Software engineering and software quality ten years from now

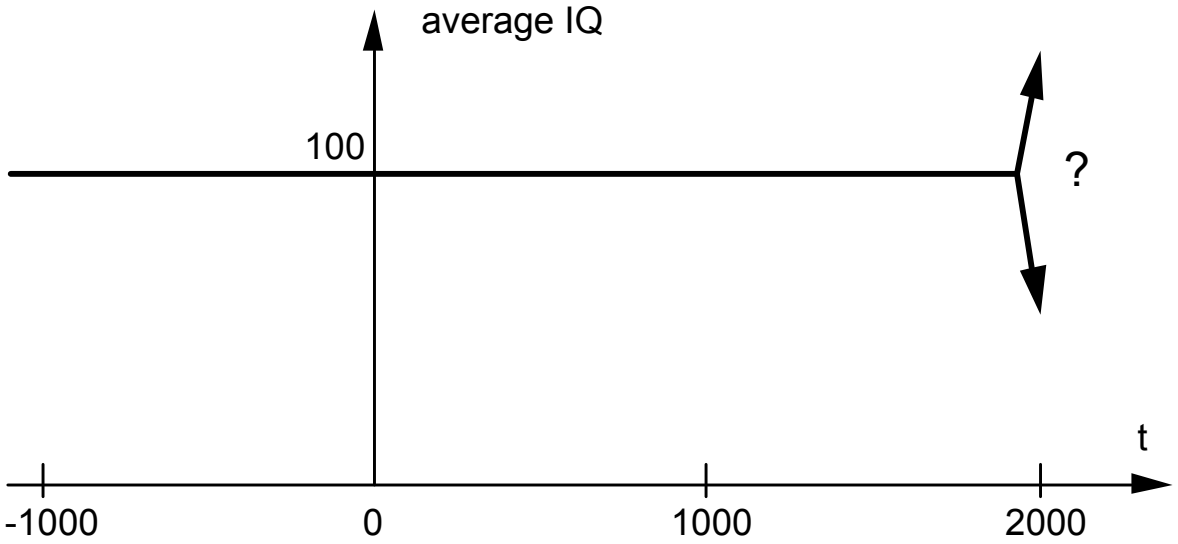
Jochen Ludewig moderates a panel discussion with all speakers

Speakers
- Jochen Ludewig
- András Pataricza
- Tom DeMarco
- Niklaus Wirth
- Norbert Fuchs
- Lucy Suchman

and votes from the audience
- Martin Glinz
- Klaus Jeppesen
- Karol Frühauf

Jochen Ludewig: The idea of this final session is to have a panel discussion with all the speakers of the day. Our host gave us a topic: Software Engineering and Software Quality Ten Years From Now. My understanding is that this is just thought as a starting point for the discussion, so we are certainly not limited to that. And I don't think we can limit the topic in that way and we don't need to. So, please feel free to participate in the discussion and also to ask questions which are not precisely within the boundary of that topic.

In order to give a very quick answer from my point of view: I have a very pessimistic view for this. I think if you look at the average IQ of the last 3000 years you probably have something like this



Most changes happen in the technology rather than in the brains. Cheap equipment, standardised OSs, fast networks have had more influence than many great ideas, including OOP.

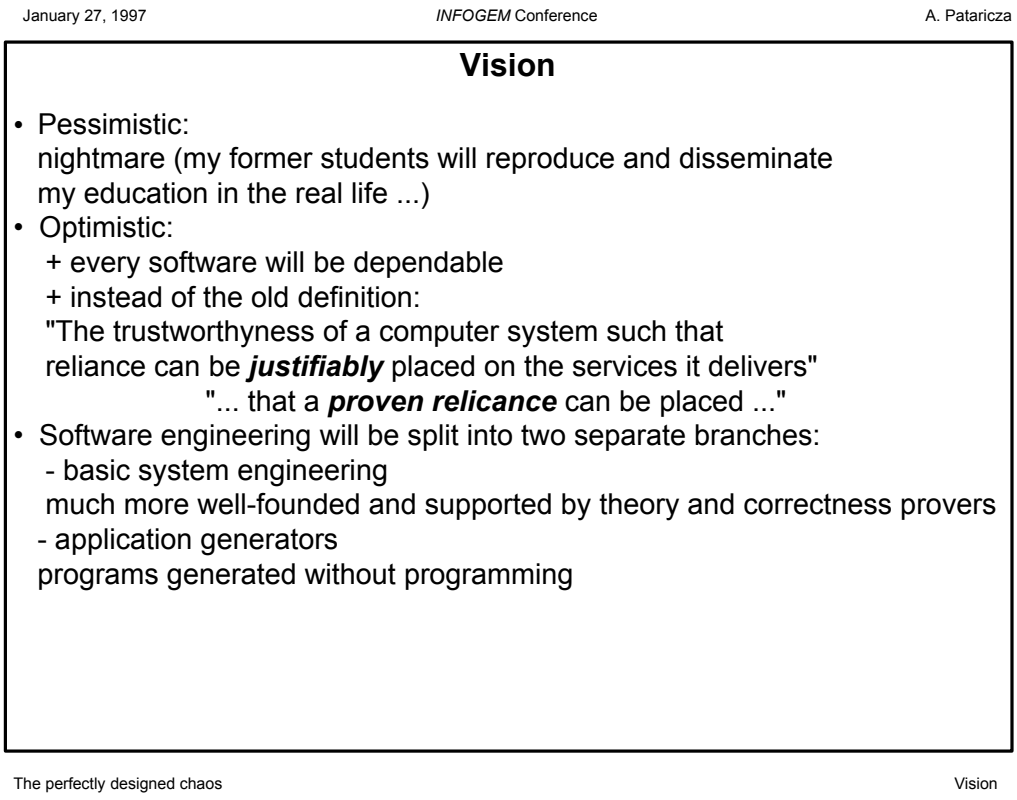
And I don't think that much will be changing in the next ten years, so my answer is: It will be just like today. But that is only meant as a starting point, and I'd say we start from this end of the panel, and shouldn't have more then three minutes per speaker.

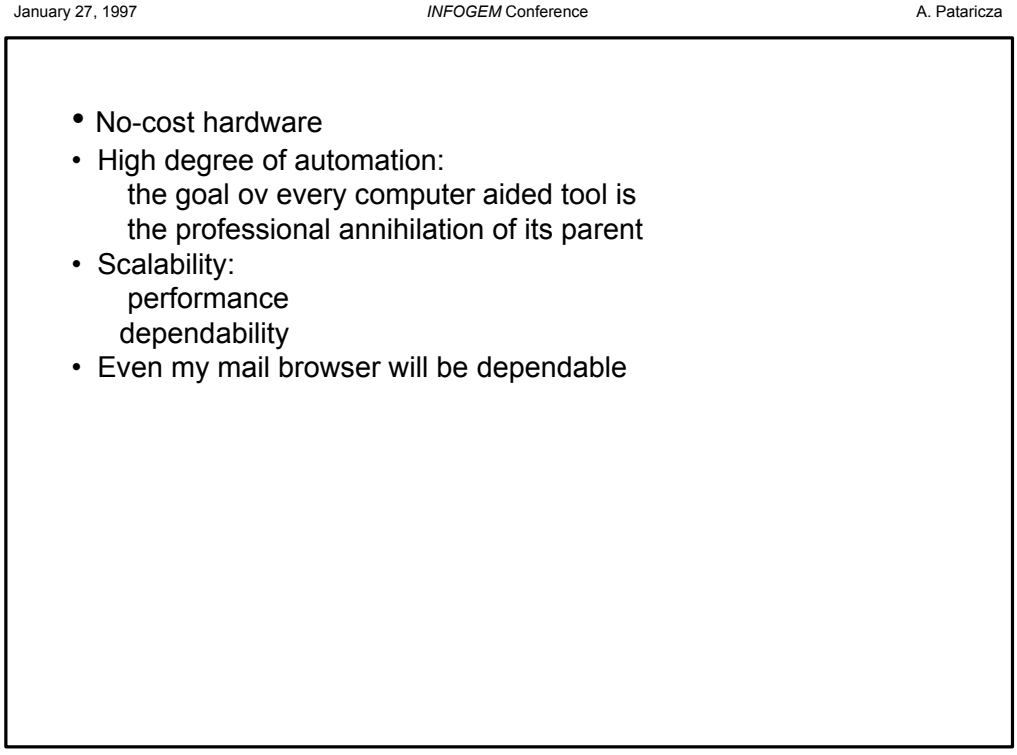
András Pataricza: As you know, I am absolutely in favour of dependable computing and in my vision there are two answers. The first one is pessimistic. That is much more like a nightmare. I'm afraid that I am as a teacher too suggestive and my students will produce goods which have only exception handling routines and no single line of functional routines.

On the other hand I am a little bit optimistic: I mean that every software will be dependable. I still remember a Euro-Micro Conference in Zurich at which there was a demonstration of a new software and both the software crashed and the machine crashed – that was an interesting spectacle!

On the other hand, instead of the old definition of dependability we will have some proven dependability of the new products. My vision is that Software Engineering will be split into two separate branches: One for the everyday people for easy programming, easy specification validation, and the other branch of system programming will get much more support from mathematics. Thanks to the low-cost hardware we will have a high degree of automation in software producing. As you know, the goal of every computer aided tool is the professional annihilation of its parents. That means a lot of people will work on such sophisticated tools which make that kind of programmer superfluous. Finally we will reach a point when

performance and dependability are equally scaleable, and I still have a dream for ten years: that my mail browser will be dependable as well.





Tom DeMarco: First of all I disagreed with a lot of what Klaus Wirth said, and Jochen has promised us a second round in which we can attack him, an ad hominem attack against Prof. Wirth. But let me take the first round to state my view of exactly what the question asked, i.e. What will Software Engineering be ten years from now?

I think we have gone through a brief period in which what we called Software Engineering was really an attempt to practise a kind of software science, a kind of computer science. We tried to do what Klaus told us to try do, to build things right in the first place, and, as Klaus said, we did not succeed at doing that very well, we were very largely a failure, but my point is that in doing that we were practising a kind of science.

I believe we are now going into a period in which what we call Software Engineering will indeed be engineering, and here I think the key note today on that subject was sounded by András (Pataricza), when he talked about building dependable systems made up of undependable pieces. Now we know how to do that because human systems, like the one that Lucy (Suchman) described in her talk of the control room for the ground operations of an airline, are made up of undependable processors – that's the people – and yet we know how to deal with that undependability, laying out the techniques that András abstracted for us this morning talking about different kinds of redundancy and cross checking and watch dog processing. We know how to do that for human systems, and I think ten years from now we will indeed be developing dependable systems made up of very largely undependable pieces.

That doesn't mean to say that we shouldn't do what Klaus has been encouraging us to do all along, which is to build those undependable pieces as correctly as possible; that is absolutely true. If you watch him through his career in software he spent a lot of time telling us about building it right the first place, but he made three very beautiful languages that help you to deal with your own failures. The languages Pascal, Modula, and Oberon, each one of them makes an enormous concession to the fact that, though you're trying to do it right, you *will* do it wrong, and they help you to isolate, to find, to test, and to deal with your own failures. He didn't yet go to another, higher level that András is talking about, of building dependable systems out of undependable parts. But I don't doubt that no-one could do that better then Klaus did, if he turned his mind to it.

And then I believe that after those ten years have passed, that even the idea of Software Engineering will cease to exist as an engineering discipline as well. Because eventually, I think, as the size of the thing that you build becomes ever smaller compared to the size of the environment that it lives in, and our human capability of understanding that ever larger environment, which contributes a greater and greater degree of the functionality of each application that we build, I think we are going to pass into a stage in which we cannot honestly talk about engineering our systems, anymore than you can talk about engineering your teenagers. We have to talk then about a discipline more akin to psychological medicine, to deal with a system which is fallible at many levels and probably largely beyond the comprehension of people. And it will become I believe, a softer and softer science, it will be appeal to a smaller number of more intellectual people, people who have the time to learn how to do correct design and at least to isolate themselves from the intrinsic fallibility of the system that they are building around.

Niklaus Wirth: Well I hope the future will prove that what you said is absolute balderdash. This morning I heard a talk in our institute, by somebody who is going to give that talk at a conference next week. He was talking about measuring performance of systems with caches, second, third level caches, and master-slave configurations, all kinds of things, very complicated systems. He did a beautiful job of measuring, but when he tried to explain what we should read out of these data he actually pretty much failed. I feel natural scientists are used to making measurements and observations of very complex systems. But, you know, they are presented with systems from nature, they haven't made them. They have to live through their misery because it was made miserable by nature. Now *we* are engineers and we build machines. Why the hell do we make our machines so goddamn complicated that we don't even know how to read data and measurements out of them anymore? I have two daughters, too, and they caused me great trouble and they are so complicated that I haven't understood them yet. But I'm really taking care that the designs I make are of a different kind.

Norbert Fuchs: I have three sons, the design was different to yours.

I will try to learn from history. knowing it is unusual in Software Engineering. So let's go back to what was ten years ago. Ten years ago we released the first quality management standards. We released CMM. TQM was still no buzzword. So without going into details I think these technology aspects were improved quite a lot. But looking to our customers: Ten years ago they had big problems with our products, and today they still have big problems with our products. So where does this discrepancy come from? We can discuss that; I'm sure you have a lot of ideas were it might come from. I want to give one explanation.

I think ten years ago we just could not handle the complexity of the systems. It was on a level we just could not handle. And today the complexity of the systems – again we just cannot handle it. This means that the improvement we did in technology and in management was compensated by the improvement we made in the complexity of our systems. And as long as we cannot change this I don't see any reason why our situation should change in the near future.

Niklaus Wirth: We always go to the limit.

Norbert Fuchs: It's part of human nature.

Lucy Suchman: I am not trained to predict the future so I generally tend to avoid it. But I'd like to point out to you what seems to me to be an underlying and so far pretty much implicit theme in much of what has been said today, which is the theme of intensification. It's very much an echo of the quantity vs. quality distinction that Klaus Wirth made. But in much of what we've been talking about there is a common theme of the interest in developing more and more in less and less time. Because there are, I think, largely economic forces, short-term economic forces, the logics of which overwhelm just about all the rest of our rationality, or which are very difficult to resist with all of the rest of our rationality.

So I guess the pessimistic view, which I'm afraid I lean toward, is that until we address the much more fundamental issue of the priority of producing (pauses) I think a good example is the attempt to create more and more software with people into whom less and less time has been invested in terms of their training and development. That, unless we can take on that much larger problem, which I think is a very large problem indeed, then things are likely to

keep going in the same direction that we've been concerned about today, unless we might reach some logically absurd extension of that, where we all suddenly come to our senses and begin to cross over and head back in a different direction.

Jochen Ludewig: Thank you very much, Lucy. So this was just the first round, and as Tom said he's now the next on the list. Tom, may I ask you to limit that to three minutes? And then, the audience.

Tom DeMarco: I disagree a little bit but I'm really just going to complain that Klaus exaggerated too much, which he said he admitted he was doing. I want to preface this remark though – this doesn't count as part of the three minutes! – by telling you that in my younger years in the early eighties I had a nearly religious experience. I went to San Diego to the Fifth International Conference on Software Engineering where the keynote speaker was Niklaus Wirth. And he gave a talk that I can remember today. (To Wirth:) I can, I think, remember that better than you can, because it's engraved on my mind. It was a beautiful talk about the business of building the Lilith machine and the business of constructing Modula-2, inventing Modula-2, or Modula. And it told not just about their successes but about some of their failures, and in particular, about their long standing battle against complexity; they vowed to make it simple, to make everything simple, and then in various places they found that the best they could do was still complex, but they valued the battle they had fought.

It was a very beautiful talk. I was filled with admiration for the talk but also envy, because there was Klaus standing in front of the most respected conference, the ICSE, and I was sitting in the very back wondering: is it possible that I'll ever have such an honour and have all of the ICSE listening to me? Well, that was the 5th International Conference on Software Engineering; at the 18th International Conference I *was* honoured to be ask to be the keynoter, and I can therefore assert to be the only person in this room who is only thirteen years behind Niklaus Wirth.

Klaus, you chided us by saying that you hope it's balderdash that the systems are not going to grow beyond a complexity so that we have to build our sensible, dependable systems, as András suggests, out of intrinsically undependable components. I wish that were true, too, but I do think that what's happening is that our environments are growing enormous. You say we didn't build the natural systems that cause natural scientists to have so much difficulty even interpreting their own data. But that's more and more true of the system that's built by these people here. They didn't build Windows, the didn't build the Apple Finder; it is something whose idiosyncrasies and whose failures, whose fallibilities are important to them and they have to build around them. That's a minor example today, but ten years from now the environment will be yet more orders of magnitude larger on the typical thing we build and we will need to be able to deal with the incomprehensibility, or the vague comprehensibility, the soft comprehensibility, just as though it were a natural system.

I think that is going to focus us away from any concept of building things right in the first place. Although we don't abandon the concept, we don't stop trying. Because it's satisfying, because it's good, because it pays off, but we just are never going to succeed. As you didn't succeed in absolutely defeating complexity, only in sometimes making marvellous little victories over it. I think eventually we are all going to have to live in András's camp of dealing with all kinds of intrinsic fallibility and assembling the infallible, dependable components out of these, as we do with human systems.

You said we don't have a topic of design in our curricula. Balderdash! I learned design at the feet of a master. I learned it through the use of languages that were built by masters. The master designers of our time have, thank goodness, designed languages, and through those languages we have learned the concepts of design. From Pascal we learned the concept of thin interfaces, we learned the concept of pushing down the complexity. From Modula we learned first hand, by direct experiment, by direct example, how to deal with data hiding. From Oberon we learned about extensible, buildable, inheritable data types. From FORTRAN we learned from John Backus, another master, we learned some good things from him, we learned about the concept of a function. Backus said that when we built FORTRAN we did not make a distinction between F = SIN (THETA), whether SIN was an array indexed by THETA or whether it was a function that computed upon THETA. We did not make a syntactic distinction there because it was important *not* to have to, it's nobody's business except the person who implements SIN(THETA). So we learned something from Backus. When we use a language like Eiffel we sit at the feet of another master, Bertrand Meyer, and we learn about the concept of contracts between modules within a program, a very beautiful concept.

Now, if we were learning from C I would agree with you. Because then we would be sitting at the feet of a group of hackers to learn how to do design. But I believe that today we *are* learning how to do sensible design, because we are learning through these languages still today, and even some systems that are built in C are very beautiful systems, both internally and externally. Let me give you the example of Photoshop or any of the Photoshop components. They are very beautifully externally designed, the human interface is very beautiful, and internally they are built by somebody who understood Modula, who understood Oberon, who understood the lessons of Pascal and built them accordingly. I do worry that a generation may be learning at the feet of Thompson and Ritchie instead of at the feet of Wirth, Meyer, et al., and I'm worried about that, too. But I do wish, Klaus, that you hadn't left us in the software world at this key moment, because we could really use a language today a level higher above that tells us how to assemble fallible components into infallible systems. We could use some linguistic help, because it's through these languages that we learn our design skills about how to do interface design. We could use a Klaus Wirth in those topics, and I think you could do those things admirably and influence yet another generation of designers.

Jochen Ludewig: Thank you very much, Tom. Klaus, if you have an immediate answer you should have the chance, otherwise I pass the microphone on.

Niklaus Wirth: No, I don't really. Thank you for the encouraging words, of course, that's always nice to hear, but I try to be a realist, and I see the work currently going in a different direction, and that's what I wanted to address. Now, if it provokes some people to point out that not the whole world is working in that direction, one of my points has been satisfied, of course. But basically I have become rather a pessimist than an optimist, I must simply confess that. And since the title of this panel is How is it going to be in ten years, maybe I should add one sentence before passing the microphone on. Last year, when I was in Palo Alto again, we were again sitting in front of these darn PC's and had our frustrations, and finally my colleague said: "Listen, Klaus, it's going to be at the very most five years, and then the world will break down." Of course, he meant this PC world with the ever ever growing complexity which is indiscriminately added on, and that will cause too many people to be frustrated like we so-called experts even are. I just got an e-mail from him the other day, and the subject was "Three years and eleven months!" So the count-down is running.

Jochen Ludewig: Thank you. Now, Martin Glinz is the first audience member.

Martin Glinz: I'd like to emphasise a point that Norbert made. I think one of the major sources or even *the* source of our problems in Software Engineering is something that we could call the Peter's Principle of Software Engineering. From year to year we are just developing software that is a bit more complex than we are able to deliver. And thinking about Software Engineering in ten years, the question is: are we able to break this circle? Maybe we can learn from other engineering disciplines, because other engineering disciplines don't. Imagine civil engineers that decide to double every year the complexity and the size of the buildings that they construct. How long would that last until it breaks down? So, when thinking about the future of Software Engineering you can have the pessimistic view in saying "well, in ten years we will be able to solve the problems of today in a systematic way, but we will be unable to solve the problems of ten years in the same way as we are unable to solve today's problems." And the optimistic view would be that we would be able to become real engineers who don't do everything that could be imagined to be done, but do the things that are really engineerable at that time where we are, and not at the time where we could be maybe in five years.

Jochen Ludewig: Well, that's a beautiful question for the panel, but still I would like a very short answer.

Tom DeMarco: Three minutes.

Jochen Ludewig: Less than that. I think you are simply too optimistic about engineers. If you find a sufficiently large number of idiots who are prepared to pay for buildings which are twice as high every year they will build it, definitely.

Norbert Fuchs: But why do they pay for the software?

Jochen Ludewig: Because they want to have it. And apparently they don't mind that it's bad.

Tom DeMarco: I think one of the things that's happened in our world is: Software made some things work more effectively. There was a great deal of successful software built that gave great benefit, that automated the easy things within the organisation, replaced drudgery with automated procedure. And it had benefits far in excess of their costs. And then we went through a period as a software industry where we didn't realise that we had succeeded in a great mass of the work that was there for us to do. We kept ourselves very large and now instead of waiting for somebody to ask us to build a building that's twice as tall, we were in the offering position, saying, well, we've got to keep these people busy, let's build a building that's twice as tall and see if we can thrust it upon people. That was something that I think happened up through the end of the eighties. We have been through a downsizing and a corrective period, and I don't think that's happening now. I think today we are much more in a responsive mode, where we are again waiting for the user community to come us and say: "I see a real benefit here to do x. Could you build me a system that does x?" And we will again be able to build those systems, not orders of magnitude larger, but slightly more complex than what we've built before, that will deliver real benefit at sensible cost. So I think, Martin, that there has been an economic correction in our world that takes us out of the mode where we are trying to thrust software on people and I think that has corrected or will correct much of the problem you identify.

Lucy Suchman: I just want make sure that no-one believes that all Americans are optimists. So I give a counter-scenario to Tom's apropos of this question or comment. Another way I think of understanding it, and that goes back to the comment I made before as well, is that we are in a kind of a vicious circle where there's an intensification of time, of money, that leads to various kinds of intractable, very difficult problems for organisations, for which they are desperate for a very quick fix, and the most obvious quick fix is a technological one. So they are willing to pay large amounts of money for software which they hope will solve these very difficult problems, and the software industry then willingly obliges by trying to produce more and more software in shorter and shorter time of greater and greater complexity, and the whole cycle continues to feed itself. That's an alternative account of why it is that we find ourselves in the situation where for reasons that seem very difficult to understand, customers are willing to pay large amounts of money for software of very uncertain reliability.

Klaus Jeppesen: I would just like to make maybe a real proposal for having a totally different approach what do you have suggested. It comes from the idea expressed earlier that we shouldn't forget that a lot of software is out there and is working actually very well and very fine. So why don't we simply lean back and say let's accept the fact that we can't do this in a perfect way. We should set up a way of living with the fact that software does have mistakes. I know this sounds weird, but let me give you an example: it could be the procedures and attitudes toward things that we should change. If you're looking at the finance department in our company, it is full of people. They are very good, all of them, but they do make mistakes, and over the years they've invented ways to recognise those errors they've made, correct the bookkeeping, and come back to normal, so that things do turn out right at the end.

Or we could maybe change the attitude. If my car breaks down I bring it to the garage, if my computer software breaks down I immediately jump at myself. So maybe we could simply change the approach of how we are dealing with software, and the procedures, to live with the fact that it does have errors.

Jochen Ludewig: Anybody wants to answer that?

Tom DeMarco: I think it would be self serving of me to agree with you because I think you just agreed with what I've said earlier. But I agree. Some years ago, Clyde Woods, a systems engineer at Pacific Telephone, proposed a model in which we stop doing software maintenance entirely. We take pieces that have varied from what we now understand to be the real requirement, in other words they are in need of maintenance, and we wrap an onion skin wrapper around them to modify their behaviour, to make them right without going to the insides. And then, when they subsequently need further maintenance we wrap another skin around them, and so forth, and we grow them in a truly organic way. Now, you can imagine doing this as a total hacker. But you can also imagine that there might be a better way to do that. There may be bad ways to do that and good ways to do that. You can imagine if we still had Harlan Mills around us, asking him to deal with that as a mathematical problem, to deal with something that has veered from what we now understand to be the real requirement, but we're not willing to modify, to wrap something around it using a formalism, and the formalism depended on a formal description of the way that it varied from what was inside. Now I'm not happy with that model, I'm not happy with what either you or I or András has suggested, because I, too, would like to build it right and know that it's right and know that I could depend on it being right. But I think that as time goes by we're going to be thrust more and more into

that model, and we're going to be looking for sensible formalisms to approach what otherwise would be a total hack. Right now, we don't know how to do that in a very formal way; we can only hack it. But I believe we need to learn how to make a very formal business of encapsulating and concealing and correcting defects.

Niklaus Wirth: For once I very much agree with you. We may be forced to live with these things. But be careful, Tom, with this technique you may be on the way to have a very very multi-layered onion, at least in your belief, and then suddenly you wake up and see it's a pumpkin. What I think, if you really are learning something as you claim, then sooner or later we should at some time be able to install and operate systems that incorporate that new knowledge, that better wisdom. Yet we have the existing ones and we depend on them. If there are bankers in this audience, they will very quickly come and say, yes, we absolutely depend on the existing systems; please don't talk about replacing them. So one of the important things that I think computer science or rather computing practice will have to cope with in the next ten years is how to gracefully replace old hacks with new clean systems. To be sure, the new clean systems will also have their problems but hopefully in new ways. And that is a difficult problem, I don't know how to tackle it. It has to be tackled by people in industry, who live with such big systems. You can't exercise that on small problems; it's just not scaleable. So that, I think, is a problem that I would like to see solved, that will have to be solved in the next ten years.

Jochen Ludewig: I think I'm also a bit concerned that if every time we find a dead horse in the road we put a new layer of concrete on top of that.

András Pataricza: In the meantime I was a little bit thinking of what would happen if a civil engineer used Software Engineering Methods. First: what is the problem with software? That we do not have any tolerances! That means a software engineer would require a laser interferometer to measure the breaks. Furthermore, there would be absolutely no guarantee that if the last stone in the kitchen floor would not fit, that in this case it is possible that the entire house crashes. That is our problem. But on the other hand, I personally as a student for a while I learned a lot of Pascal and those major concepts. Why are civil engineers successful? Because they have a clear structure and that structure gives the house a robustness, and that is something which resembles those concepts which were evaluated during the last three decades, among others, by Niklaus Wirth. That means that we have to have such implementation technologies that those elegant ways of thinking and our poor and imperfect implementation can be combined.

Jochen Ludewig: Thank you very much, András. Well, ladies and gentlemen, the time is now almost five minutes to six and I think we can't start a new discussion now. So I would like for my part to finish this session. Thanks again to all the speakers. I'm pretty sure, in spite of what Klaus said, that there are five people sitting here who do a pretty good job on Software Quality Engineering, and I think they will continue to do so. Thank you very much for your attention, you have been a great audience for me and probably for the speakers as well. And now, Karol Frühauf, your turn.