ADVANCED SYSTEMS ANALYSIS & DESIGN
Professor: Alan Hevner, Ph.D.
Guest Lecturer: Robert J. Schaaf

Part 1: Requirements

This presentation about software project management, actually, compare that with-- I present a class in project management, and typically that's a four or five day class, 8:00 to 5:00, with exercises. Tonight, of course, we don't have enough time for exercises. And in this class I cover about 30 subjects spread over five days.

Tonight I'm going to talk to you about three subjects. One is project management itself, specifically the idea of the broadness of the project management. I come to that later on, and I explain what broadness is here.

But I will also talk about two tools that absolutely determine the flavor of project management that you get from me. The first one is requirements and requirements engineering, which I look at as a project management tool. It's more than that, but allow me to talk about it as in project management.

And the other is something that I will really explain, and that is process assessment. It's also a project management tool. And more specifically, it's a risk management tool.

And these are the three subjects that I'm going to cover tonight instead of the 30 that you get in a full-time, one week course. Unless I specifically exclude it, all of this applies to both the development and acquisition and the profession of software as a service. So do not assume that I am just talking about software development here. When I talk about software, I specifically mean software engineering to include development and acquisition and the profession of a service.

OK. Let's start with the first section, requirements. At the same time that belongs to the subject of requirements, I'm going to talk about stakeholders and stakeholder needs. And I'm going to talk about quality. And I will define those and I will illustrate that.

But before I actually get into the technical thing, I need to give you a warning and heads up here because what you hear in this section is not generally accepted. Now, there are many people that do accept it, but there are other people that don't. And that's what I am trying to put forward here. And that is in the profession, the requirements practice remains controversial.

If you have a computer now or later, you can Google, for example, "no stinking requirements," and you get a number of hits from software people that are still of the opinion that we don't need requirements in order to get good software. I say you need requirements to get good software, and doing requirements engineering determines the form of project management that you can apply.

So what are the objections? I mean, these people have their reasons, of course. They say, we don't have time for requirements. And requirements do change, so why bother?

Some other people say, oh, but we have alternatives. Instead of requirements, we come up with a project charter, a kind of a document that outlines what the project is all about. And I've done that myself. I've dealt with managers that said, we don't have time for requirements. Why don't you do a project charter?

And you learn, of course, along the way, and now I know better. But I've done project charters. And you write things down, what this project is all about.

And then the document gets filed, and nobody ever looks at it. And if somebody would look at it, it would not answer their question, why they looked at it. So indeed, that is wasted time and effort, to do a project charter.

So specifically as a project manager and as an advisor to project managers, I still have to fight in many cases for doing the requirements, overcoming the objections of we don't have time. There are alternatives.

And then my arguments, of course, are the requirements will answer, why this project? They set the scope of the project. And what I have to do then is admit up front that requirements will change.

Yes, they will change. It's taken into account that they will change. This is not the idea that you start with requirements up front, and then you move like the waterfall down to design and implementation and test. So I admit that up front, that requirements do change.

But then I also point out that they are signposts and yardsticks in the course of the project. And they are critical in judging the readiness of the software at the end of your project.

But overall, whether you do requirements or not do requirements determines the kind of project management that you can apply. And what you see in the following slides is all based on the fact that, yes, there will be requirements.

OK. I should also point out that what you see here, we did not learn in class. We did not learn from books. We learned this in practice along the way. And I made mistakes, and hopefully learned enough from that to do better the next time around. But I could not point to something in the literature or in class where we picked this up originally. Not at all.

Let me see. I promised to talk, in addition to requirements, about three other concepts as well. Let's start with the idea of stakeholders. The idea of stakeholders counters the idea that all we need to be concerned about is the customer or the users. And that's a very important point where we still stand out with general practice as well.

So what are these types of stakeholders? A stakeholder is a person with an interest in anything related to the software, which can be the software itself, and then any property of the software. Originally, software engineering started out with an idea of interest in the function of the software. But I say here, not only the function but any property or attribute of the software, whether it is the throughput of the software, the quality of the software, et cetera, not just the function of the system.

And the kind of stakeholders that we talk about there are, sure enough, the customers there, the owner, the users, but also, say for example, the legal department. Nearly from the beginning in software project management, you have to deal with the legal department, as an example, because before you know, you may be using somebody else's intellectual property. And you may be sued if that somebody finds out and takes you to court.

So the interaction with the legal department is very strong. And the legal department is a stakeholder often, nearly always, in my projects. These are people with an interest in the software itself.

OK. The next one is not only in the software itself, but there may be stakeholders with an interest in the environment in which is software. This software may be part of a larger system. And the stakeholders, their interest is not in the software itself, but maybe in the things around it. For example, I use here the designer of a business application process. He's not so much interested in the software as in finding out what the software is going to be so that he or she can do the business, design the business application.

And finally-- that took me some time to find this-- the stakeholders also include the people that may not be interested in the software or even the environment in the software, but are interested in the processes that you use. I know it was eye opening for me that along one of the projects, the internal audit people-- it's a typical department in a large corporation that makes sure that, financially, anything that goes on inside the corporation is above board.

And they came along and said, we are a stakeholder in your project. And I objected. I said, but you're not going to tell me how I do my work, are you? And they said, yes, we are going to tell you how you do at least on the following points because we want you to work in such a way that you can be auditing, that we can know for sure that what you tell is going to be in the software indeed will be in the software.

So these people, they don't have an interest in the software. They don't have an interest in the environment in the software. But they are interested in the processes that we'll use for software engineering.

OK. Here I have to tell you an IBM story. Somewhat in the beginning I was a project manager, a small team. It was not a very big team. But I was asked to take a piece of software, a program, that a

salesman had written for his customer, a large American insurance company. It was a very handy program. Also, the customer liked it.

And IBM said, gee, probably other customers will like this program as well, this app. Rob Schaaf and his team are going to generalize this for a worldwide market, not just for this particular insurance company. But we are going to generalize it for everybody.

And sure, I started the job. And soon I found out that this particular project, relatively small as it was, had more than 160 stakeholders, different organizations, different groups, different people, but always some stake in this not-too-big application. Now, there was the legal department in it. There was the pricing people in it. There was the world in it.

Also in the case of IBM, IBM was at that time very much split between the American part and the rest of the world, so every function that you had on the American side, you have on the world trade side as well. So that was automatically doubling of the number of stakeholders. But I was surprised to find more than 160 stakeholders for this relatively small program.

On the other hand-- and I will mention it later on as well-- there's a relation between the number of stakeholders that you take into account and the quality that you can achieve. And with these 160 stakeholders, we ultimately produced a piece of program, an app, that you find back in every piece of IBM software these days. As well, it has been duplicated, imitated, in many other pieces of software itself outside IBM, all of this because attention was paid to a multitude of inputs on what this general piece of software had to look for, where initially it was nothing else than an app program by a single salesman for a single customer.

I'm going to distinguish-- and this is an essential point. Again, we kind of stand out compared with other stories that you may hear from other people. But we make a very sharp distinction between stakeholder needs on the one hand and software requirements on the other hand. And I'm going to explain the difference.

But let's start with stakeholder needs. Stakeholder needs is a condition that must or should be fulfilled according to one or more stakeholders, with the strength of the need varying between one need and another, with needs hopefully stated quantitatively, not good and bad, high or low, hot or cold, large or small, but in numbers, also with a clear owner of that need. If I have questions about that need, I know where to go and get more information.

But also one of things that you find out if you do these things is that stakeholders are not too god in coming up with their needs. I mean, you have a session and you say what are your needs? And the first thing, a look of, what do you mean? So you have to-- elicit is the word that we like to use in that-- draw it out of them. And there are several ways of doing that. I'll talk about that later on as well.

So that's a stakeholder need. And the fulfillment off that need may depend upon the software itself and/or the environment of the software and/or the process of the software. For example, with the process these days, there's a lot of interest in cyber security. Part of cyber security has to do with effect of the processes that you use. And so in order to get the proper cyber security, the processes are involved as well in the fulfillment of that cyber security need.

All of this suggests-- and I'll show you on the next page-- that there is an infinite number of types of categories of stakeholder needs. Every new project that comes along, there's a possibility that you run into a type of need that you've never seen before. And originally, there was only one need. There was functionality.

But then all these others came along. And I've just tried to remember a couple of projects, and I've written down what I could remember. But I like I said, a new project comes along, and you may have a new need.

Let me give you an example. It has to do with cyber security as well. For very good reasons governments ask certain pieces of your software to be breakable. I don't know whether I listed it here as a need, but systems must be secure but not too secure. If there are, for example, lawful reasons-- for example, criminal reasons-- to go into records, it must be possible.

So I'd never encountered myself that particular requirement. But like I said, every new project, there's a possibility that you run into a type of stakeholder need that you have not seen before.

And I'm not suggesting, not at all, that every project has all these kind of stakeholder needs. Typically, there are four, five, six, seven, if it's a large project, tens of stakeholder needs that really do matter. But altogether as a universe in software engineering, these are the type of stakeholder needs that you may encounter. It all depends very much on the field in which you are working.

OK. I will now use a couple of diagrams to explain the difference between stakeholder needs and software requirements. So let me start with this simple diagram. It's a glop of software. I am not suggesting nice, rounded software but a glop of software in an environment. It's a kind of an abstraction, but all I try to do is try to explain the relationship between stakeholder needs and software requirements.

So here, this is my starting point. The next one-- in this diagram, I am now showing you what stakeholder needs are. And stakeholder needs, as a matter of diagram, are all over the place. In fact, you see the piece of software is toned down. I mean in fact, stakeholder needs should not be concerned at all about the borderline between the software and the rest of the world. They may be and they should be all over the place if things are normal.

But I say there they are irrespective of the software project boundary, and they are problem-oriented or goal-oriented. But they certainly are not solution-oriented. I will come to that with software requirements.

Because if you keep this in mind and look at the next picture, you can start to see what I am driving at. And that is, the software requirement in that same kind of picture looks like this. This is what a software requirement looks like. And they delineate the subject software in words. Software requirements are a condition on the software or on the software process to make software acceptable to the stakeholders, acceptable to the stakeholders. That's what a software requirement is.

Now, there are many similarities between stakeholder needs and software requirements. And that's why it's easy to put them all together and treat them as the same. They are not.

But things that are the same between needs and requirements is that with requirements as well, the strengths may vary. Some of the requirements are "must." Others are "should." Others are "may." Not all requirements are equal. But the same is true for needs as well.

But requirements should also have a permissible cost of data. I mean, if it's a strong requirement and permissible and the cost to engineer this piece of software, this requirement is low, it's more than likely that you will pick up that requirement and include it in your project. On the other hand, if the strengths of a requirement is so-so and the cost to engineer this requirement is very high, it's more than likely that you will drop that along the way because you need to complete your project, presumably, in a certain amount of time.

Also-- and that's another point-- state your requirements in the positive. For example, it's easy to say, there should be no security holes. But it's very difficult to prove there are no security holes. Tell me what you do to close off the security holes, and then I can verify whether that requirement has been fulfilled. So yes, one way of coming up with requirements is requirements in the negative, but they are very hard to prove. And I try to avoid them all together.

OK. With that said, like needs, requirements may change, and they do change.

Talk about requirement types like I talked about the stakeholder needs. Similar to the needs-- they address everything of interest to the stakeholders-- requirements must address everything that may influence the acceptability of the software at the end of the project. And here you have-- I could have come up with a similar length of list for requirements, but various examples of types of requirements.

I include very much software processes and schedule. Schedule is one of those requirements that people typically try to put on the side. No, they say, that's not a requirement. But then you say, can I come up with this project anytime you like? No, no, we need to be finished by then. That is a requirement that needs to be-- and probably will be-- traded off against function or some other quality.

All these things are requirements. What you see here is a very broad approach to the idea of software requirements.

Next I will put them next to each other on the next slide. Like I said, I spent a good amount of attention to the distinction between stakeholder needs and software requirements because it's so essential to the approach of software project management that I will talk about.

Stakeholder needs are the responsibility of the stakeholders. On the requirements side, it's the software engineers that are responsible for it. Left is what the problem is, right, what the solution should be. On the left, we do it by, for example, modeling of the domain, the business processes, the workflows, domain scenarios, and mostly in the terms of feedback.

Same as needs, it's very difficult to get-- what I already mentioned-- stakeholder needs all of the stakeholder, so it's primarily through feedback, probing, that you get an idea of what the stakeholder needs are. On the right side, you have the modeling of the software in the form of use cases. That probably was mentioned in class as well.

My own favorite is defining software requirements in the form of test cases. I'm very much-- my most successful projects and my easiest projects are where I was able to convince or where I was able to consult with project managers that do test-driven development. Do you talk about test-driven development in class?

Mention it.

Mention it. I love it, if it's possible, because here you sit, as a software engineer, as a project manager, you put some test cases together, and you start talking with the stakeholders. You do a review or you do an inspection, and you start to talk. And before you know, you not only talk about this test case is right or wrong, it's a good test case or no, we don't need that test case.

Very soon you see the stakeholder scratching their heads and asking themselves, what is this system supposed to do? And only because you put in front of a test case where they say, no, that's not a proper test case. But before too long, they will ask themselves, what is the requirement?

And then finally for both of them-- and you will see that coming back as well-- you see ideas of agile engineering here. Stakeholder needs and software requirements are things that play throughout your project. There's not the assumption that you can do a kind of a waterfall model and nicely do implementation and testing on the basis of very hard stakeholder needs and software requirements. They will strengthen along the way, but they certainly are not there in the beginning.

After requirements and stakeholder needs, I have one more subject that I promised to talk about under the heading of requirements, and that is quality and consensus and quality. We are looking for stakeholder consensus. And again, this is an element where we kind of stand out. There are several other ways of skinning the cat. I don't think they aren't nearly as good as what you see here.

This is what I've done. And that is that for stakeholder consensus, I stay away from stakeholder needs. I do not try to convince one-- typically, stakeholder needs do conflict with each other. Not all of them with all of them, but some conflict with some others.

And the typical process there is with you as project manager to talk with the stakeholders and try to split the baby or something like that and come up with a need that they both can agree on. I have found that is a waste of time. I've tried it. I've done it, but unsuccessfully.

What I have found and what I do ever since is that I get the discussion away from stakeholder needs, and I get them to agree, hopefully, on software requirements. Use requirements as a base on which to come to a consensus. So I am happily move along in a project with conflicting stakeholder needs as long as we can get consensus and negotiate consensus about a set of software requirements, a priori on what is to be engineered and a posteriori after the fact that the actual software indeed is acceptable.

OK, more. That's also a very essential point and a difficult point to act on. But the transformation from needs to requirements is non-deterministic. What I mean with that is that if you have a certain need, it does not follow that that leads to certain requirements. You have a choice.

It's non-deterministic, which is very important because in many cases, you end up with-- you seem to end up with a set of requirements that either you cannot get everybody to agree on or are infeasible. The project is too large. And what you then have to do is you have to keep in mind, OK, if these were the needs, even if they were conflicting among themselves, maybe I can come up with some other requirements that avoid the problem of a too-lengthy project or a standoff between on set of stakeholders and another set of stakeholders.

I mean, I can tell myself that the transformation from needs to requirements is non-deterministic, but to act on that in the middle of a project is very, very difficult. It's one of the most difficult things because here you are set on a certain path. You have it in your mind. You have it all figured out. And then you run into barriers that seem impossible to overcome.

And then somehow you need the agility, the flexibility, to go back and say, OK, but my transformation from needs to requirements was non-deterministic. Maybe I can do a different kind of transformation and end up with a whole other set of requirements that avoid the barrier in which I seem to run. Easier said than done.

And I have to remind myself in each situation as it arises, that keep in mind that you can go back to the needs and try another transformation into different requirements that avoid your problem. It's a great opportunity, hard, very hard to realize, but very powerful.

OK. With that said, there's more. A requirement must be formulated in a way that makes it possible for to test for conforms in the course of the project. It's difficult to come up with an example, but requirements could be stated in something that can only be verified after, say, a year in use.

For example, reliability is a typical example, where a certain amount of uptime is guaranteed. One thing, it's a requirement that I try to avoid and try to replace with requirements that I can verify at least by the end of the project, not after a full year of operation.

So again, it's a thing that I have to keep myself aware of, that you have to avoid any requirement that is impossible to verify before the end of your project.

And then finally, a working definition-- I emphasize "working" because I'm going to do a little bit more about that in a moment-- is that quality is the degree that the software, at the end of the project, meets requirements. That's a working definition, but from everything that I've told you so far, that probably seems to you like a reasonable definition of quality. And in fact, it's a definition of quality that is used by many in the field.

I learned many things from IBM, but one thing I learned from IBM was that this is not enough. And some of you in previous sessions like this raised, in fact, two questions that I am going to talk about in a moment because they put question marks behind this statement of "requirements define quality." I've already put wiggly lines around that like a warning that more to come.

OK. Yeah? Let's talk about the question. And they were excellent questions. The first question that I got in one of the sessions was, what if there's disagreement on the requirements? OK?

And the answer to that is-- my answer, what I have done and what the answer in class was-- the project manager negotiates. To a large degree, a software engineer is negotiating, and particular the project manager. And he negotiates in terms of requirements rather than stakeholder needs.

He has to be creative, or she has to be creative. You have to come up with different views. Changing the language helps very much in the negotiation. And I wish that more universities would teach as part of the software engineering class negotiating skills because negotiating is a big part of the job.

OK. And he or she can negotiate. For example, my typical approach is, OK, here's a current iteration of the software requirements. What's wrong with it? Tell me what's wrong with it. Give me your words, OK?

And then we start talking. And I can see maybe an opening where I can reformulate a requirement or change it all together. And that's what I mean with "rinse and repeat." I formulate something, and then you start over again. Do you agree with this? Et cetera, et cetera.

At the end of the day, what matters here is the profit and the mission of the organization.

But then there's always the possibility that there's a persistent disagreement. I will mention the case here of IBM. It's long ago. I can talk about this openly.

IBM at one point had a project underway, Future Systems. And it was a big project, more than 5,000 people, engineers, involved in the project. it was so big that in fact they drew-- which was really not allowed at the time-- engineers from Europe to the United States to help because more engineers were needed. I was one of the people that was drawn out of Europe, and that's how I ended up in the United States, with IBM's Future System.

And basically Future System was the kind of system to end all systems. It was defined-- IBM had come up with things like IBM 360, IBM 370. And this was the next generation, and there never will be a need for any more systems than that. OK?

And the technology was there, it seemed, but there was persistent disagreement. And at the end of the day, IBM was not able to resolve that disagreement. The stakeholders that were opposing this approach of Future System didn't articulate their reasons very well, but later on, you could reconstruct it.

It basically was a standoff between the technologies coming up with a system, incredible functionality, incredible other qualities, and here were the sales people. And there was a situation where there were host applications, and all they needed were more cycle, more power, OK? They could sell computers like it were cookies.

And if there were more powerful machines, they would sell them right away. They didn't need more functionality. They didn't need systems that ended all other systems. They needed more power. So they kept disagreeing. Future System was technology-driven, and the sales people at the end were able to get the project killed.

But the technology was still good. I mean the underlying idea. So instead of great new functionality, the sales people got more power. I mean more cycles, machines more powerful.

And for example, airline reservations systems were running out of power. The computers were not large enough, or airline scheduling or radar tracking or all of these applications suddenly were running out of cycles. And with the killing of this project, another approach was taken where the hardware improvements were passed through as more powerful machines. And the idea of the system to end all other systems died on the fine.

So what I'm saying here is that yes, disagreements among the stakeholders or disagreements between the project manager and the stakeholders can lead to a discontinuing of the project. And every time that it occurs, it's not very pleasant, but it's much better than dragging on and wasting a lot of money and time and in fact leaving your salespeople without powerful machines that they could sell right away to anybody coming along and ability to pay for it. So that's the answer to one question.

What if disagreement on requirements? The first answer is negotiate. The second question is you may have to face the end of your project if the disagreement really continues.

Now, the second question assumes that there is agreement on the requirements. All the stakeholders and the project manager, everybody agrees these are the requirements. But the question from the class was, what if the requirements are inconsistent, defective? Whether we have realized it or not, there may be things in these requirements that are nonsense. So that was the question from one of you in a previous class.

And I answered that with basically remarking that requirements are not only, at least initially, defective, they are largely unknown to begin with. Every project has a beginning. You have to build it up. But even after you've built up a number of these requirements, there are the emergent properties. I call them the bane of the software project manager.

Emergent properties are properties that appear that you could not predict by making models or making prototypes, beta software. You find things, behavior of the software, that were hard to predict from what you put in. So that's one reason, the unknowability of software requirements, at least initially.

There are several more reasons, but another reason that I would like to mention is that whether we like it or not, the design that you choose after you've done the requirements may influence your requirements, OK? Over and over again, you run into situations where you start with requirements, so you come up with a design. You show that to your important stakeholders, and they say, yeah, but now that I know how you're going to solve this problem, I have the following requirements. So they go back to their requirements and reformulate them. And that's a very true case, and one that you have to accommodate.

So altogether, it's not just that may be defective, but they are, at least initially, largely unknowable as well. And you have to work on that.

OK. What you see coming here is, of course, agile engineering, which I have shown here with a spring and increments. That's how you can work this. What it is is an incremental evolution of the needs to requirements to the software, each increment based on the best available needs and requirements, each increment a complete engineering cycle, but short. The theory is three weeks. I'll talk about that. Each increment is released and used, and because of use, you get feedback. Again, this is theory.

But that is not theory, and that is, define quality as satisfying the stakeholders. And that's one thing that forever I will credit IBM with. They got us away, they got me away, and many other people away from the idea that the quality is a matter of meeting requirements. No. Ultimately, quality is satisfying the stakeholders, the customer, the users, but everybody else that is in that large group of stakeholders that your profit probably has.

OK. To a large degree, this is what you would like, that you hope will achieve. Again, I show this picture as a feedback loop. On the left, you have the stakeholder needs. And here is one increment, the engineering of an increment.

Remember, I always say "engineering." I don't say "development" or "acquisition" because that same loop applies to development and acquisition.

You engineer the next increment. You use the increment, and then you have new or changed or refined stakeholder needs and software requirements. And you start the loop again. And after a number of increments, you come to the end and you release your final increment.

One to four week cycle, dependency on users able to take the delivery because you're very dependent upon their use. And the stakeholder participation hopefully would be used and feedback.

Not too long ago, I've done a project very much in the spirit of agile engineering. Sorry, I was a consultant to the project manager for this project. And the agile method that they used was the well-known Scrum method. And they did everything according to the book.

But what we found altogether was a number of very practical problems. The first one was that designing in small chunks is nearly impossible, certainly impractical. You need more force. You cannot take part of your backlog and design that and hope that that design will fit with what comes next. So practically speaking, that was a big problem.

And sure enough, before too long, we had an architecture and design group on the site looking further ahead and testing the each increment against their overall design. But that was already taking away from the pure idea of agile engineering.

We also found that the quality of the increments-- getting the increments to a quality where they could be used was nearly as big as getting the final system to an acceptable level of quality. So most of the time, the one to four week rule was violated. We kept the increment longer in tests. And even then, it was very hard to get to quality.

But altogether, the time it took-- I mean, this was a company that made some business software. Very much for their cash flow it was very important that they came out with new releases every 12 months and that the new releases had quality. They had time problems and they had quality problems. And customers were not satisfied with new releases. And if your company depends upon satisfied customers, they were feeling it.

So that's why they went to agile to begin with, after which I was called in to see if I could help them. Altogether, the project did not take 12 months. It took nearly two years.

So what I learned, at least in this particular case, was that certainly it's not faster. Yes, there's this idea that after one to four weeks you have a new cycle and you can do some testing. But that's not how it worked, at least in this particular case. So it took much longer.

The absolutely exciting thing-- and at the end, the company was very satisfied with what they had done, including the agile approach-- that the quality of the system, the customer satisfaction with the new release was great. It was bad before, but the customers were excited.

Yes, they had to wait, but for a customer to wait nine or 12 months, that's not too bad. For a company that is dependent upon cash flow and income, it's very bad. But the company was able to overcome that.

But at the end, the customers were excited about the functionality and all the other attributes of the software. So in my book, at least based on this particular example, it's a great way to get to customer satisfaction, which is one of my goals. But it takes time. And it's certainly not a speedy method to get over and done. Yeah?

Talking about the designing in small chunks, the big part that includes is the impact assessment that you need to do on the existing system.

Absolutely.

And there again, when you have a distributed system, for example, you have the same software launching in North America, and you have that same software launch in Europe and in Asia. Because of the different countries involved and so on, the impact assessment that you have in North America could be significantly different than the same software functioning in Europe or in Asia. So the one to four weeks rule, though the software works perfectly in North America, might absolutely blow up in Asia.

Yeah. I will give you a little bit more about the background of this. I am not going to mention companies, but this is a company that makes time keeping software and time keeping hardware. If you look at your local Wal-Mart store, it has a couple thousand employees, and time is kept with a machine, and carts and software that tracks all these things and makes sure that you comply with labor laws. There's a whole lot. There are million of lines of code behind this.

And you're absolutely right. This whole software and this whole system is not just for North America. It's for the world. It's for China. It's for Europe. And what we may think here works may not work over there.

You get into different-- like I already said, labor laws are very different between jurisdictions. So there were complications galore to this. To just think that after one to four weeks you can get back to another working system is baloney. More questions.

I still now and then go to a store and I casually ask people, I mean people that work at Trader Joe's and places like that, you like your system? Oh, yeah, yeah.

Part 2: Software Project Management

First of all, project management is a learnable skill-- as opposed to, for example, design. I've come to the conclusion over the years, design is very hard to teach. Either you have the gift or you don't have the gift. On the other hand, project management, you can learn. And in this world and in this country, there are many institutes, including universities, that do a very good job in teaching project management. That's one thing.

The other thing is this is transferable. It's a transferable skill. Not only is it a teachable skill, but it's transferable. I've had good project managers coming from very unrelated engineering fields-- I admit that-- and do and pick up on the software aspects quickly enough that they could do a very good job.

In fact, one of the best project managers working for me at one point was somebody, a Dane. He had been the project manager for a project that built one of the bridges. Denmark is made up of many islands and there are suspension bridges between some of these islands.

And he had been the project manager of building the [INAUDIBLE] suspension bridges. And he became a very good software project manager. And the reverse is true, as well. It's a learnable skill and it's a transferable skill.

I will be, in this presentation, in this part, in this second section about project management, I will be talking about one particular case that I was involved in. I call it System X. I will not give it its real name. It has been long enough, but it's still in the sales manual of the particular company. So I don't want to get into legal problems here. So I'll call it System X.

The picture is real, by the way. That is the System X that you see there. It's a large, complex system, and my responsibility had to do specifically with the software. That system, it was a product family in the communication markets, in the telecommunication market, was a make-or-break product for the company. In other words, so much was invested in it that the better would have success with it or bad things would happen with the company. It's was not a "doesn't matter" kind of project, not at all.

In my case, I had 1,000 software engineers altogether, spread over 10 centers and three continents. Now with IBM, I already had learned to manage in different time zones, on this side of the ocean and another number of engineers on the other side. But here, I not only had the United States and Europe in a number centers, but I had Taiwan as well. So I really-- I learned to deal with different time zones, and summertime, and daylight saving time, and all of that.

And that project, before I became the project manager-- Yeah and I became the first project manager-- was already a couple years under way and it had continuous coordination problems. And then that, in fact, gave the top of the company the idea that hey, if there are coordination problems, maybe we need more than a coordinator or a set of coordinators. Maybe we need a manager, where all the responsibilities come together.

So I will be using this, and I will tell you how this project fared, and I will refer to it a number of times. But let me tell you about the typical software project management problems. All of them experienced in this project, but they are typical for a lot of software projects.

The number one source of failure of a project is the fact that people try to do it with project coordination as opposed to project management. And I will explain the difference between the two. And in fact-- but that's precisely the number one-- was the reason why the company decided we needed a real project manager as supposed to some guys that do schedule coordination, resource coordination. No, we need something more.

But assume that you have settled, that you're OK with number one, that you do have a broad project manager. The second typical problem is that there's little or no planning, or not adaptive, agile planning. Always surprising to see how people try to do things without clearly envisioning what they are going to do, when they're going to do it, how they are going to do it. But that's the fact.

But assume that you're OK with number one and number two. You have brought project management. You do planning. But then the next problem is the plans that you produce are implausible-- meaning that there's nothing in the immediate past that suggests that you can do it this way. Yes, I mean, you may argue that you should be able to do it, but there's nothing that tells you you've done this before, and you know what you're talking about. So that's the number three.

But assume that you are OK with one, two, three. You still may run into trouble with execution. Execution is easier said than done, as well. Execution of a plan-- the tracking and follow-up.

But assume that that is OK. And this is, in other words, a hierarchy of problems. Number five is the risks-- and there are always risks-- that you encounter are unmanaged in such a way that small problems grow into larger ones.

But assume that that is OK, the number six, the source that can kill you is that if you're not careful, you build up expectations among the stakeholders around you that are on unreasonable, and that you are never able to satisfy. And then the unnumbered, the wild card, in all of this is personal.

I will touch on each of those very quickly. And as I already have suggested, I'm going to talk about what project management includes. But before I do, I need to make sure that we agree on terminology.

Very quickly-- and there's no disagreement in the field about this. Everybody will repeat this definition. A project is a way of organizing work with a beginning and an end. It leads to a unique results. And it's the opposite of ongoing work. For example, maintenance work is typically seen not as project-oriented or projectizable but as ongoing work, repetitive operations.

OK, talk about organizing a project. I'm working on terminology here. There are three archetypes to talk about here. Actually, there are two archetypes-- one is the functional organization and the other is the project organization. I will give you more about that.

The [INAUDIBLE] matrix is already a mixture of functional organization and a project organization. As a matter of fact, most organizations in this world are matrix organizations, but I will pay attention to the two extremes, the function and the project. Also because in the software in particular, most of the organizations that we have are rather extreme project organizations. We try to do as much as possible as a project and as little as possible with this functional.

Now what is a functional organization? Here you see a diagram of a functional organization. It has a chief executive and it has-- this is all very simplified-- three functional departments. For example, in software, you can think about-- and this is, by the way, how things starts out. Originally in software engineering, many years ago, we started out with functional organizations, because everything was new and the skills were not very broad.

So on the left, it was not unusual to have a design department. In the middle, you had a programming department. And on right side, you had a test department. And along comes a project-- for example, a new print application. And what would happen is a coordinator would get a designer, he would get two programmers, and he would get one tester, and that would be his project. Meanwhile, these people were reporting to a functional manager, to the design manager, the programming manager, and the test manager.

And that's-- in fact, at the very beginning, I worked in an organization like this. The good thing about that is that it in general in a functional organization, people understand their own work. They understand design. They understand programming. They understand testing. But they have trouble understanding the total, as you can imagine here, because it's pick and choose, as opposed to what you have with a project.

But the other thing, of course, that's a negative-- there's friction. If you have multiple projects that you have pushed through a functional organization, there are conflicts. I mean, you have resources that are taken away for another project. And definitely, a functional organization is unfriendly to complexity. What is happening then, often, if you want to work with a functional organization, you've simplified the work to the point that it becomes unreal, that's hoped-for and not factual.

But even today, most of the software maintenance work and software operations is done with functional organizations. And outside software engineering-- for example, banks are typical of a functional organization. They don't do much. They have a few projects, but in isolated departments.

Another example that I always like to use for functional organization is a university. A university is a very good example of a functional organization. You individually, as students, are the project. Each of you is a project. And you're pushed through various departments. You get information sciences here, you get accounting there, you get English literature there. You're pushed through the organization to pick up your knowledge.

OK, that's far outside software engineering. As far as culture is concerned, I learned, for example, that the French are still enamored with their functional organizations, more than-- Americans, somehow, are much easier to work in a projectized way. But you go to the French side-- and by the way, managing in France is a thing by itself. But one of the things is that they love their functional organization approach. They are at least-- and this is already a number of years ago that I last managed over there. But they love their functional organizations.

OK, which of course requires me to talk about the project organization the same way. Here you see the project organization. The chief executive-- and I've put a sponsor to that. And now you have three projects, three project managers. And let's take an unreal example, but Microsoft, the left one is the Word project, the middle is the Excel project, and the other is the Note project. So it's done by product that you have organized, and they are three projects with dedicated staff.

However-- and now I forget about Microsoft-- but this works if you can keep your teams to relatively small, 10 to 15. And when IBM started to do more work in a projectized way, organize in a projectized way, in fact the limit that they had was seven or eight, maybe nine, people per team. Because although it was a fully responsible project manager and [INAUDIBLE] complexity and uncertainty, the point is that the typical experience is that people working in a projectized organization, in a projectized way, they understand the totality but they have trouble explaining what their role in the totality is.

So you need a good-- and I called it a first-line manager, these managers. The terminology "first-line manager" sometimes means the guy at the top or the lady at the top, but I mean that in the sense of immediately with staff below him or her. The first-line manager continuously needs to explain to people what their role in the totality is so that they can do their proper job. OK, but always, still always, they have trouble grasping their own contribution in the totality. But at least for software engineering, for development and acquisition, it's the dominant model these days.

Now let's talk about-- I mentioned there a fully responsible project manager. Let's talk about what these project management responsibilities are. And I start by three slides, giving you the view of the Project Management Institute. It's a private organization, but it's a good reputation. They are one of the organizations where you can be taught project management skills, and they do it very well.

But I give you their view. They've written this down, and I wholeheartedly agree with it. After going through these slides relatively quickly, I will give you my personal take. Because I think there's a difference between the project management responsibility in total, as opposed to the project

manager individually. There's a lot of delegation going on, and not every project management task needs to be carried out by the project manager.

So let me start by going quickly through what an organization like the Project Management Institute sees as responsibilities. They have nine responsibilities, although I saw their new draft. They come out with standards, with guidebooks, and I think they have added one or two. But never mind, this is from 2008, their view.

One, two, and three deal the who, the what, and the when. The who is-- we're talking project management here-- is the organization. The project manager is the overall manager of the project, meaning that's where the buck stops. Whether it is technical, whether it is financial, whether it is-- that's where the buck stops.

Now financial, I showed in a previous page, there is a sponsor above the project manager. Somebody needs to come up with the money, or someone needs to come up with the bodies to do the work. But other than that, the project manager is the manager of the project. Total responsibility, accountability, and authority for the project and the project result. And there's much delegation of responsibilities, et cetera, down below, because an individual cannot do that.

Now I spoke about the limited size of the group in the previous pages, 10 to 15 maximum. If you really push it, you can have, instead of a project manager-- yes, the project manager, but he or she has a project management team of, say, 10 people, 15 people. And that makes it possible to cover not just 10 or 50 but, say, 15 times 15 people. So an organization of about 200 people, you can do reasonably well with a project manager in this sense.

And indeed, I've known a serial entrepreneur, a Swede, and he was a serial entrepreneur. You had terrific product ideas, software product ideas, and he pursued them one after the other. And he had terrific ideas so he was successful.

But he never grew his company larger than 200 people. If he had a new idea, he would create a new company-- separate, totally separate from the previous one. Because he didn't necessarily agree with me every word that I said, but he understood something about scope, and that with a team, a project management team, you can cover not just 10 or 15 people, but you can cover 15 times 15, about 200 people altogether.

And he understood if he would push it beyond that, his company would become unmanageable. And he would rather create a whole new company, separate finances, et cetera, et cetera-- no sharing of anything between those companies. So but that means that even with that, there's a lot of delegation of responsibilities from the project manager down.

The project scope is a matter of what, meaning that it's a project management responsibility to identify the stakeholders. Oh, that's where, in fact, PMI came out with one of their new responsibilities. They singled out-- they thought this one was so important, the identification of

stakeholders, that they made it into a separate area, [? 10. ?] So they took it out of number two, showing how important they, as I do, it is to get an identification of the stakeholder and the stakeholder needs.

But I'm not going through all of this here. The use of time is a project management responsibility. I go to the next slide, the costs. And this is far beyond the technical matter, as you can see.

But then there's quality. Let me stop there for a moment. Whoa, whoa, whoa, whoa, whoa.

Quality. Talking about delegation. Quality is highly delegated, to the point where I always work with the idea that everybody in my project is responsible for quality in his or her own way, whatever the tasks are. But it's not somebody at the top, that he or she is responsible for quality.

No, at the end of the day, it's all of us that are responsible. And if they are not totally responsible, we have to keep their feet to the fire, as I say there. And all of that-- remember, the previous subject of requirements? The ultimate measure is stakeholder satisfaction and getting the project outcome accepted.

Staffing-- moving quickly through the slides now. Communication-- I will come back to that. Stakeholder expectations-- I'll discuss that. I'll discuss risk, my view of that. And I will not discuss procurement acquisition, although it's very important, but we have limited time here.

OK, let me sum this up before I go to my view of project management. Here we have System X again. And let me tell you whether I got everything that I thought I would need and did not get.

I already mentioned the first three lines. I was the first project manager, because previous approaches of project coordination had not worked well at all, and the company was smart enough to figure that out. I got full project management responsibility for all the software. And not only responsibility, but accountability, as well.

Now there are three things in management theory. There's responsibility, there's accountability, and there's authority. So how about authority? I only have authority over one third of the staff. OK?

Now I had-- and I will talk about-- yeah, I mentioned the countries there. In each country, I had country managers for software, and they had functional reporting to me, but I could not hire or fire them. So as far as authority, I could tell them, I could press them, but at the end of the day, they had other managers. They had local management as well.

But I had direct authority over three things. Extremely important. I had full authority over the operating system.

Namely, I had a development center here in Connecticut of about 200 people, engineers, that were doing the operating system for this software package. I had a smaller group in England doing the

tools. And I had an even smaller team being the project management team in Brussels. I was located in Brussels for that purpose. I already had moved myself to the United States, but temporarily I was back in Europe again.

And it turns out, if you have these see things-- the operating system, the tools, and the project management function itself-- you have chips to trade. You have buy influence. I mean, these country managers were still kind of free to do more or less what they wanted, but they needed an operating system. They needed compilers, and I was to provide a folder of compilers. And they didn't care much about my project management, my plans and controls, but I had a say there, as well.

So I will come back to how this project fared, but here you see a typical example from industry that yes, the theory is you need full authority, but at the end of day, it's very seldom that you get full authority. You get the full responsibility and the full accountability. You have to send up what happened and not happened. But that's the reality.

OK. And now I showed you a Project Management Institute idea, and I agree with what project management responsibilities is. I'm now going to tell you my personal take in this System X project. And in fact, I always have these kind of preferences. Remember, I make a distinction between project management responsibility and project manager responsibility. I pick my own parts and I delegate the rest.

OK. I already mentioned that System X discovered from all of the problems. But given that-- and I'm going to talk about planning, execution, risk, and personnel. And these were the four things that I really paid attention with this System X project.

Let's first talk about planning. There wasn't a plausible plan. And as a reminder, unrealistic planning leads to real waste in time and effort. There are many planning methodologies. I'll give you mine.

That is based on points. We do with it iteratively, meaning as time progresses, you revisit. You plan the close-in activities in great detail, meaning the first three months. Between three and six months, you do it in between, and you only have a rough outline for whatever comes after six months. This leads me-- and that's general experience in the industry-- never believe a software project manager talking about anything beyond the next six months. He or she doesn't know what will happen.

And then important is I work with a golden triangle. I will explain that in a moment. And I do size-based planning, and I will explain that.

This is my golden triangle. There are three corners, and somehow you need to bring those corners into balance. And the scope includes quality, and time includes cost. Time and money, you can understand that, and scope covers quality, as well. And somehow you have to bring this thing into balance.

So I mentioned size-based planning. By the way, this grew over the years. No textbooks that taught this. And you will see, even if you go to a reputable course, you get slightly different flavor. But this is what I did.

I start with the requirements. Remember, the requirements is something that we have agreed on, hopefully, with your stakeholders. Otherwise there would be no project. And if there are design specs, they count as well.

We estimate the size of this thing, this software to come, to be developed or to be acquired. Size is one of the things-- people know about it. If I consult with projects, one of my first questions is, how big is this thing? Wild looks for a moment, and "What do you mean?" Or an answer, maybe, "This is a project of 50 people."

OK. I said no, tell me about the size of the software, the number of lines of code. Now when you say that, you may get a little story about how imprecise lines of code are, and I would agree with that. But it's better than nothing, so I work with that, and a few other things that I'll talk about in a moment.

And so from these requirements, I want to know, how big is this thing? Is this 10,000 lines of code? Is this 100,000 lines of code? 1 million lines of code? Whatever.

By the way, if you have that conversation with people and you press them to come up with a size, a day or two later, they will come up with a size. They say, this is 50,000 lines of code, small project. 50,000 lines of code.

OK, you do other things. You work with them in the weeks that follow. There comes a point in time where you say, gee, I mean, wasn't this 50,000 lines of code? I look at these test specs and this is starting to suggest much bigger than that.

Oh, doesn't matter, doesn't matter. I say, you really think that with the same number of people, you can do 50,000 lines of code and 500,000 lines of code? You're out of your mind. So you cannot.

I have a lot of respect, a whole lot of respect for Bill Gates. Bill Gates was one of the few people, very early on, that understood that the crux of the game was not the design or the architecture. It was simple lines of code. He fought for every line of code. I mean, he tried to acquire it, and he did acquire, a number of lines of code from IBM, a whole number of lines of code from IBM.

And he understood that that's where the investment in software engineering is. The investment is not in design. It's not in testing. So yeah, it's in lines of code that do work. That's where the investment is.

So very important to keep an eye on those lines of code. And like I said, you can tell me a story whole imprecise the measure of lines of code is, and I would wholly agree with that. But it's better than any other alternative that I have seen.

OK, here we have size. And the next thing you do is from the size, you estimate the effort. So you start working around this triangle. And you may-- size may not only-- you may throw in complexity. I mean, from size to effort, if it's highly complex, you do and you make it into a larger effort. But you can do informed iteration there.

Based on size and effort, you can do time and cost and schedule. And if the schedule is not attractive, you circle around a number of times. Like I said, you try to bring these things into balance.

Circling around means that you don't compress. But the only way to work on this is, first of all, you can come up with more people-- but I'm not a believer of that. Adding people to a late project makes it later. So the real opening there is that you can change-- you can cut your requirements so that the size goes down, and you can circle in such a way that you come out with everything acceptable.

OK, so that's size-based planning. Yeah, there's a question.

Do you actually-- when you talk about effort and you talk about lines of code, right? Your 10 lines of code could be a simple and it could have a conflict. So do you have some kind of a weighted--

Sure, sure.

[INAUDIBLE]

Guessing. Guessing. But you start with-- and you can iterate through this. And like I said, you only do it for the next three months in detail. Everything beyond six months is pure. So you roll this forward with greater and greater precision.

So yes, complexity comes in. And in fact, in a moment, I will show you that it's not just lines of code. There are other things that we can estimate, as well.

OK, this is an iterative process until you have these three corners in balance. Here's my-- it's not just lines of code, for size. But you could talk about requirements. And in my case, I already mentioned I like test-driven engineering. So I could talk about a number of test cases, what does the number of test cases suggest?

But you see, these are all alternatives to using the lines of code as a measure. The design-- I can look at design models and see if it's complicated or not. Architecture. The test is where I look. Parts to

be-- components to be integrated. The amount of document-- all of these things are quantifiable, and they tell me something about the size. But the basic thing is the lines of code. OK

No, we have done this. I make a distinction between iteration and increment. This is in increment. And as I said, the theory would be the one for the next one to three weeks or four weeks. But this fits in incremental engineering, this cycle, this plan-do-check-act cycle is also called a gentleman-- Shewhart cycle. You plan something, you do something, you check the outcome, and you act on what you find in the check, which means that you have to redo. That's incremental engineering. How does one relate to the other?

No, first, doing in software is a matter of developing or acquiring. At the end of doing, you have delivery. You have user feedback. And you have updates in requirements on the acting side.

How does one relate to the other? Here you see how for each increment, you do an interative planning process, and then you roll it forward increment by increment. OK?

This is what we did, finally, in System X. But there's more to it. A plausible plan has-- oh, oh, look, look. I need to point-- has commitment and buy-in. There may be a plan, but if nobody buys-- I mean, if you ask people, are you bound by this plan? Oh, no, no, no. Somebody made that plan, but no, I work other ways.

Or another, and that's really a trap, you ask the project manager, have people agreed with this plan? Is there buy-in? Oh, yes, there's buy-in.

Did you ask them, one for one, do you buy in to this? Oh, no, we had a presentation. Nobody objected. In other words, too easy to take silence for acceptance. The way I learned this at IBM was you have to ask each people individually, do you buy in 100% to this plan? And if not, why not?

OK. Brevity. Most plans that I see to begin with, when they start, there's-- there's too much detail, and to a point where you say, OK, this is your plan. Let's see. In this plan, it says that by now, you should have this and that. Oh, no, no, no, it's not that important. We'll do that later. I say, yeah, but it's in the plan. Oh, no, it's not important. My point is, things that are not important are not in the plan. Should not be in the plan.

OK. But a plausible plan also has identification of risks and dependencies. That speaks for itself. And a project plan is perishable, so it must have currency. I wholly agree with the slogan that what is important here is planning. The plan itself, as I said, is perishable. It can crumble very easily.

But the real crux of the matter is in the planning activity. Because that brings out the discussion. That brings out the disagreements. That brings out the points where you need to gain consensus.

And then common sense. When I get into a project, probably a project that is in some kind of trouble, and it's not too difficult to-- you say, OK, what is your plan? Have you done planning? Yes, yes. Here, let me see the plan. What is your planning?

And then the simplest question is, have you done this before? Does this make sense? Can you really do this in such a short time with so few people? Or for example, you see a plan that depends upon staffing. Assume they have 10 people now, and they depend upon 40 people in a little while. I've never seen a project take on that kind of additional manpower in a short time. So those are all plans that lack common sense.

And then a plausible plan also has a few major milestones, because you want to track progress and provide broad feasibility. And I'll explain that funding rely of this project, the continued funding also depends upon this existence of a number of major milestones.

OK. That is planning. I'll now get to the execution of the project plan, which from a project management point of view is basically checking a follow-up. I already mentioned that the project management, at the end of the day, is delegated all the way down to everybody. So at the end of the day-- excuse me-- at the lowest level, at the individual level, you have-- sure enough, in scrum, we know that as the daily standup meeting. You have the daily reporting. I like to do that in standup meetings as well, short, in the morning.

But then at the next level, the manager level-- whether it is the first-line manager or the second-line manager-- all of them weekly reporting against plan and forecast. Forecast is-- OK, this is your plan. But you forecast at next week, this really will happen and that really will happen. So reporting progress against planning forecast.

Weekly forecasting for the next week and the next four weeks. And weekly accounting of shortfalls and recovery plans. And a weekly accounting of changes in plan and forecast.

Yes, there's a lot of reporting. But it doesn't have pages and pages. I've worked for people and that's how I learned it. People have said, whatever level you were, if you cannot write it in a single page, don't bother. I'm not going to read it. So you have to limit yourself to what really matters.

But you also don't have to-- weekly reporting is not a matter of educating your audience. You may assume-- you should assume that your audience knows what you're writing about and what you are reporting about.

Every manager then manages to every item in his or her plan. OK, which means manage to every plan item. That which is not managed will not happen, except by luck. You really have to go after it. And if it's not that important-- I already mentioned that. Oh, it's not that important-- it should not be in the plan to begin with.

But also, we have to be realistic. Every plan has plan items that cannot be achieved in time. The point there is to recognize deviations as quickly as possible and recover-- hopefully locally, meaning without changing the overall plan, without ripping this problem into other problems. But if not by plan changes, which means cutbacks or delays-- I already mentioned that additional resources, most of the time, are not realistic, but I mention them there anyway.

So deviations can come in many forms. The two top lines, the first four examples, are all time-related. Outcomes not available in time, activities not available in time. They are easily observable, that's true. But all of these four examples that you have there, the trouble already has occurred. You are already late. I mean, your time has passed. So they're trailing indicators, as opposed to what I'm looking for, leading indicators.

Sizes. Again, sizes are a strong indicator for me. If I see things growing beyond a plan, I can see that there are delays sooner or later. If I see that we were dependent upon certain amounts of staffing and I see less staff, people will say, oh, don't worry. We have too few people, but I was promised So-and-so and So-and-so and a whole department here, and we'll make up. Now in my business, making up never happens. That is another thing. We like to believe in it, but it doesn't work that way.

Complexity, hired, unplanned. Hidden assumptions that suddenly you run into dependencies that were not mentioned before, and you start asking questions. Oh yeah, a new dependency. Could it impact the schedule? Yeah, now that you mention it. So these are all leading indicators of trouble that has not occurred in time yet, but to may soon well occur.

OK, back to the managing to every plan item, and I gave you examples of recognizing the deviation as early as possible without even relying on delay in time. More to say about managing about every plan item. I have learned over the many years that if you lay out your plan and you put expectations on people, people respond to that in the most positive way.

Now there's always the exception, but people really like to-- I mean, they feel part of the project They're made to feel part of the project, and people respond. But they respond not to talking, but to action-- your action, the project manager's action, the action of their first-line manager, et cetera. So you have to act on it, meaning that if certain schedule items need to happen, major milestones need to happen, you need to track that.

I was in a situation for System X. And System X, it was, and is, a distributed control system, meaning that the machine that you saw there has hundreds of cards, and on each card, there are maybe one, two, three, maybe 10 microprocessors loaded with software. So there's an incredible model of software and processing going on.

And this is distributed control, meaning that there was no central controller in this machine. We're always asked to show the principle of distributed control. We invited visitors to take out any board out of the machine with the intent of stopping the machine, OK?

And some people would take up the challenge, and they would take out a card, and nothing happened. Yeah, an individual piece, some subscriber lines may not work. But the machine as a whole kept working. So we asked them again, take a second card out-- your choice. The point is with true distributed control, you can take out a good number of cards and still have the whole system working.

The system engineering that learned this trick is very much based on the telephone network, nearly 100 years old, which is set up in such a way that if there's a big problem in Thailand, say, in Thailand, it doesn't affect the calls that you make from Illinois to Connecticut, to mention and example. The internet itself is set up a little less distributed control, and sure enough, we have enough hackers that try to affect large chunks of the internet. This machine was truly distributed control and stood out, but also meant that we have to write a lot of software.

The point is, at a certain point in the engineering of that machine, of the software, a cancer started to grow in the machine. The machine started to get deactivated. Not the whole machine right away, but it started to lose boards. I mean, they were not seen anymore. They were still physically in the system, but the system was not accounting for them anymore. And we called that a cancer.

The cancer was growing. It started with a few lost cards, the feasibility of cards, but it grew back. And sure enough, I was put on daily reporting to headquarters in New York. Here I was in Brussels and I had to gather my input from various places. But I had to report on a daily basis, because there were enough people high up that understood that this whole idea of distributed control was not totally proven yet. Will it work?

And this was so important, because here we could be on the trail of something that would invalidate the whole idea of distributed control. And I had to report on a daily basis to New York what the status was. But that told me also that even people up there, they took this very serious.

So I was after it day and night to get all the information. Not only get the information, but to get enough minds on this that we could get this growing cancer in the machine under control. And sure enough, we found the problem. It did not go to the essence of the idea, so we were able to avoid that.

Point is, you have to act on things. Just talk is not good enough. That doesn't impress people.

OK. Early items are as important as later ones. That which is important must be made first of all by management action. That's my summary of that.

I mentioned this sponsor. I want to go back to this. Remember this guy or lady at the top that has the resources, the money, or the bodies? And I, in this System X project, reported to and issues on a weekly basis way up the chain. I also had regular reviews with the top in New York. At least once a

month, I traveled from Europe to New York and back, at least once a month. And regular reviews once a month. I was the lead presenter, and often I was the only one.

And basically, this was for the information of the sponsor, to get him-- it was a him-- up-to-date on where we were. But also in that same meeting were the sponsor staff-- the chief operating officer, the chief technical officer. They were all sitting there, and they had their staff, and they were making notes, and they were listening. They were asking probing questions.

As a result of those meetings were not really course corrections-- I mean changes in plan-- not at all. But they were preparing for the next thing, namely about once in six months, you have these major gates. Remember in the planning, you put in a number of major gates. Here they show off what they do.

You had another meeting in New York tied to a visible project milestone like a delivery or an acceptance by a customer or something. I would be one of the presenters in that meeting, but certainly not the only one or the major one. But also, business stakeholders, the major stakeholders, in that meeting presenting their assessment of what was happening, their critique. And of course, remember the first meeting, the monthly meeting, we had the staff, and staff had done their things, and they're talking. So these stakeholders were all very prepared.

And this was the meeting where really, a global decision would be made. Yes, you go ahead with the project. No, we stop it right here and we'll do something else.

Remember the future system of IBM. I faced a number of those monthly meetings, go/no-go decisions. But if the decision was go, it meant-- IBM was always managing the technical people on the basis of bodies. At IBM, you got bodies assigned, and you didn't have to worry about monies. At ITT, you got monies. Theoretically you could hire more people, but you were somewhat limited, as well.

So these are two different approaches to managing your project managers. But one way or another, after that meeting, I was safe for another four to six months, until the next major milestone came up.

OK What-- let's talk about risk. Some people have said-- and not the least of them, Professor [? Beem ?] in South California, University of South California. His thesis is that project management is nothing else than risk management. I, to a large degree, d agree with that, but not 100%.

Anyway, I'll define what I mean with risk. It's a combination of the probability of an event and adverse consequences of the event if it would occur. There's many sources of risks-- actions, assumptions. I already mentioned dependencies. Estimates are a source of risk. You may have the wrong estimates.

In risk management, there's the evaluation of alternatives. There's the situation of fallbacks. Always know what you could do if you have a substantial failure.

But altogether, risk management is something that-- and I show that as one of my personal things. Always, I was always busy, not only planning and execution, but going after risk. Risk assessment and mitigation is continuous.

My last section-- and I have to hurry up-- is about process assessment, which is a form of risk management, a tool of risk management. Here I would like to emphasize a nasty source of risk, and that is you build expectations. You build unreasonable-- before you know, the expectations become unreasonable. And they really could kill you. I've come, a number of times, close to that because I didn't pay attention to how out there among the sponsors, among your stakeholders, you can come with expectations that are unrealistic and should not be.

So I learned to communicate up, down, sideways. Plain-speaking, blunt, all of that is good. The earlier the better.

The point also is that if you tell people the truth, somehow they have a way to recognize the truth. They recognize truth. I mean, naturally, at least with me, there's a hesitation to be blunt. But it's better to be blunt and tell them the truth than sugar it up, because then somehow they know that you're fibbing it.

OK, I talked about staffing. This is the fourth activity in my project management emphasis. Remember-- planning, execution, risk. I will talk about more risk in a moment, but staffing.

And staffing is number four. Numbers don't make up for too few good people. Bubbles in staffing are unrealistic. They never happen, or hardly happen. Level head count is-- and level head count means that you have people with many skills, multidisciplinary. So they are worth their weight in gold.

But also, if you have a plan that depends upon changes, or even if you have a level plan, you have to keep the people on-board, not leave. So you have to implement your staffing plan with the highest priority. I already mentioned the idea that making up later for shortfalls is nearly impossible. What you have to if there are shortfalls, if there are delays, extend the schedule or cut back on the scope.

Another question that came up, but this is the answer-- pick the right people to work with. Very-- well, it's the most important thing. People performance follows an exponential curve.

Good people-- these are made-up numbers, but to give you an idea. Good people work 10 times better than average people, and average people work infinitely better than poor performers. Poor people make work for other people. average people do the work. Good people avoid the work, to a large degree.

And I used to have a design component in here, in this presentation, lack of time, et cetera. So but specifically in design, by judging this thing, and good designers are rare, and it's a talent. You cannot learn the skill. But in good design, you can cut the work in half, as opposed to good tools can cut the work by 5% or 10%. And these are numbers that we have measured.

So what to do with poor performers? Identify them and grow them, or move them out. OK, that was relatively quickly, but let's talk about first-line management, because I get back to these poor performers.

If you have people-- first-line management, those are the people closest to where the work is done. They are very important. They are not only the technical leaders but the personnel leaders, as well.

Specifically in IBM, seeing there was a personnel department, but it wasn't very important. I mean, the personnel managers were the first-line managers, the managers in general. You had a management responsibility for the people that reported to you, and you had to do enough to make sure that they were indeed not only happy, but good performers.

All of that, specifically IBM. There's the other idea-- there's the Google idea. Google is famous for doing interviews, job interviews, and having the job candidates be interviewed up to the top of the corporation, to make absolutely, positively sure that you have the right people to begin with.

IBM did a reasonable selection, as well. You could not just sign up and join the company. But they had another way of dealing with this, and that was the first-line managers.

So let's talk about these first-line managers. They should be experienced. Technically excellent-- not just a manager, but technically excellent. Good enough to take over from any person in their group. Introducing new people, breaking in new people.

Also, good enough-- and not only good enough, but doing principle contribution to the design of the software or the architecture. Responsible for the design of test cases. Responsible for reviewing 100% of the code if it a software development. When it comes to-- and you probably in class have spoken about inspections, and I really believe in design inspections. But when it comes to code, I'd much rather have my first-line managers review the cold and do a code inspection.

OK, the first-line managers are responsible for group-level planning and tracking. Optimal work environment-- not just technical methods, but work environments. Are we having enough space? Are we close enough to the test center? Et cetera, et cetera. Uptime of the various networks.

Decides if the group needs to be limited. I mentioned that we started out, IBM, first-line managers could have eight, seven or eight, maybe nine people. I have heard that it has grown, even at IBM, to larger groups of 15 or so. But it's an investment that you're glad to make, because you get so much back for that.

What it means, namely, is that the manager gets to learn who the good, average, and poor performers are and can do something about it positively. He can find out whether the software will work, at least in his or her part. They can step in in the case of staffing emergencies.

Code reviews are better and cheaper than code inspections. Or the manager reviews are better than code inspections. And the benefits justify the increased number of managers.

Oh, another thing that you were not allowed to say, in IBM at least, was the term "supervisor." No, we are managers. Supervisor is too much of-- it's something less than an overall responsibility.

OK. How did the System X story end? There was a release 1 before my time, and that was totally the center in Connecticut. That's where the idea-- and at the same time, the internet idea-- this was the 1970s, really, I mean ages ago. It was my first big project. But there was this idea of distributed control that was catching on.

And so in this center in Connecticut, there were the people that had grasped this idea of distributed control, and they must have had an idea for release 1. But the fact of the matter was that this company was highly-- that their home market was not the United States, but their home markets were in Europe. This was ITT. I can say the name. I cannot mention the name of the system. And the home markets in Europe were the country B, that's Belgium, and the country G, that's Germany.

And so here we had release 1. Nobody ever spoke about that, certainly not by the time I joined, became the project manager. But I was very much involved with release 2, that was on the way. And the Belgians in Antwerp, they really had to grasp this idea of distributed control. But their processes were kind of chaotic, and certainly not what you've heard so far from me about project management.

So on the other hand, we had this non-starter in Connecticut. And the Belgians through they could do better. They started to work on release 2. And the Germans were also a huge factor in ITT, so they came up with release 3, and it had a very good process. I learned something-- not something. I learned a lot from them. But good process. But they had not-- maybe I cannot say that they had not grasped the idea of distributed control, but they were working with another local supplier, Siemens, and Siemens was absolutely not convinced about the idea of distributed control. And in a country network, you need exchangeable systems, so the Germans had trouble with distributed control.

I came along, and release 2 under my thing came to fruition. And release 3 came to fruition, kind of. All of these things worked well enough as a prototype, but certainly not the end product.

So here I was, and pretty soon while release 2 and 3 were already on the way, I came up with the idea of release 4-- the target for convergence. We wanted to converge the ideas of Germany, Belgium, and Connecticut.

But it was unsuccessful. Basically in analyzing this later on, I and my staff never had access to the German home to the stakeholders and the customers in Germany and in Belgium. They always got filtered opinions to us.

And so after a while, release 4 was declared a non-starter-- which is bad for a project manager. It's not good for his health. But the top of the company came along and saved me. Because they came up with a country and with the idea that both Belgian manufacturing and German manufacturing would deliver exchanges together into Norway.

And the customer can accept a lot of things, and like a PTT in Norway can accept a lot of things, but they cannot accept that this machine and this machine looks different if they are having the same number on their shields. So by coming up with a customer that we could talk to unfiltered and could get, we were able to get to a unified design, very much distributed control, and with coordinated processes. I mean, later on, I've learned more and more, but this was my first big project, and I would rate that sufficient.

After all-- and ITT sold this particular business, because the company, the top of the company, was absolutely not comfortable with having a good piece of software in their repertoire, responsibilities, the system became a great commercial success. And it still-- how many years later? It's still in the sales manual, still being sold.

And all of that in many countries, served by a single package, OK? Yes, there came dot releases, 5. I was personally 5.1, 5.2, 5.3. The last time I looked was 10 years ago, and they had come in the meantime to release 5.19. At that time, at least, there was never a release 6-- in other words, a step up in technology. And there was always the compatibility with this one. So great, great success.

On the other hand, that's long ago. Since then, I've seen the task of project managers only become more and more risky. In many ways we see a return, for good reasons, to functional organizations. For example, architecture is a function that typically gets functionalized, meaning that's several projects are dependent upon one and the same architecture group, because the skills are so rare.

There's more outsourcing. All of these things are a matter of-- where the project manager loses control. More acquisition, less development. More attempts to reuse, more service-based components-- all of that is-- and I've seen that project management today is harder than it was 10 years ago. It's harder than it was 20 years ago, et cetera.

Part 3: Software Process Assessment

A short subject on process assessment. I included that because this is what I've been doing for the last 20 years. Remember, I was project manager to begin with. And then the following 20 years, I've been helping project manage. I get invited in by the top of the company, but I'm asked to help

project managers to get on with the project. So the idea behind this is that if you have a better process, you have better software. I'll explain.

The first important thing to do grasp is that a defect does not equal a failure. A software defect is not a software failure. To take the state of the art at this moment. No, what was it-- six years ago, an authority in this area estimated, for good reason, that every million lines of code that good software engineers deliver have at least one thousand undiscovered defects in them. OK? I estimate that, in the last number of years, if you're really optimistic I think we have improved an order of magnitude. But that still leaves 100 defects in a million lines of code.

And I've told you what a million lines of code is, but I also should tell you that many systems that I deal with have 40 million lines of code, or 10 million lines of code. So that's bad news. The good news is that even with defects, software works most of the time. In my System X, we had about 5 million lines of code. Using optimistic ID of remaining d-- by the way, this is a very vague notion. I mean, there's no precision to this ID. It's a guess. It's basically unmeasurable. But again, there's nothing better and we can live with that.

It's reasonable to assume, and the evidence that we have with some Bayesian logic, its says that in the five million lines of code the System X at the time, there were 5,000 plus defects, one thousand in million. With that, we could achieve 5/9 reliability. Meaning that if a machine is up 40 years, you only have 200 minutes of downtime. So yeah, this was just a bit of control, and remember how you could sync stakeout, and things like that. But everything that we know about that is that this 5,000 defects, at least 5,000 defects, we still achieved a measured-- not in hypothesized, but a measured-- at least 99.999% reliability. So defect is not a failure.

On the other hand, if you know that the Boeing Dreamliner has about 10 million lines of code on board. And if you take an optimistic thing of 100 defects per million lines of code undetected, this, your plane, you're flying with 1,000 defects at the same time. Now, that may be in the entertainment software, but something may spill over from one system to the other. And a lot of the real big troubles are always spillovers.

OK, but the point is that we really need to improve, even further, the software from where we are already today. And there are too many fields here where even a single failure is unacceptable. Medical devices, nuclear energy, obviously security, railway, Wall Street. A single defect night, the brokerage where the plus and the minus was exchanged led to the fall of the company, an immediate loss of a $300 million, $400 million and it was the end of the company. So single failure are unacceptable. Still, a defect is not a failure.

But another thing to be pointed out there, these fields are highly regulated, all of these. So we, other people as well, we estimate that we need another one hundred times improvement in remaining defect density. In other words, one defect per one million lines of code. But then, the observation is that we have reached the limits of testing and redundancy, which were the basic

reasons why you can have so many defects and not that many failures. So the focus now, specifically in that regulated environment, is more and more on process.

But we're talking about here, of course, is processes. You asked me what are the processes. Depends on who you ask, but I saw the standards organization has one. This is their set of software engineering processes. I use this as a reference model. The Project Management Institute has a slightly different one. Instead of 30 plus, they have nearly 50 processes. And they don't really map very nicely to each other. But you get the general idea of what software processes are.

What we really are talking about here is the risk of stakeholder dissatisfaction. Remember, I promised one more thing about risk. Here it is, what to do. I already mentioned what we have done so far, which is basically removal of code defects. Testing is hunting for defects, then fixing them. OK? And you can do that to a certain degree, but fixing a very defect-prone piece of software is impossible. So this has its limitations, its own limitations.

It's also, by the way, it's after the fact. The code defect is already there. So you're talking about, instead of removal of a defect, you like to prevent the defect. Which you can do if you hunt for defects in work products, in specifications, in test cases. And that helps. But what I'm after is the third one, and that is the prevention not only of-- not only the removal of defects in these work products, but the prevention of defects, even in the work products. And of course, in the code, itself. Which means that to hunt for defects in the processes as they are used, as they are planned, and as they are designed, so that you can fix them before they hurt.

Here's the model, just as a reminder. Here is stakeholder dissatisfaction, software failure. Software defect causes the software failure. This is the thinking behind software process assessment. The thing that I've done for the last 20 years in helping project managers.

So how do you hunt for process defects? And there are two ways. Root cause analysis, and I still do that. For example, one company was, of course, sending out invoices to their business customers, a telephone company. But the invoices were not going out in time. Sometimes they were a couple days late, sometimes they were a week late. Not only does it mean cash flow, but it turns out the people that receive those invoices are fairly dependent. They know it's the 10th of the month, so we expect the invoice coming in the for the services that we have used from that company.

So in general, they had a problem, this on-time delivery of invoices. Which means a lot of the software-- there are millions, literally millions of lines of code that are getting into the collection of data and getting that down to the invoice that goes to the customer. So that was a problem that had already occurred. And that is the draw back of this root cause analysis, it's after the fact of the problem actually caused trouble. Oh sorry, let me take this-- can I take this back? But by weekly going after the one or two big problems, asking questions. What was the cause of this? I never want to see this problem again. What do we need to do in order to never see this problem again? In doing that week after week, in a matter of months we had the system where it reliably delivered monthly invoices to its customers on time, in a matter of the same day or the next day.

Asking questions. What was the root cause of this not going up? I mean, before you know, you go back too far. And you have to go only that far that you can do something about it. But root cause analysis is a very good method. But what I'm really after is software process assessment, which is an activity that analyzes how work is being done as it occurs. Looking for anything done or not done that could ultimately result in stakeholder dissatisfaction. So it's a matter of observation, as it occurs. That this what software process analysis is.

And who does this? One or more processors. Dr. Hefner and I have done it together. I've done it in other places, as well. It's done independent from the project manager. There's nothing that the project manager couldn't do himself, but he's too partial. So you lose something, not everything, but you lose something if it isn't done by a group independent. And typically, we are then called in by either the guy at the top or the lady at the top, the Chief Technical Officer or deficient president. That's our sponsors.

The "what" is the processes of one or more projects, or to processes of the organization. Sometimes we are called in to evaluate an organization. For example, one company very much wanted to sell itself to Cisco. And they wanted to have a clean bill of health, or at least know where their weaknesses were so that they could show Cisco that they were aware of certain things, and that they were having certain plan to do with it. I've done that.

We've done for a whole organization. We also do it for projects. In one case, I did it for two competing projects, where AT&T wasn't sure whether they would continue this project or this project. They were in competition with each other. Typical for a large company to do two things at the same time and then make the decision later. But they were not sure about the processes of this company. So we looked into the processes of this project, excuse me, and evaluated that then. Whatever they did with it, I don't know. So you can do this in various environments.

The "when" is as I say, sometimes it's after completion of the processes, it happens. But I'd rather have it in the course of the process performance. And even better, if they do it in the planning of the process, because that's really ahead of the game. OK?

And nearing the end? How do we hunt? Here we are, I'm the assessor. I interview selected project members and stakeholders, very much stakeholders. And if you have multiple assessors, they can all do each about 30 to 40 interviews in about four weeks. You have leading questions. You really try to get these people to talk.

The first question is, is the configuration management performed? And if they say yes, you ask who, what, when, et cetera. Is the process performance, is the use of configuration management planned and controlled? And if they say yes, you ask follow-up questions. Is the process designed? Again, the who, et cetera. Is the process predictable? What are your standards of performance?

The initial answer always is yes, it's predictable. But then, if you start asking the follow-up questions, show them to me, you get often-- This is, again, a hierarchy. You start by performance, and if configuration management is not done, you don't have to worry about the planning of configuration management or the designing of the configuration management. But this is a pattern that you repeat to get to the facts.

And if we sense that with these questions that there are problems, we drill down. And if there appears no problem, we are not wasting time. We move on to areas where there seems to be a problem. And indeed, we even look at a piece of code, of work products. But definitely, we don't do any extra testing. We all rely on the organization, itself, to give us the evidence.

OK. The outcome of an assessment effort is the [INAUDIBLE] report in the presentation. I'm sorry to say, but that's what the sponsor needs. The findings are reported and they are unshakable facts. I mean, typically the sponsor when we present these reports, has the subjects there, as well. If they would object to the findings, then we are in deep trouble. Because these findings need to be unshakable facts, and we make them unshakable.

Recommendations of what needs to be done, and action plans. These are the general increases where process maturity should increase. And action plans are kind of IDs on what to do. At the end, it's not unhappy talk. If the project manager sits in the meeting, you can see him or her stiffen up. It's terrible. For some people walk out, cannot take it anymore. But here's the top of their company, so they cannot make too much of a spectacle.

We don't fix any problems. We leave it to the project manager and his team to fix the process defects. And if the sponsor's nice, he keeps us on a little longer to monitor the implementation of the action plans. It's a proven management tool, and specifically a risk management tool for stakeholder satisfaction.