeper problems in the communications between management and tech team. Watch for the difference between team members making fun of the *style* of the management vs. team members actually displaying anger at management's lack of concern, understanding,

Here are two more team-related indicators. These are the ones I believe are most significant for a project.

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.

Team dynamics - countermeasures

- Take the temperature of a team
- Define "done" and be truthful
- Respond to signs of technical inadequacy

Some other 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.

Progress signs

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

OK, let's move on to signs that relate to "Progress". On projects that are large or complex, it is often hard to tell how things are going until you get near the end of the project. So to get a feel for whether things are going well, you may need to dive down into some of the details.

For example, the first sign shown here is that certain bugs or problems reported during testing have shown up more than once. When the same problem is reported repeatedly, it means either that a "fix" for the problem did not in fact fix it, or it means that it popped up again after some other changes were made in the system. In either case, it's not a good sign, because fixes should work; and problems that recur are often signs that the design is not good enough.

If you receive a schedule from your team, and the team keeps asking for small changes in the schedule (in the only direction that they ever ask for – later), this is an indicator that they are not estimating their work very well. Or else the work is being complicated by other factors. In either case, the project is going to take a big hit in schedule by the time they address the whole project. So take these small slippages as a sign that the entire time-scale is off.

The third sign here – you are using "amount of code generated" as a measure of progress – is just a warning. Once you set a measure such as KLOC (thousands of

Progress – key signs

3 Backlog of bug fixes grows linearly with time

4 Team cannot demonstrate a working prototype

Here are two more "progress"-related signs that I think are more significant.

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

Progress, or lack of it – 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 signs

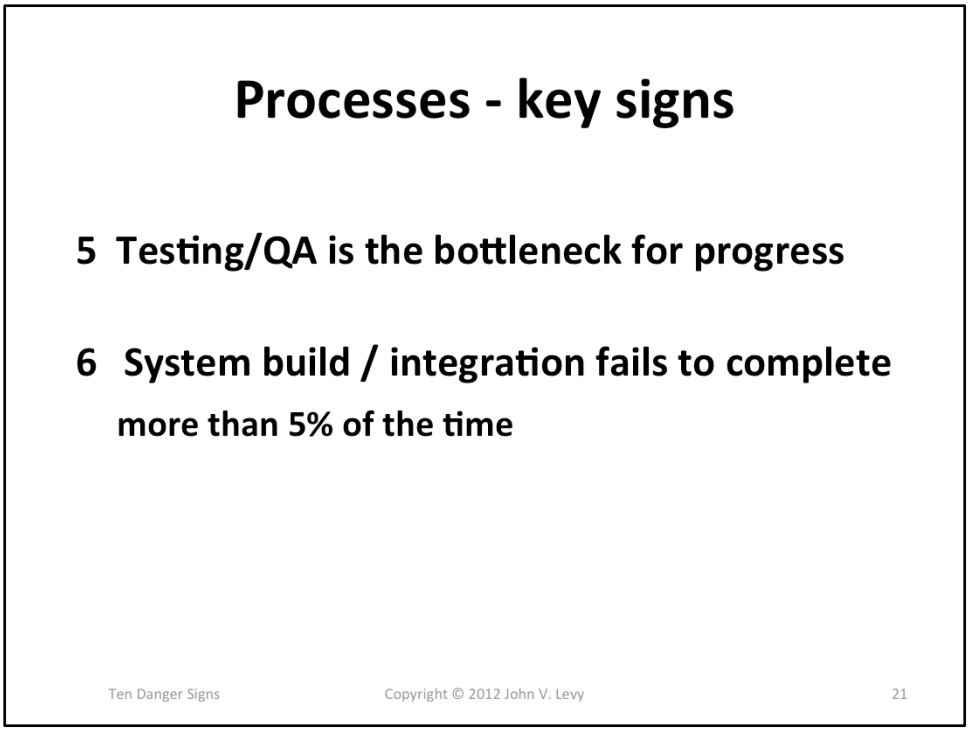
- Bug fixing is done by a person other than the original author
- System integration (or system build) takes more than 4 hours

OK, let's move on to Processes as a new category. Processes refers to how the process of development and integration is being carried out by the development team.

Here are two signs that things may not be going well.

First, when a bug is found in testing, the person asked to fix it is NOT the same person as the one who wrote the code (or implemented the feature, however it is done). This is a symptom of a misguided attempt to have people who are not software engineers or "designers" work on bug-fixing. Often this is done because there are too many bugs to deal with. And if the designers were asked to fix their own bugs, they would not have time to create new code.

This is not a good situation! Designer and software engineers should always be responsible for correcting their own errors. If they are not kept "honest" by seeing their bugs and fixing them, they will not learn to avoid the situations that led to the bugs in the first place. So I strongly recommend that you stop this practice right away. If the designers cannot produce enough good code to make significant progress on the project, then they should be replaced by others who can. In addition, you may need to invest in better testing tools so that the designers are not wasting a lot of time waiting for results of tests. We'll come back to the idea of



Continuing with the Processes theme, here are two significant indicators of trouble:

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

Processes & tools – countermeasures

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

OK, we've discussed the countermeasures in the previous slide. Here's a recap:

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 signs

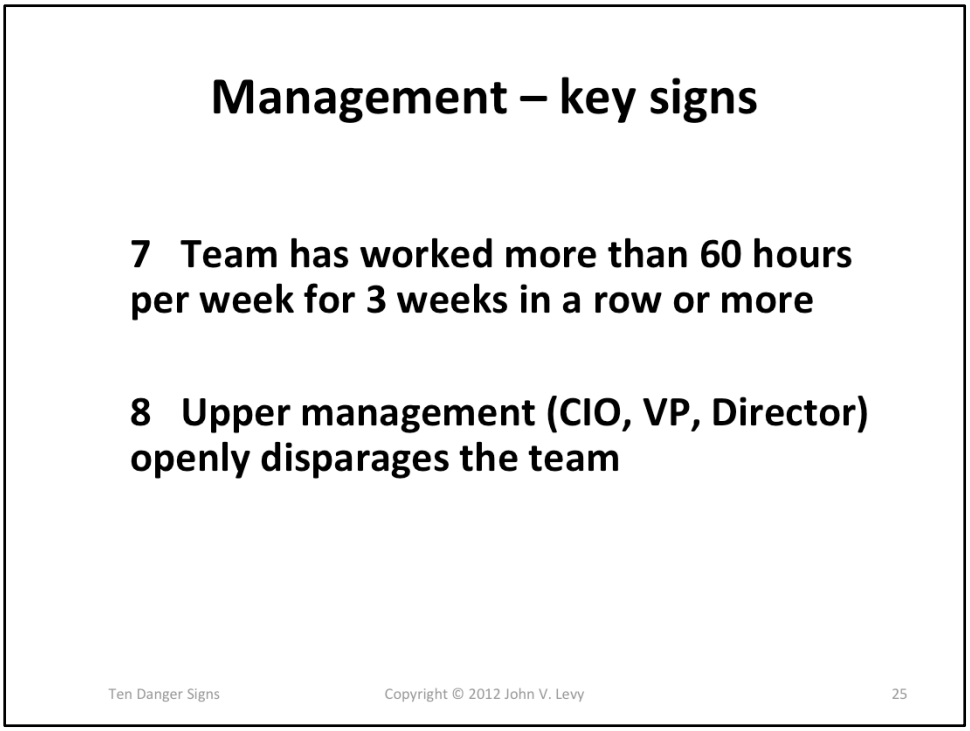
- **Two schedules**, one for internal team use, the other for management reporting
- Team provided **an unrealistic schedule** in order to get the project approved
- **Two sets of requirements docs**
  or two sets of project plans
- **Responsible manager** (or Product Owner) **does not attend milestone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

Now we move on to the fourth area – Management.

Management of course covers a lot of levels. Here I am focused on the interface between the project managers and upper management, and also on the team to manager interface.

Any time there are two schedules, or two sets of requirements, you can bet that the project will get in trouble. People create a separate schedule for management when they don't want to report the REAL schedule upward. Why would they do this? Usually because the management has imposed the schedule and doesn't want to hear anything about it not being possible to meet. Another way it sometimes happens is the team underestimates the schedule (possibly knowing that it's unrealistic, but usually not) and later finds out that it can't meet it.

I had an experience years ago where the project I was leading gave a very realistic schedule to management and started on the project (which was to take about 2 ½ years). About a year into the project, another team bid to take over the project by giving a much more aggressive schedule. Management went for it, and my project was cancelled. Of course, the team that took over actually delivered on the date that my team had promised, not on the one they had given.



Here are two key indicators in the management area.

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

Here are a couple of approaches to helping management encourage projects to succeed.

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.

Complexity signs

- 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

OK, now let's move on to indicators related to Complexity.

Complexity can overwhelm a project. Here are some of the indicators that a project may be too complex.

TOO MANY PEOPLE – over 150 people, the "team" can't all know each other. Even with 20 people, it's not a single "team", but almost certainly multiple overlapping teams.

CORE TEAM – people who have to interact daily – should be less than 12, and preferably no more than 8, because communications among 6 to 8 people is very significantly better than with larger teams.

NUMBER OF INTERFACES is a technical criterion – and I think that when you have 5 or more interfaces to deal with, the project may be threatened with complexity. The interfaces may be programmatic interfaces (such as APIs), or they may be hardware or network interfaces, such as TCP/IP or SATA or FC. Keep the number of interfaces a few as possible. And particularly avoid having more than one or two NEW interfaces to deal with.

CHANGING REQUIREMENTS – this is the bane of many projects. While you want to

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

Here are two key indicators of complexity.

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.

I promised to give you some ideas for how to prepare for future projects. Here are a few of those ideas.

TRAINING is one way to prepare yourself and your team for the future. Whether it's Agile process training, technology training, or management & team training, you can have a team that is better-prepared by investing the time and money to get training.

PRACTICE is the way to becoming a master of your trade. Consider some ways to help team members and managers to get better at what they do – You can pair them up, even though they may be at the same level of experience, and let them learn from each other. This is one of the secrets of the value of pair programming, by the way. Seeing how someone else solves a problem can be very educational. And when a person working on a problem gets feedback from an observer, they may also learn something.

For managers, having a mentor can help, too. Create a mentoring program or ask your management to create one.

RETROSPECTIVES, which are a key part of the Agile program, help the team see what they've learned, what they can improve, and what they've done well. Encourage retrospectives for your teams, and make sure that they are not filled with criticism

Reduce risks

- Contingency plans
- Allow some duplication
- Have excess capacity
- Align teams – internally
- Align teams – externally (across functions)
- Establish a culture of trust & truth
  - Reward honesty
  - Ask for proposed solutions with problem reporting
  - Do not punish failure (in the small scale)

Reducing risk is another theme in project management. Here are some ideas….

CONTINGENCY PLANS apply not only to the project team itself, but also to broader contexts, such as the building you work in, the infrastructure you use to do your work, the membership of the team, and so on. Make contingency plans in advance for all of the items you can think of. For example, what would you do if your most senior developer quit or got sick in the midst of the project? Think about your backup plans for all of these items.

Duplication and excess capacity are actually needed to make work flow smoothly. This has been proven in manufacturing systems, and it is true in development as well (see the Reinertsen reference at the end). Plan to have a little more capacity for work than you think you need. The result can be lower risk (when someone is not available unexpectedly) and also better flow (when someone can substitute at a bottleneck in the process).

ALIGNMENT means making sure we're on the same page – have the same understanding of what is needed for the project. You should be spending a significant amount of your time verifying that external teams share the same vision of the result that you are planning on. And also repeatedly reinforcing the team's goals and end-results.

Summary

- IT projects are challenging
- Know where the challenges will come from
  - Management, Team dynamics, Complexity
  - Processes & Tools, Progress & goals, rewards
- Plan for contingencies
- Pay attention to culture
- Get help when you need it

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

We all know that IT and development projects are challenging. We've been exploring 5 major areas in which there can be problems and trouble indicators. The more you can classify the issues you've seen and remember them, the more you will be prepared for the next project and its problems.

Plan ahead for contingencies, as we've suggested in the last two areas. Give yourself time to think about what COULD happen in your project.

CULTURE is another area that warrants attention. When your team is spread over continents and/or countries, there will always be cultural differences. And even right here in the U.S., where teams are often composed of people from many lands, culture is a factor. Keeping respect for all cultures, try to understand how culture affects communications, interaction style, and ways of working. The more you can get the team to acknowledge and accept each other's cultural differences – and motivational sameness – the better the team will work. By motivational sameness, I mean that we all want to succeed, we want our projects to succeed and we enjoy having our work valued.

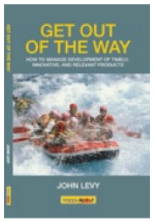
Finally, getting help when you need it is always a good idea. For example, there may

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

"**How to Save a Failing IT Project**"

November 13, 2012 – 10 AM

http://johnlevyconsulting.com

Now that you have heard about the danger signs, you'll want to implement actions that will keep your projects on track. To see how to do this, I recommend you join me in two weeks for another webinar titled "How to Save a Failing IT Project." In this webinar, we'll look more closely at the specific actions to correct the problems in your projects.

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

Additional Resources

Fred Brooks – *The Mythical Man-Month*

Gerald Weinberg – *Perfect Software*

Don Reinertsen – *Product Development Flow*

Here are three of my favorite resources in the form of books.

Brooks' book is a classic on managing software development. His experience includes the major operating system "System/360 OS/360" for IBM.

Weinberg has written many books on managing software and on consulting. This one covers the ideas that managers should understand about software so they know why you CAN'T have "perfect software" every time.

Reinertsen is a consultant in California who has combined ideas from manufacturing and from economics to help us understand how product development works. See this book for ideas on "excess capacity" being needed for flow.