Design, Composition and Performance

- **Speaker: Rich Hickey**
- **Conference: QCon San Francisco 2013 - Nov 2013**
- **Video: http://www.infoq.com/presentations/design-composition-performance-keynote**

[Time 0:00:00]

```
slide title: Design, Composition, and Performance

Rich Hickey
```

Hi. Thanks for coming. I am very excited about this conference. It is always great, and I would like to thank the organizers for inviting me today to talk about design, composition and performance.

[Time 0:00:07]

```
slide:

Analogies are like equivalencies, except when they are not
```

So we start with a legal disclaimer prepared by lawyers.

We are going to have some fun with analogies today and the cool thing about analogies is, they are as much fun when they are wrong as they are when they are right.

[Time 0:00:32]

```
slide title: Design

'to prepare the plans for (a work to be executed), especially to plan
the form and structure of'

--'decide upon the look of'-- [strike-through of that definition]

_designare_ - to mark out

make a plan, write it down
```

So, design is something we talk about a lot in software development.

But I think it is something that is somewhat beleaguered these days, not because people do not do it, but because I think people are in a hurry and they are trying to get things done. And I often get developers asking: I would like to be able to work at the next level and talk about the design of things before I just code up solutions.

So what does it mean to design something? And as you all know, all I do to prepare these talks is go to dictionary.com and look stuff up. So I looked up design, and one of the definitions is this, which is really great. Definitions are always great. It says, "To prepare the plans for a work to be executed, especially to plan the form and structure of that work."

And I think that is super important, the notion of executed. It means that somebody is going to do this. We also have a different notion of executing things in software, which is that this is going to run, which is also interesting.

There is another definition, which is to decide what the look of something is. And there is nothing wrong with that kind of design. I just want to say this talk is not about that at all. Never when I say design do I have this meaning in mind.

And, of course, we go to the roots and see that the root is in to mark stuff out, and that matters. So there are really sort of two things here: We want to make a plan, and we want to write something down.

[Time 0:01:59]

```
slide title: But...

+ We already write down code
  + do we still need designs?
+ Generate docs from implementation?
  + not a plan
+ Monolithic designs, ugh, been there
  + plans, not good ones
```

And, of course, here is where everybody is like, ugh, you know, we did this. We did this in the '80s, and we had all this stuff, and it was terrible. We had these phonebook-sized specifications and nothing got done, and it was all awful.

So we already write down code. Isn't that enough? Do we still need designs if we do this? And the answer is: yeah, because that is not a plan. That is just what you did.

Can we generate docs from implementation? The same thing. That is not representing a plan to do something. It is like I already did it, and it and so somebody asked me for a documentation or a design, and so I pulled the lever and this came out.

And of course there is this complaint. I do not want to do this. These things are big. They are top down. They are waterfall model, etc. etc. We did that already. And it is true that happened, and they were plans, but they were not good plans.

[Time 0:02:57]

```
slide title: Good Design

+ _Separating into things that can be composed_
+ Each component should be 'about' one or a few things
+ Composing them to solve a problem
+ Iterative
```

And so what I want to talk about today is a little bit about what do we want out of a design, and when we encounter in the world, what do we see?

So I think that a simple idea behind design is to look at it in terms of taking things apart. This is the opposite notion we typically have. Typically people think about design, and they say: design is making this big, involved plan that is going to solve every issue that the system is supposed to address.

But I think that is not what you do when you are trying to get a design that is going to survive and live over time. I think, instead, what you want to do is break things apart in such a way that they can

be put back together. And that is fundamentally what design is about: taking things apart so you can put them back together, because obviously taking things apart and walking away is not really going to help.

The other thing you find in good designs is that they are always about one or very few things. Designs that survive, designs that really foster reuse are about a single thing, generally.

And then you put them together. So the first thing you do is you take everything apart. Then you compose them, and then you are solving your problem. But the first thing is the taking apart.

And there is nothing about this that is in conflict with iterative methods for developing software. This can be an iterative process. And all that happens then is that you get feedback during development to your design.

[Time 0:04:18]

```
slide title: Design is taking things apart

+ requirements
+ time / order / flow
+ place / participants
+ information / mechanism
+ solutions
```

So, what kinds of things would we take apart? There are a whole bunch of things that we could take apart and, in fact, you are constantly finding more things.

You can take apart the requirements for a system. You can take apart the order in which things happen, who is going to talk to whom, information parts of your system for mechanism parts, and you can actually take apart different solutions to assess their merit.

[Time 0:04:43]

```
slide title: Take apart requirements

+ Move from want / need
  + to _problems_
+ knowns / unknowns
+ domain-side / solution-side
+ cause / symptom
+ unstated
```

So let us just look at each of these in turn.

Taking apart requirements: This is something we do not do often enough. Somebody says: I want a system that does X. I need Y. You know, it has got to do Z. We get these feature lists.

And the first thing that we should do when we are handed that is to break them apart and try to find, in the set of requirements, in the set of features or desired things or needs, the actual problems. And it is only by doing that that you can start to move forward and say, OK, I have broken your need into problems that you have, and then we can try to make things that solve those problems.

The other thing you are going to do initially with requirements is: divide them up. The simple way to take them apart is to say: these are things I know how to do and these are things I do not know how to do, so your knowns from your unknown.

You are going to take apart requirements that are domain side – the system must do this to satisfy this business thing – from solution side – things like: we need to run on AWS, or something like that.

Often you get requirements, especially for systems that already exist, that are about: it is not working. And everybody has heard that. How do you fix it? It is not working.

The first thing you have to do when you are trying to fix what is not working is to separate out what is the cause of this problem from what is the symptom of this problem. Somebody says: my screen is black. I would say, OK, I know how to fix black screens and start typing because there is not a generic solution to the black screen problem yet.

And then there are a whole bunch of requirements that are unstated, and these need to always be enumerated, if not taking it apart, but they need to be in mind at all times. Unstated requirements are the things that everybody wants the system to avoid, like I would like a system that does not keep crashing, or use up all the memory, or cost too much to run, or use too much energy, or require a lot of manual effort, or the users will hate. And so the unstated requirements are often a set of things that your software is supposed to not do, not cause attributes it is not supposed to have, so we want those on the table.

[Time 0:06:42]

```
slide title: Take apart time / order / flow

+ queues
+ idempotency
+ commutation
+ transactions
```

Other things, just completely different dimension of things we can take apart when we do design, which is time. You can take apart the order of things, how things are going to flow from one to the other.

You can break systems apart, so there is less direct calling. You can use queues to do that. You can support redundant activity with idempotent approaches.

Commutation is a very important concept that is going to be more and more prevalent as we try to build systems that are highly distributed. Which says: I can make a system order-independent by supporting operations that are all commutative. Then I do not care how things come in. So it is a technique for breaking apart. I used to have this order dependency, and now I do not. So now I have two separate things I can talk about independently.

And transactions are the opposite when you say: I do need to know these things are going to happen together.

[Time 0:07:34]

```
slide title: Take apart place, participants

+ "All problems in computer science can be solved by another level
  of indirection"
+ Also authors
  + independent development etc
```

We can take apart place and participants. And there is a certain sense in which design is always about this. But there is this old adage: you just add indirection.

But here we are talking about possibly the whole process of building something. Having a design is a thing that lets two teams work independently, or people work in two independent languages. Taking apart things is what facilitates the participants, the authors, as well as the participants, for instance the systems. By breaking things apart you are able to say, well, run this on this machine here, or run this in this tier, or we will put that on the Web.

[Time 0:08:12]

```
slide title: Information vs Mechanism

+ Set of logged-in users
+ Set collection class / construct
```

This one is kind of interesting, because I do not see it talked about often enough, which is to separate information versus mechanism. So there is always information that our system manipulates. For instance, your system may have the notion of the set of users who are logged in. That is an idea. That enumerated set is a piece of information.

And then you have, you use the set class, a collection class from your favorite framework library to put the logged in users.

And one of the problems I think we have in software development is: we use the same stuff for both of these things, but these are two very, very different things.

One is sort of this device into which you stick stuff and you can go back later, and it is kind of a little bit of a place.

And the other is a piece of information, which you should really not treat that way at all. So pulling these things apart, talking about your system and clearly differentiating the stuff that is information from the stuff that is sort of the mechanics of your program is quite critical.

[Time 0:09:15]

```
slide title: Take apart solutions

+ Benefits
+ Tradeoffs
+ Costs
+ Problem fit
```

And then, finally, once we think we have an answer, we have a potential solution, we have not implemented it yet, but we are looking at it, or maybe we have implemented some of it. You want to take those apart to see not just the benefits. Those are pretty evident usually.

But also the tradeoffs: What part of this is not going to work? How much is it going to cost to run? And does it eventually fit the problem? Because a lot of times what can happen is: you can adopt a solution that is larger than your problem. And then what do you have? You have two problems, right? You have your problem, and now you have this thing that was too big for it.

So it is not just about getting answers. It is about breaking things apart in a coherent way.

[Time 0:09:57]

```
slide title: Why Design?
```

+ Comprehension
+ Coordination
+ Extension
+ Reuse
+ Testing
+ _Efficiency_

So I am a big fan of design. I think that we need to do a lot more of it, and we need to talk about it more, and we need to spend more time on it. But I think it is pretty easy to rationalize why we need it.

The first is: so that we can understand the system. A design is hopefully smaller than the code that implements it, and so it is easier to get our head around what it is about.

The other thing, as I was talking about, is design is fundamental to coordination. If you do not have some plan, you cannot just send two people off to write. You write one-half a system; you write another half of the system. And that is the end of the conversation. What is going to happen? Well, they are going to wonder which half of the system they are supposed to write, because there is no plan. So there is no way to have coordination and to have multiple groups working on something without a design.

Design also facilitates extension and extensibility. People are always like, oh, I want to make something extensible. But the easiest way to make something extensible is this breaking it apart thing. Because when you have broken it apart, you end up with two separate things. You end up with pieces that are meant to connect to other pieces, which means there will be connecting points on those pieces. And therefore, when you want to do something new, you can make a new extension, and it can leverage that connecting point, because it had to be in place because the things were separate.

The flipside of that is this reuse aspect, which is: when you have broken stuff up into separate pieces that have nice interconnecting points, you can pull them out of one context and put them in another context. And that is how you get reuse. These are not like magical things, and they are not attributes of APIs, necessarily. They mostly fall out of this decomposition.

Finally, testing is greatly facilitated by design. Ideal testing takes some design constraints, some specification, and turns it into tests, as opposed to sort of embodying design inside tests. That is inside out. But again, that is something we have to work more at. Systems like QuickCheck are interesting because you are basically starting with propositions about your system, which reflect the design, and saying, "you write the tests, computer".

[Time 0:12:02]

And, finally, I think the thing that is often most readily pulled out as an argument against design is: I do not have time. This is going to slow us down.

And in fact, I think it is the opposite. In particular, there are all these adages about: when is it easiest and least expensive to fix a bug? Not out in the field. If you have already shipped it, it is the most expensive. People are like, oh, we should fix it in QA, or we should fix it in our code and do test-driven design.

But the thing is, you can keep moving back. It is easiest to fix your problems in OmniGraffle. You just say, ooh, that is not going to work, and you move some boxes around, and it is fixed. It is much cheaper than fixing software.

But even after you have shipped, I think that there is a lot more efficiency in systems that have been designed, because you are going to be able to go back to something. And usually the answer to your problem in the field is: I have just insufficiently broken something down, and so the solution I am

going to need is just breaking it down more. And that is less expensive than: I created this giant ball of everything and I need to untangle it. So I do think, in the end, it is more efficient.

So that is design.

[Time 0:13:22]

`slide:`

`Composition and Performance`

The talk is about design, composition, and performance, and so I am going to take composition and performance together.

[Time 0:13:28]

`slide:`

`[ Photos of Béla Bartók and John Coltraine ]`

Figure 1: 00.13.28 Bartok and Coltrane

And one of the beautiful things about dictionaries is: there is more than one meaning for each word. And so there is more than one meaning for composition, which of course we think about composing

systems out of pieces like I was just describing, and we think about performance in systems usually as: how fast do they run?

But when I think about composition, I often thing about Bartok. And when I think about performance, I often thing about Coltrane. And so these are two musicians. Now, Bartok is a Hungarian composer, but he was also a performer. He was a pianist and taught piano.

And Coltrane is a famous saxophonist and great performer, but was also a composer, so I am not trying to pigeonhole these guys. But we are going to use Bartok to stand in for the composer and Coltrane to stand in for the performer, and talk about two different notions of composition and performance, and maybe how they might inform software.

[Time 0:14:20]

```
slide title: Composition

+ Self-imposed problems / constraints
  + like other art forms
+ Design for performers
  + ditto screenwriting, choreography ...
+ Organization challenge
  + a plane or design addressing those challenges
```

So composition – music composition and other kinds of art creation – is about addressing constraints. It is about addressing problems, but not real world problems. Art does not solve real world problems, in general. And if it does, it is more than art. It is something else.

So it is quite interesting that the first thing that composers tend to do when they have a blank page – they can do whatever they want – is make up a bunch of problems for themselves. They actually create a bunch of self-imposed constraints. And that is true of all the other art forms. In general, you are going to see this.

And composition is design for performance. So we saw that definition of design on the first slide. And it said, "to be executed". And that is what composition is. You are writing something. You are anticipating someone is going to perform it or do it later.

And it is the same thing. Screenwriters presume people are going to act it out. Choreographers presume somebody is going to dance it. So it is design. You are solving constrained problems by creating your own constraints, and you are designing that something to be executed. So it is very much a design problem.

And it is an organizational challenge. You are trying to address these constraints that you have set up for yourself, and that is what composition is.

[Time 0:15:40]

```
slide title: Specificity and Scale

+ Fully orchestrated / arranged
  + typical at larger scales
+ Melody + changes
  + increased latitude for performers
  + increased responsibility
```

And it is quite interesting that, when you look at music composition, you end up immediately seeing a tremendous variety in the specificity of compositions and the scale of them. And it is telling that software sort of straddles these two things.

The first is: you see fully orchestrated music. It is fully arranged. All the notes are written out for every part. This is typical at a larger scale, so bigger compositions, orchestral compositions, and operas and things like that tend to have full orchestration.

And then smaller compositions, you might have only a melody written out and the chord changes for, say, a song. We will call it a song, but we are going to not talk about words today. And in those compositions, you have a lot more latitude for performers, because you are not saying, "you must play this note or this register on this instrument at this time, this loud". You just said, "this is the melody and have at it". So there is more responsibility for performers. So there is this whole spectrum.

I think that when people push back against design, they are afraid of this first one because, again, back in the '80s we had this stuff, and people had plans that you would draw pictures and push buttons, and it would write programs. Maybe people still have those plans, but they are conducting them in secret.

But I think programmers are like, "don't repress me, man. I do not want to see this big thing". But I think that, again, you are going to have the spectrum. You are going to need a lot more writing down, especially if you are going to share amongst people. And then, in the small, when you are talking about your own individual effort, maybe you do not fully annotate the same way.

[Time 0:17:19]

```
slide:

[ Image of sheet music labeled "Concerto for Orchestra" on the left,
and for "My Favorite Things" on the right. ]
```

So we can see two pictures of this here. I do not expect anybody be to able to read that, but on the left is the "Concerto for Orchestra". It is a Bartok piece. And there are the parts are all written out for the strings, the percussion, and the winds. And it is all specified.

Although it really is not, though. I do not know if you can tell, but there are red markings and some other things on here, which were notes taken by who? The conductor, because the conductors says - or, "I do not know what to do here. You did not tell me exactly, exactly, exactly what to do, so I have to decide what the tempo is," or how to balance these two sections against each other. But, in general, that is pretty fully specified.

On the right, we have "My Favorite Things", as it would appear in like a jazz real book. And this is the tune from Rodgers and Hart "The Sound of Music". And this is all you would get if you are a jazz musician. Here are the changes, and here is the melody, and you move from there.

[A jazz real book: https://en.wikipedia.org/wiki/Real_Book

The musical "The Sound of Music" has music written by Richard Rodgers with lyrics by Oscar Hammerstein II: https://en.wikipedia.org/wiki/The_Sound_of_Music]

[Time 0:18:18]

```
slide title: Constraints

+ Most compositions are 'about' one or a few things
  + melodic, harmonic, rhythmic, timbral ideas
```

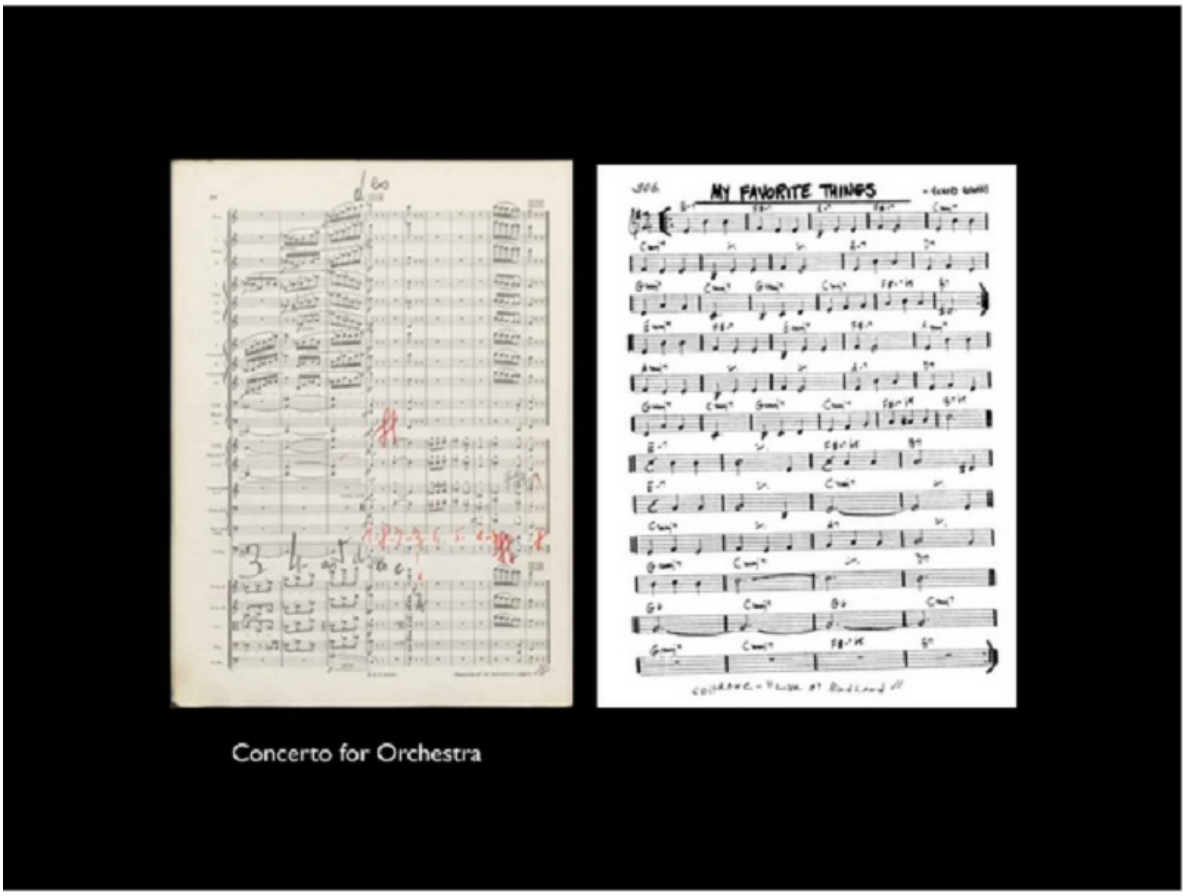

Figure 2: 00.17.19 Concerto for Orchestra

```
+ motif or theme
  + variations
  + resolution
+ Larger works, more structural components
```

So I talked a little bit about constraints. And, again, it is quite interesting to see how this lines up.

So most compositions are about one or a few things. The same kind of thing. You are setting out. You are saying: what is this piece going to be? You rarely say: oh, I will just use these notes for a while, and then those notes for a while, and then those notes for a while, and then call it done. Never. You never do that.

Composers come up with little motifs and things that they are going to reuse or transform and sort of riff on as time goes by. So there are all these ideas that you are going to set up as boxes in which you are going to work. And you have variations of those things, and you will do resolution.

And then the scale of the composition really just determines how many of these things that you have, and maybe how many different levels there are.

So a big Bartok composition is going to have very, very fine-grained constraints about melodic motifs in a very particular part of the piece, and then, at higher levels of structure, deal with big form kinds of decisions. But again, they are self-imposed constraints.

[Time 0:19:27]

```
slide title: Improvisation

+ _improvisus_ - not foreseen / provided
+ Melody + changes provide constraints
  + Performer provides variations
+ _Tremendous preparation, practice, study_
+ Deep musical knowledge and vocabulary
```

When we move to the performer side of the coin – the improvisation side like Coltrane – I think it is quite interesting. Again, it is an interesting word. It means "not foreseen or not provided". And "not provided" means you did not have the answer up front before you went and did it. Like you were not handed a complete plan before you went.

And so in the case of a jazz performer, you are going to have melody and changes. Then you are going to go and provide variations, make something up.

But I think that people have a tremendous lack of understanding of what goes behind improvisation. For instance, a lot of people think Coltrane is just this genius who is spontaneously emoting. They think that improvisation in music is just making stuff up off the top of your head. It is just amazing. It is like hacking. Just, I am so awesome. I am so bright. I am just going to make this up.

But it has been quite interesting to see, as we have gone back through the archives and had these new releases of old recordings where they put the alternate takes in there. Because you will see Coltrane. He had this solo. It sounds incredibly spontaneous. But then you listen to the other six versions, and you realize that everything that went into the solo that you thought was this amazing one-off, he had worked out. And he was trying them in different orders, different juxtapositions, different cadences and levels, and maybe the order of it was spontaneous, but there was a tremendous amount of preparation associated with that.

So there is a sense in which improvisation is dynamic composition of prepared materials, of planned material. And that to be a great improviser means to make those smaller plans, or have those kinds

of prepared abilities or approaches or sensibilities that you can apply when the time comes in a live situation.

And you have to have a lot of knowledge to do this and a lot of vocabulary to do it. It is just not something that you make up. And Coltrane was a genius at this preparing. He practiced more than anyone in order to seem as if he was making it up most fluently.

[Time 0:21:48]

```
slide title: Harmony

+ 'accord, congruity'
+ 'simultaneous combination (of tones)'
+ 'the art or science concerned with the structure
  and combinations (of chords)'
+ _Harmonic sensibility is a key design skill_
```

So another thing that sort of crosses the lines in composition and performance in music is this notion of harmony. And again, we get this nice word for it, which is accord or congruity. How do things line up?

Again, there is this lining up notion in the simultaneity associated with harmony. So we have melody is sequential, and harmony is parallel. So music did all this before we had computers.

And so this is, how do things work together at the same time? If I play these three notes at the same time, what will happen? Or for Coltrane, if I played this note while these chord changes or this set of notes while these chord changes are happening, what will that be like? Bartok had to imagine, when the strings are doing this and the winds are doing that, what will it sound like all together?

There is also sort of a mathematics of harmony, which is the science behind it or the way you study the rules, if you will, of harmony.

And I am going to contend that harmonic sensibility is a super critical design skill. This is the thing that you want to nurture in yourself. And it may be a little bit hard to see how the mapping works from music to software, but it is fundamentally what a good designer has. They know if they make this choice in this context, that is going to go together, and those two things are going to work well together. And they know that because of their experience and the study that they have done of working systems.

[Time 0:23:17]

```
slide title: Bartók and Coltraine

+ Masters of harmony
+ _Students of harmoniousness_
  + Beyond the rules
+ New systems that preserve / explore harmonic essence
+ Towering intellectual effort, while totally rocking
```

So I think both Bartok and Coltrane are interesting, even though they are in completely different genres of music, in that they were both masters of harmony. If nothing else, you can say the two are similar because they totally mastered harmony.

And in fact, what was interesting about both of them was that they were students of harmoniousness, if you will. That the thing I think that they were most interested in was: what makes things work together well?

Bartok studied obviously the classical tradition, but his music was not compliant with those rules, and it is because he brought a whole bunch of influences in from studies he had done of folk music of Hungary. And what he studied in that music was the sonority that was possible in these tunes that did not follow the classical rules, but they still worked. And so he pulled out what worked about that, and wrote pieces that are hard to really recognize as being completely tonal, but they are tonal, and they are satisfyingly consonant as tonal music is, which is quite, quite astounding.

Similarly, Coltrane invented whole new ways of doing reharmonization over chord changes that had that same sensibility about harmony.

So I think that what was cool about both these guys is that they both sort of developed new systems that preserved what was essential about things being harmonic or being consonant.

And then the other thing that is quite interesting is that, on both halves, whether you listen to a Coltrane improvisation or the most beautiful, engaging piece of Bartok, what is behind this is a tremendous amount of intellectual effort and activity. I mean you can listen to this Bartok piece and be stunned by it, just blown away by the emotional content. Then you go study the score, and there are all these Fibonacci numbers and ratios in it. And you are like: oh, my God! This was the constraints he set for himself before he wrote this thing that seemed, or was, so emotionally powerful. So there is a lot to appreciate in both of them.

[Time 0:25:26]

```
slide title: What does this have to do with our Langs and Libs?

[ Image of Clojure logo ]

+ Is Clojure like a song?
+ Is it like a symphony?
+ More like an instrument
```

But what does this have to do with anything that we do? In particular, what does it have to do with languages and libraries? Which is really what I want to talk about today: languages and libraries.

Is a language like Clojure – or any other language. It does not matter. This is not really about Clojure. Is it like a song? Are languages like songs? Are they like small compositions? Are they like big compositions?

I do not think so. I think that languages and tools, to me – if you are going to map this analogy – are more like instruments.

[Time 0:26:00]

```
slide title: Instruments

[ photo of a guitar ]
```

So let us talk about instruments. That happens to be one of my favorites. I have that one.

[Time 0:26:06]

```
slide title: Excitation

+ Most instruments are 'about' one thing
+ Pluck, vibrate, strike...
```

Figure 3: 00.26.00 Instruments

[Photo of fingers plucking strings, mouth blowing on a reed instrument, piano mallets striking strings.]

Figure 4: 00.26.06 Excitation

Again, instruments are sort of their own design problem. Instruments start with something called excitation.

And there is a sense in which most instruments are about one thing. You pluck a string. You cause vibration on a reed by blowing on it. You strike strings with the mallets of a piano, or you hit drums, or things like that.

And what is quite interesting is that very few instruments are about more than one kind of excitation. Most instruments are about one kind of excitation. It is quite rare to see the other.

[Time 0:26:44]

slide title: Control / Interface

[Photo of keys of a saxophone, keys of a piano, strings and fret board of a guitar, foot pedals of a piano.]

Figure 5: 00.26.44 Control/Interface

Then this is combined with some sort of control, or interface, or technology on instruments and saying: then there is an interface. So there is excitation. Then there is this interface for people to go and shape the excitation.

[Time 0:26:55]

```
slide title: Projection

[ Photo of the flared bell of a trumpet, the open lid of a grand
piano, and the hole in an acoustic guitar body. ]
```

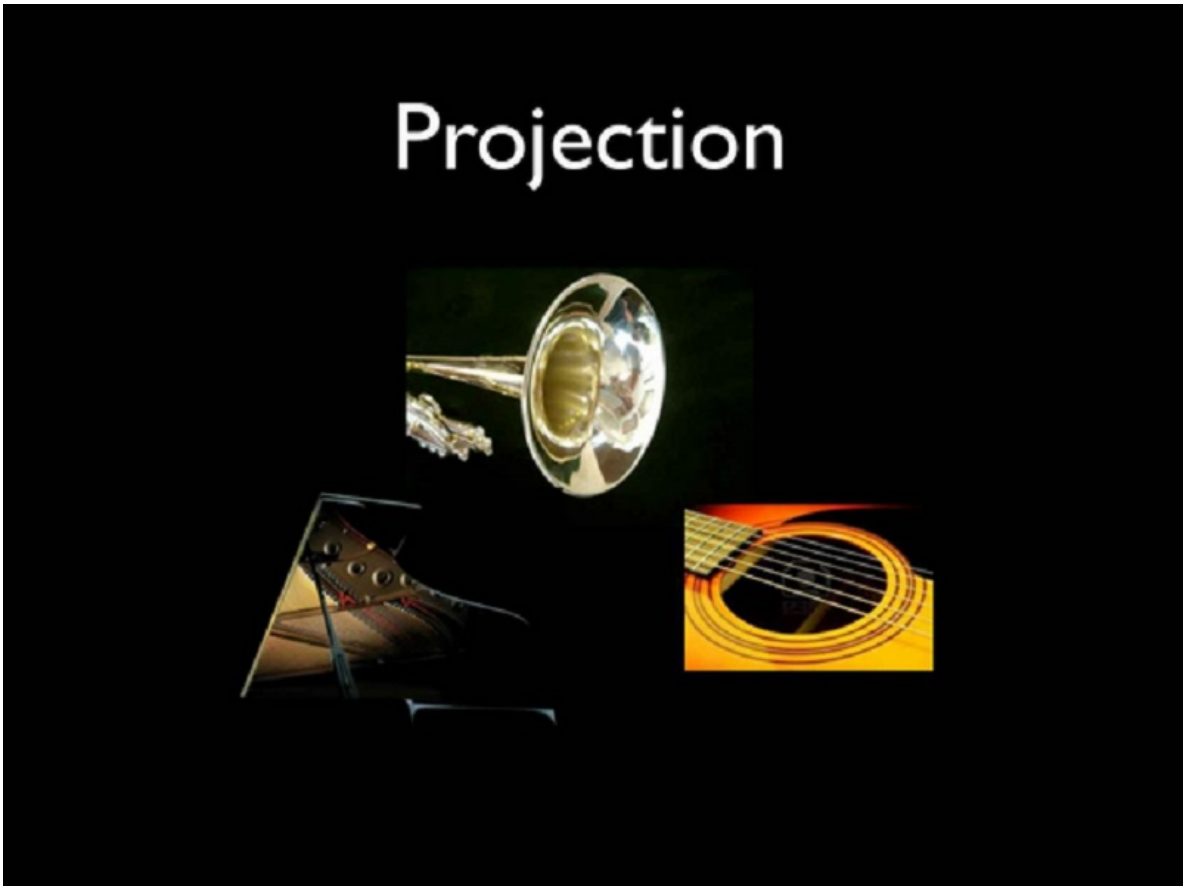

Figure 6: 00.26.55 Projection

And, finally, there is an aspect of an instrument, which is sort of its fundamental goal in the world, which is to take that excitation and direct it at a problem. And the problem for most instruments is: how is somebody going to hear this? How do we get the sound across the room so somebody can pick it up?

And so instruments are about directing the force or energy of the excitation out to the audience. They are directed at an outcome. So there is a little piece of design work associated with an instrument.

[Time 0:27:22]

```
slide title: Resonance
```

[Photo of violin.]

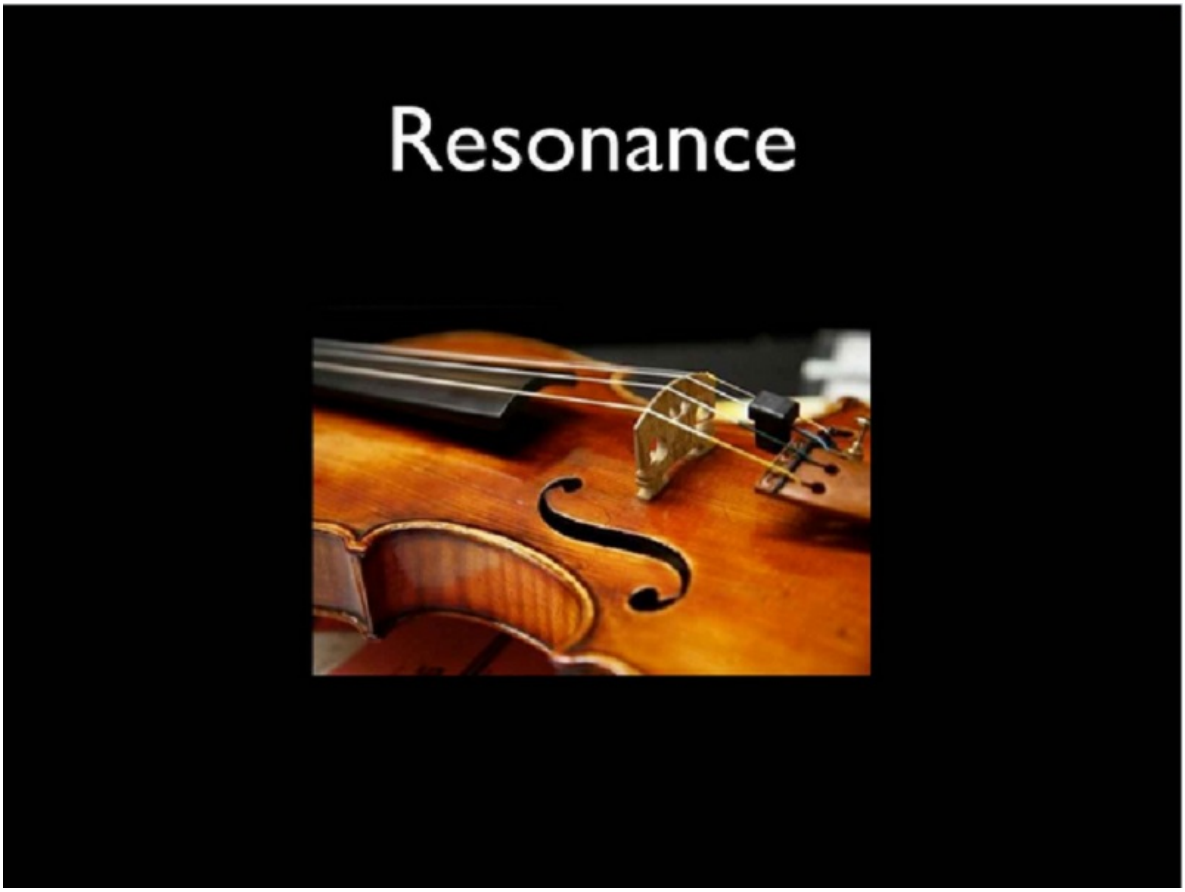

Figure 7: 00.27.22 Resonance

Instruments also have this other interesting aspect, which is resonance. When you design an instrument, especially something like a violin, or a guitar, or anything that has a vibrating body to it, the body itself is going to interact with the excitation. So the excitation of the string is going to vibrate whatever, and the body is going to go and say, woo, that is - I like that. I am going to amplify that. And it will amplify some things more than other things.

So there is a design problem, and there is a harmony problem to the physics of an instrument to say, well if I build an instrument whose body resonates at a frequency that is not a harmonic relationship to the strings themselves, it is going to sound awful. And it is actually a physics problem to get that harmony right in the wood.

[Time 0:28:09]

```
slide title: Instruments are limited

+ Piano
  + no in-between notes
+ Sax et al
```

```
  + one note at a time
+ Minimal, yet sufficient
  + no missing notes

[ Photo of blues harmonica. ]

            DSL?
```

But instruments have a lot of other characteristics, and one of them that is quite striking is that instruments are limited. They are *very* limited.

Piano - cannot play any in-between notes. It can only play specific notes: the 12th root of 2 all the way across or maybe you stretch it a little bit, but there is no in between notes.

Saxophone can only play one note at a time. This is awful.

[Audience laughter]

I mean, and these things have been around for hundreds of years. I mean they did not have GitHub, but somebody should issue a pull request.

[Audience laughter]

And fix this! Right?

But there is a sense in which they are minimal, yet sufficient. For instance, most instruments do not have any missing notes. For whatever range they cover, they have all the notes. At least we are talking about Western instruments and Western scales. But they will tend to have all the notes.

But, not all of them will. Blues harmonica does not have all the notes.

There is a kind of musical DSL, right? It is like: you do not need all the notes. You are just a businessperson. I can give you just the blue notes. That is all you get.

And so, is this something to fix? There are all kinds of limits, not just in the notes they can play, but the registers they can play and things like that. Why haven't these all been fixed? Why cannot every instrument do everything?

And there is a sense in which the players can overcome this. How many people here play piano? So what do you do to deal with the fact that a piano cannot play the in between note? What do you have? You have grace notes and trills and mordents and stuff that give you all that sort of feel around the note thing.

John Coltrane famously became so adept at the saxophone, and he had such physical prowess and muscle memory, and combined it with this gargantuan knowledge of harmony, that he could play these scales so fast that he could imply not only chords, but entire tonalities – superimpose entire tonalities – over chord changes by just playing sheets of sound, is what they called it, over music.

So it is not necessarily the case that the shortcomings of these things need to be fixed in the instrument. There may need to be room for the performer to do it.

[Time 0:30:31]

```
slide:

No one wants to play a choose-a-phone
```

And there is another good reason why we do not fix everything, which is that no one wants to play choose-a-phone. No one wants to play an instrument that does everything. You could push here, and it makes a piano sound, and then it makes a drum sound, and then this happens and that happens.

[Time 0:30:45]

`slide:`

`[ Photo of Keith Emerson, described further below. ]`

Figure 8: 00.30.45 -image-

So some people do want to play a choose-a-phone.

[Audience laughter]

This is Keith Emerson sitting in front of a Moog modular synthesizer back in the day, and that was just, wow! You could make it do anything if you plugged in the wires the right way.

So I will take a step back and say maybe some people do want to play choose-a-phone,

[Time 0:31:05]

`slide:`

`No one wants to compose for choose-a-phone ensemble`

But no one, I bet, wants to compose for a choose-a-phone ensemble.

[Time 0:31:10]

```
slide title: Complex target

[ Two photos of choose-a-phones, showing complex wire connection
patterns. ]
```

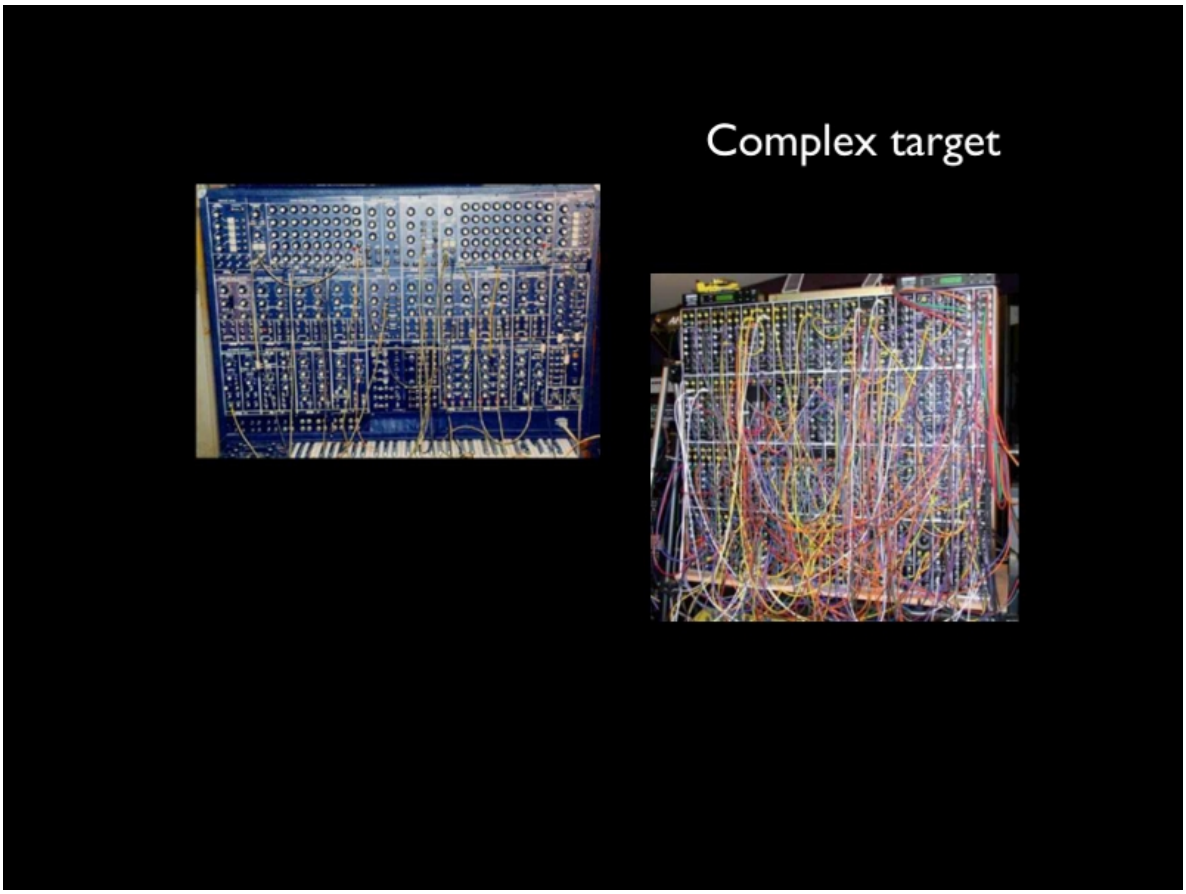

Figure 9: 00.31.10 Complex target

So just imagine that you are sitting in front of an orchestra, and everybody in the orchestra had one of these in front of them. And they put the wires in and whatever. And you are the conductor, and you went like this [raising hands up in the air], and when you say go, what is going to happen? You have no idea. You have no idea of what even could possibly happen.

If you are sitting in front of an orchestra, there is a certain category of things that you think might possibly happen, but you can kind of get your head around what that might be.

And so the problem here is that where you to try to build a bigger system out of something with as much, let us say, parameterization, as these synthesizers, you would end up you are trying to target something that is complex and build something bigger still. That is a recipe for disaster.

[Time 0:31:57]

slide title: Complex target

[Another photo appears, similar to the other two.]

Nested design problem

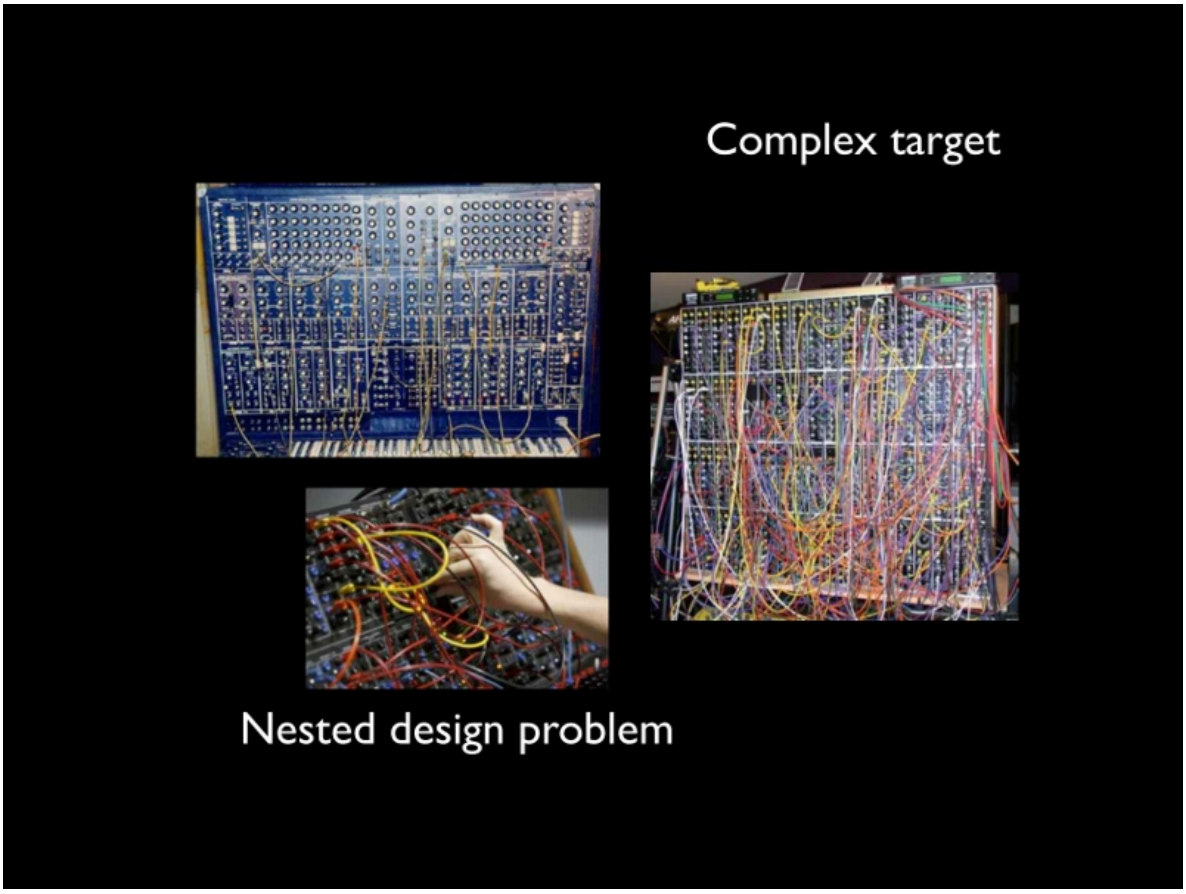

Figure 10: 00.31.57 Complex target - build slide

And there is a sense in which this is just the wrong way to go about things, because you have got this design problem that is actually multilevel, and it is nested. What happens when you say go? Well, it is the sum of what happens for each person.

What happens for each person? Well, it depends on where they put the wires. And what happens, what determines what happens when you put the wires. Well, each module has a different thing that it does. It may be a filter. It may be a sound generator, something like that.

So each . . . There is a level. There is a set of levels at which there must be design. I must design the modules. I must design the sound, the patch that hooks them together, and then maybe I would try to take on a piece with all of this.

But unless there was a way to talk about one of those arrangements and get your head around what it implied, you could never build up higher.

[Time 0:32:43]

```
slide title: Instruments are for players

[ Photo of a man playing an electric guitar. ]
```

Figure 11: 00.32.43 Instruments are for players

So another stunning thing about instruments – which is just, again, it is astounding that the world has continued – is that instruments are made for people who can play them, who can already play them. I don't know. Hasn't everybody heard of, like, "Explain it to me like I am five," or whatever? We are not supposed to do this anymore. We are supposed to make everything for beginners.

But instrument makers do not do that. They do not make anything for beginners. They make everything for experienced players. Instruments are made for people who can play them – 100% of the time.

[Time 0:33:19]

```
slide title: Beginners are not yet players

+ Should cellos ...
  + auto-tune?
  + have green / red lights when you are in / out of tune?
  + not make any sound until you play the entire piece correctly?
```

We have this problem, right? Beginners are not players yet. This is going to cause the world to stop. If you cannot have a website with three buttons on it and everything that possibly could happen can happen, we are done. So we should fix this, right? Because we are technologists, we know how to do this.

We start with the cello. Should we make cellos that auto tune? Like, no matter where you put your finger, it is just going to play something good, play a good note.

[Audience laughter]

Like, you are good. We will just fix that.

Should we have cellos with, like, red and green lights? Like, if you are playing the wrong note, it is red. You slide around, and it is green. You are like, great! I am good. I am playing the right song.

Or maybe we should have cellos that do not make any sound at all. Until you get it right, there is nothing.

[Audience laughter]

And then - then you get it. So, I mean, do we need to fix this?

[Time 0:34:13]

```
slide:

[ Photo of many children playing cellos. ]
```

Here we go. We have a bunch of children, young children, being subjected to cellos.

[Audience laughter]

There is nothing different about these cellos. These are regular cellos, and they are all sitting there. They are out of tune, it hurts their hands, and it is just awful. I think somebody took off, took away their shoes until they get it right.

[Audience laughter]

This is terrible! But it is what happens, because what would happen if they had any of those other things that I just talked about? Who could ever learn to play cello? No one. No one would *ever* learn to play cello.

There is this great article in the current issue of The Atlantic about sort of the tradeoffs, let us say, not the perils, but the tradeoffs involved in automation. And it has got a great line in it, which is that learning requires inefficiency. And it is quite important. And when I read it and was thinking about this talk, I felt like, wow, that is . . . it does go together.

[Time 0:35:10]

```
slide title: Players wanted

<rant>

</rant>

[ Three photos of blisters on the fingers of players, labeled
"Guitar", "Harp", "Double bass". ]
```

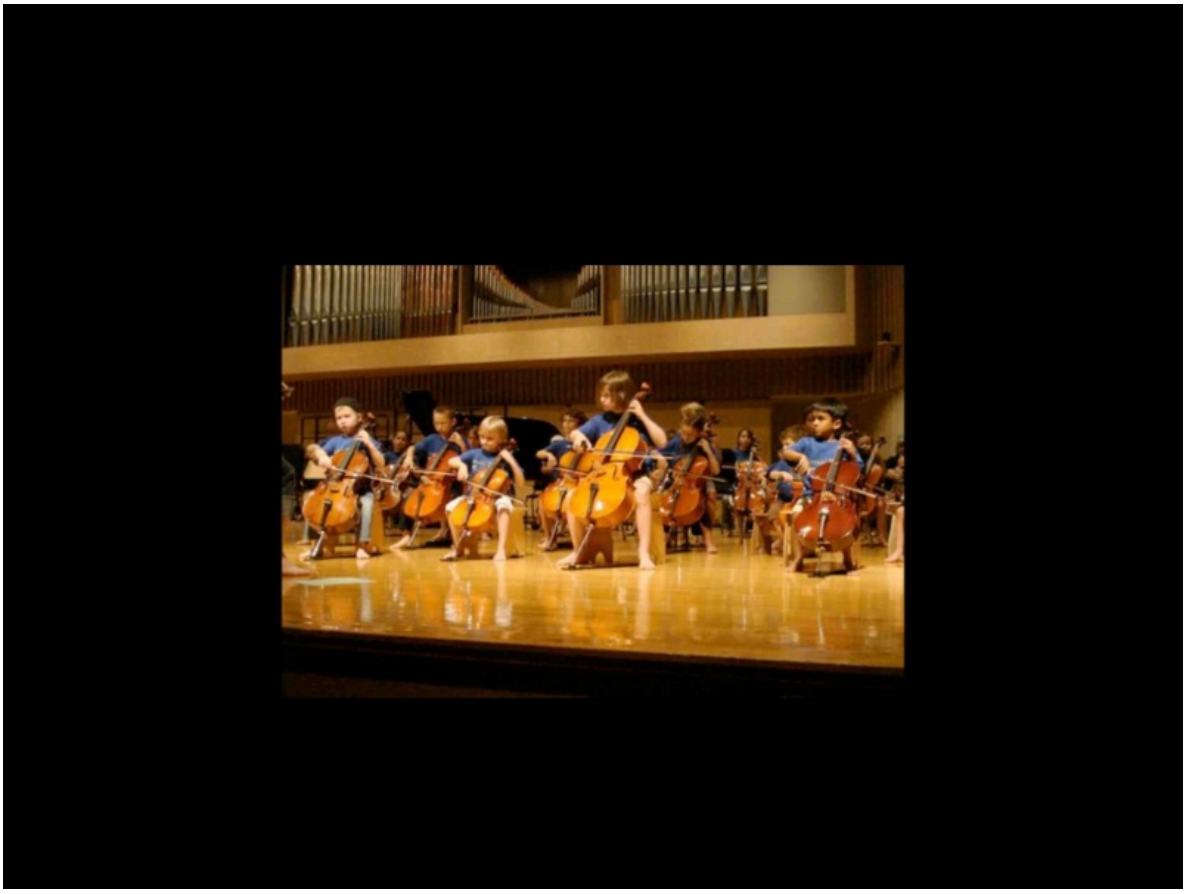

Figure 12: 00.34.13 - image-

Figure 13: 00.35.10 Players wanted

So we need players. I would rant here, but I will not. But look at this guitar player with blisters. A harpist has blisters, a base player with blisters. There is this barrier to overcome for every musician. Imagine if you downloaded something from GitHub and it gave you blisters.

[Audience laughter]

The horrors! And yet how many people here play an instrument, or have at one point in their lives? Yeah, a lot of programmers do. And for how many people did you just pick it up and it was awesome? How many people wished something could have made it more straightforward to get started with and just made it easy? And how many would have believed after that that they could play it later? No, not at all. This is ... it is actually quite important. The level of engagement that is required is quite important.

[Time 0:36:03]

```
slide title: Humans are Incredible

+ Learners
+ Teachers
+ Neither effort-free
```

So we should not sell humanity short. Humans are incredible. In particular, they are incredible learners.

One of the things that is really cool is you give a five-year-old or, I do not know, eight, maybe, a cello and some decent instruction, and they will learn how to play cello if they spend enough time doing it. In fact, humans will pretty much learn how to do *anything* that they spend enough time doing. We are incredibly good at it.

And we are also really good teachers, in general. So I do not think we need to go to our tools and our instruments and make them oriented towards the first five seconds of people's experience, because that is not going to serve them well.

It is especially not going to serve anyone well who wants to achieve any kind of virtuosic ability with the tools. No one would become a virtuoso on the cello if they had red and green lights when they started.

So neither of these two things is effort free, but we should not be in a game to try to eliminate effort.

[Time 0:36:55]

```
slide title: We are novices

+ Briefly
+ Permanently
```

Because we are novices. There is a sense in which we are only going to briefly be novices. You are only a complete beginner at something for an incredibly short period of time, and then you are over it. It is like we should not optimize for that.

But, on the flipside, we are always learners no matter how much time you spend on the violin. Who sits there and says, "I am done. I have completed learning violin. I finished it. That is awesome"? I personally do not play violin at all, but I do not think there would be a player on earth, no matter how great they are, who would say, "Yeah, I finished violin and I moved on to something else." We are constantly ... It is just the human condition to do this.

[Time 0:37:33]

slide title: Effort

[Two photos of musicians making unusual faces while playing.]

Is this the face you make when your emacs/IDE auto-completes?

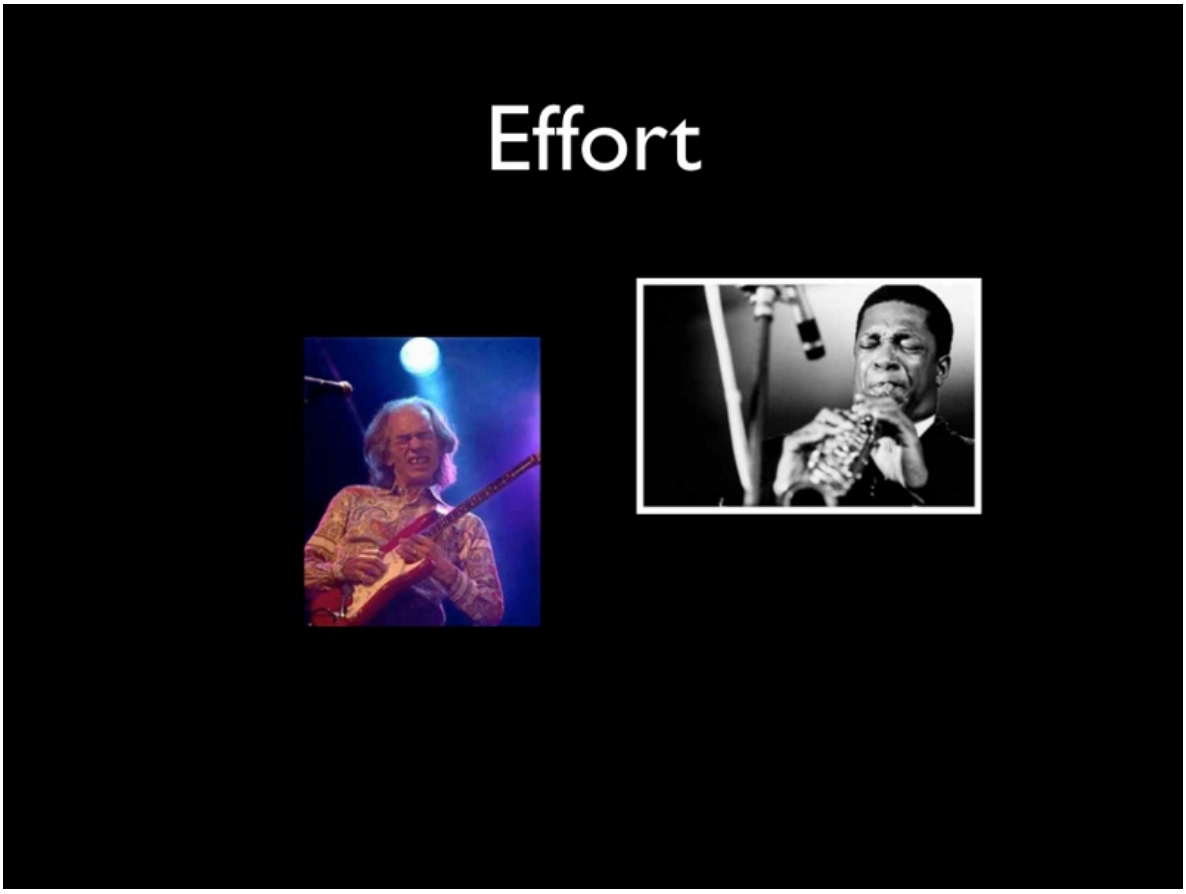

Figure 14: 00.37.33 Effort

Things take effort. Just like we should not target beginners, we should not try to eliminate all effort. Look at these two guys. These two guys are experts.

Is this the face you make when your IDE auto-completes?

[Audience laughter]

Does it look like that? Oh! java.util - oh, man. That does not happen. Your life has just been automated away. And I think that sort of what is interesting is that, yeah, it sort of looks hard and, in fact, it is probably not hard for either of these two guys. But what you are seeing here is a sense of engagement in what they are doing.

How engaged do you feel in what you are doing when you are programming with an IDE that is doing everything for you? You are so isolated from what is happening. So effort matters.

[Time 0:38:27]

slide title: Instruments (and tools) are usually for one user

[Photo of two men using a two-man saw to cut a tree trunk.]

Pairing?

Figure 15: 00.38.27 Instruments

Another interesting observation that is not really that important to this talk is that instruments and tools are usually made for one user at a time. Like this whole notion of two guys on one keyboard to program. That does not happen in instruments.

Now you make ensembles of instruments. I am doing this and you are doing that, and we are doing them together in the room, and it sounds great. We do that. But this two people pulling on one tool, that almost never happens.

So I wonder if this pairing thing is just a way to keep us from typing all the time, to buy one person some time to think a little bit. Whoever is not pulling has got an easy ride. Of course, this is pretty fast switch back and forth.

[Time 0:39:14]

slide title: Planning / Performance

```
[ Figure of pie chart with two colors.  Yellow portion is about 90%,
blue portion is about 10%. ]

+ What ratio of time spent?
  + compose / study / practice
  + vs perform / record
+ Why do we think we can just show up?
  + unlike other creative people

'In order to be creative you have to know how to prepare to be creative.'

     -- Twyla Tharp.  The Creative Habit.
```

So it begs the question. What ratio of time should we have between planning and performance? Which is which in programming? Which one is typing the code in? It is yellow. It is.

But is that the way other things work? No! How about for an orchestral musician? How much time do they spend practicing versus at the concert? Way more time. Way more time. And do they go and say, "I practiced at college, so I am done practicing"? No.

Why do we think that we can do this? We went to college or wherever we learned, whatever, and then we are going to go, and we are going to do it every day. We just do it from here on. Went to school; we are done.

And I think that we do need to assess how much time we spend. How many people spend 10% of their time designing? 25%? 50%? I am going up. No more hands are going to go up. No one spent 10%. It is quite sad.

[Time 0:40:20]

```
slide title: But ...

Coltrane couldn't build a web site in a day

<rant></rant>

Software isn't made of wood or metal
```

But there is a sense in which -

[Audience laughter]

There is a sense in which – I mean, it is sad. It is. It is actually sad. It is not sad as a joke - sad. It is actually sad.

But there is a sense in which it is like, all right, well, this is all so ... it is so different. Coltrane could not build a website in a day. I could. You know, I could do that.

Actually, I personally could not, but I know other people who can. And that is where another rant would go about how important is that? Why do we put so much priority on how fast can a beginner do something, and how can you regurgitate a template in a day? Because none of these are things that we need to do on an ongoing basis to solve problems for the world.

But it is a fair point that software is not like instruments. It is not made out of wood or metal. We have these ones and zeros. There are so many combinations. There are so many ones and so many zeros. It just seems so open. So how is this connected?

[Time 0:41:15]

slide:

[Photo of Theremin.]

Theremin

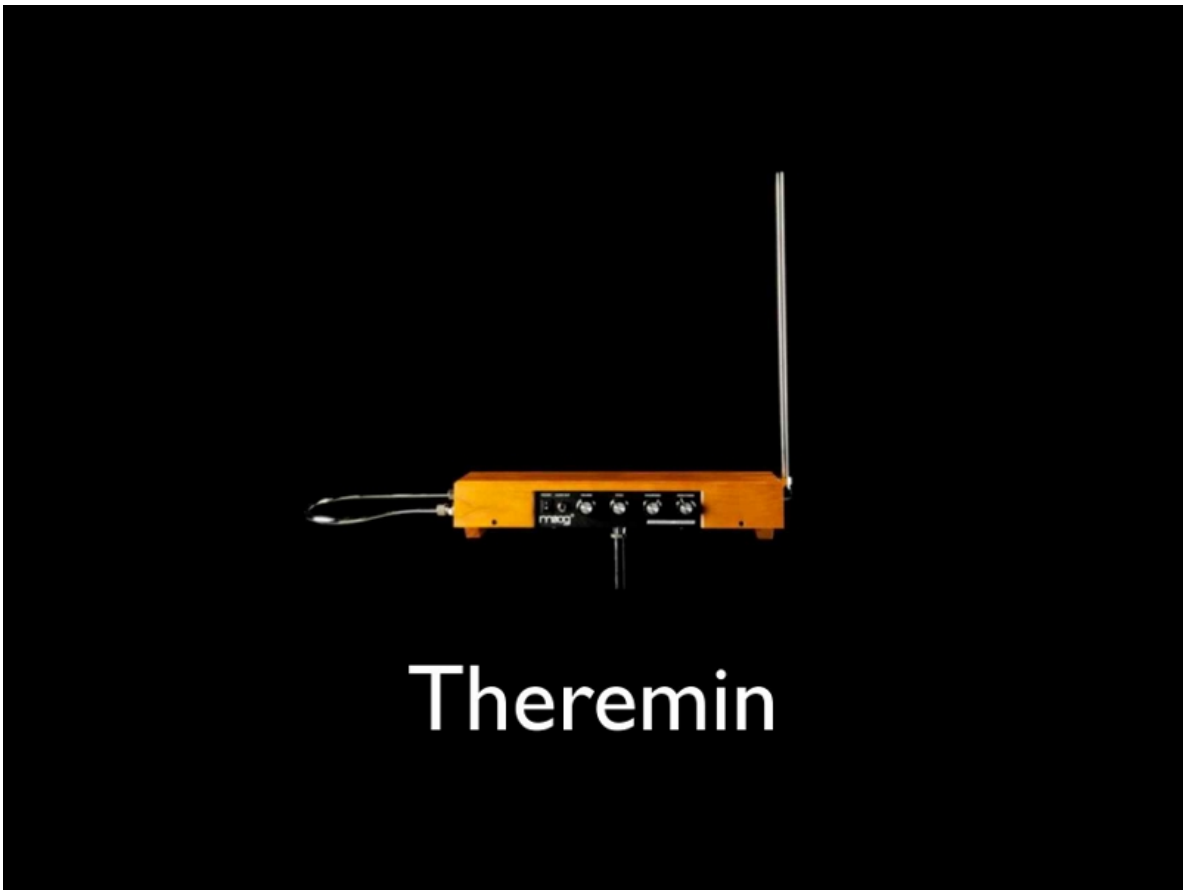

Figure 16: 00.41.15 Theremin

It ends up that there is this connection between instruments and things that are more technological, and they are technological instruments, or electronic instruments. This one I want to share a picture of, because I also have one of those. It is called a Theremin. No sooner did we have the ability to turn electrical signals into vibrating loudspeakers by having recorded stuff in order to send the signals through that somebody said: I wish we did not have to record stuff. I wish I could just make up an electrical signal and send it to the speaker. Let us cut out that performing and recording part and let us just do . . . let us just go right for the sound.

And so electronic music was born, and this is one of the first electronic instruments where you play this thing. It is two antennae, and the vertical one controls the pitch. The closer you get, the higher the pitch. The further away, the lower the pitch. And the horizontal one controls the volume. The closer you are, the lower the volume, and the further away you are, the higher the volume. So you can silence it by touching it. You do that, and that is all you got.

You do not actually touch them at all, and the knobs just change the timbre a little bit, but really not very much. It is not really about that. This is an incredibly difficult instrument to play, but it was

one of the first ones.

[Time 0:42:28]

```
slide title: Modularity

[ Photo of other electronic devices for producing or modifying
sound. ]

Control voltage!

[ Photo of inside mechanisms of one of the earlier things, I
believe. ]

Human vs machine interface
```

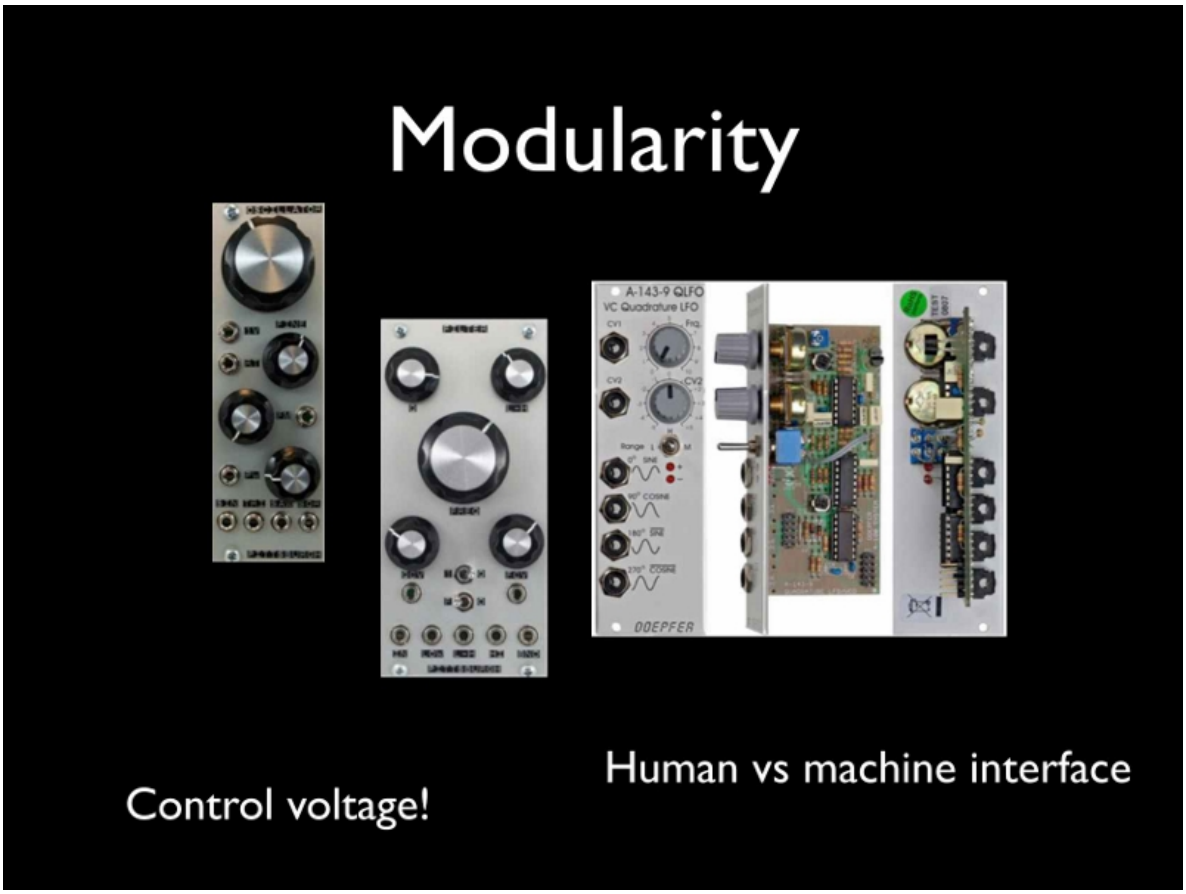

Figure 17: 00.43.48 Modularity - build slide

Then things grew up.

And now we are starting to see things more like we know, like we understand. These are some of the first electronic instruments that were made. These are the pieces of that instrument you saw before. Each module does a particular thing. It might generate sound, or generate certain wave shapes, or it

might be a filter that trims off high frequencies from those shapes, or it might generate a low frequency oscillation you can use to multiply something else.

And one of the really cool things about this is not only do you see examples of physical modularity, but you also see examples of control. So there are these little holes, these jacks on the front of these things, and they actually take in or output control voltage. It is just voltage. You send a voltage in and then, depending on the module, the voltage might control the pitch or the frequency of oscillation or something, or the frequency at which the filter kicks in, or the wave shape, or various things like that.

And then the knobs are redundant things that sort of give a human interface to what you could have done through control voltage by plugging something into the jack.

And there is something incredibly interesting about this. And then you see behind these, there is a circuit, so we have the layers of effort.

But there is a really good lesson here about human versus machine interface. These things had a machine interface first. It was all control voltage. And then they put knobs on it. So you could patch the control voltages around and build customized things.

Imagine if someone had built something – SQL – without any machine interfaces – Unix.

[Audience laughter]

But primarily with human interfaces, just the knobs. Like if I gave you a bunch of these modules and they just had the knobs, and I was like, "Put it together." You are going to be, "By doing what? Putting little remote controllers on the knobs?" Sort of like generating SQL text strings or something. Why would I want to do that? Or parsing random output from Unix programs, or specifying command line arguments. It is awful, right?

But these hardware guys are smarter than we are. So they built a human interface on top of the machine interface.

[Time 0:44:44]

```
slide title: Design stack

[ Photo of Yves Usson standing before a rack of electrical equipment.
Another photo on the right is of a device with a piano-like keyboard
and many other controls above that. ]
```

So we are seeing this thing, this stacking that occurs. This guy, whose name I am going to mess up, Yves Usson, is somebody who is awesome. He can work at all the layers of the stack. I think he is a biochemist or something. But in his spare time, when he is not a biochemist, he is a C++ programmer. In his other spare time he actually designs these modules. You see behind him a rack of these modules. But he designs the modules, so he can do the electronics work associated with building a module and building an analog filter, for instance, or an analog generator.

And then he obviously can compose them. He helps people make kits and then you can build them into racks. Then you patch them together. So then you are at another level of design where you are patching things together and setting the knob positions and designing a sound or patch, they call them, but setting a sound out of the module. Then maybe sometimes there is a little keyboard next to them. Sometimes he gets to play the keyboard and make music with this. But it is all these layers associated to what he does and can do.

He happened to work with this company, Arturia, to produce an analog synth, which is rare these days. They used to be, all synths were analog like the pictures I was showing you, but now they are kind of rare. Everything has become digital.

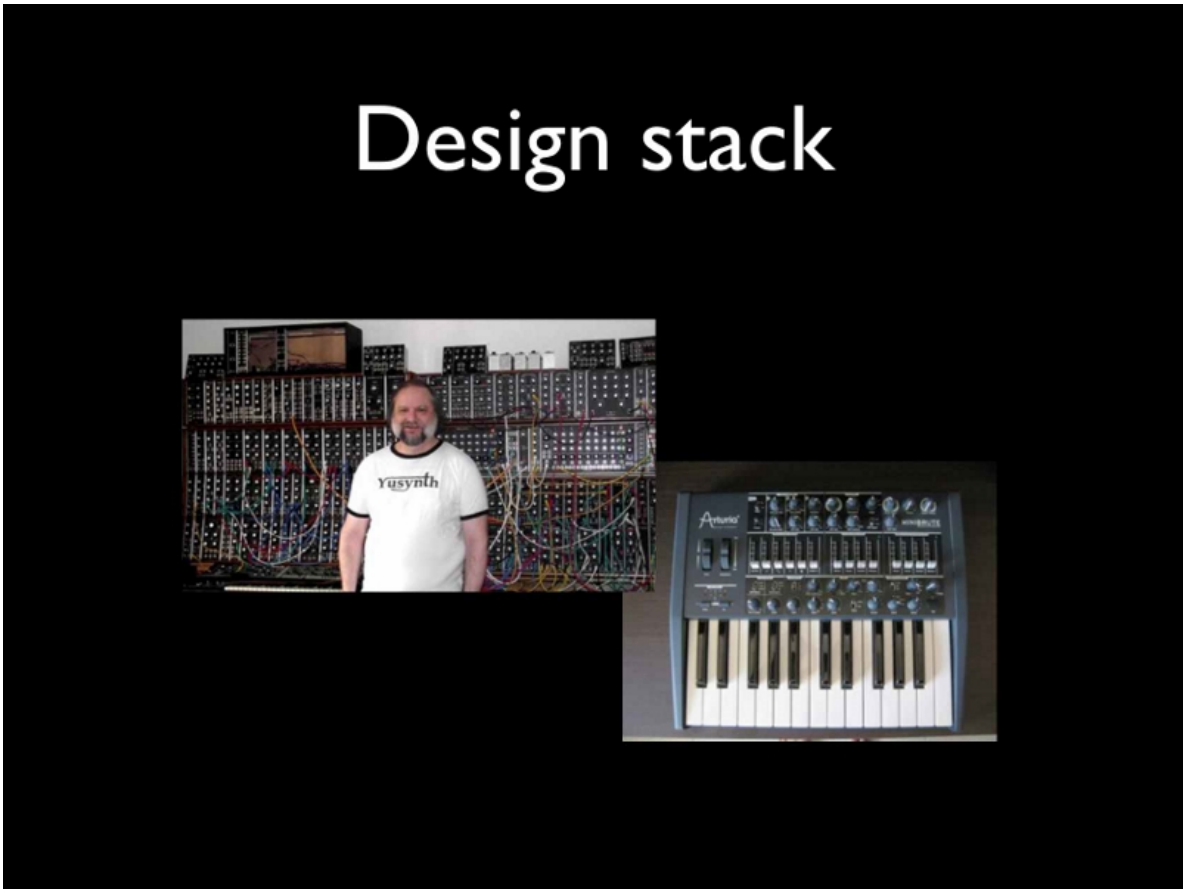

Figure 18: 00.45.49 Design stack - build slide

They came out with this analog synth, and he helped them design it. And he really did design. And what is interesting about what he did there was: this thing does not have wires coming out all over the top of it. The decisions about – it has the same kind of modules inside it, but the decisions about how they go together he made, he said, or he helped them make. He said we should make this go to that, and this is how the filter should work, and these are what the parameters should be.

There are still knobs on the top, but a lot of the other stuff has been incorporated in a design that allows people to only work at the next level up. They do not need to care about what is inside this box.

[Time 0:46:38]

```
slide title: There will be no music today!

[ Photo of a person's hands using what looks like a soldering iron to
add solder to an electronic board. ]

"You should use emacs!"
```

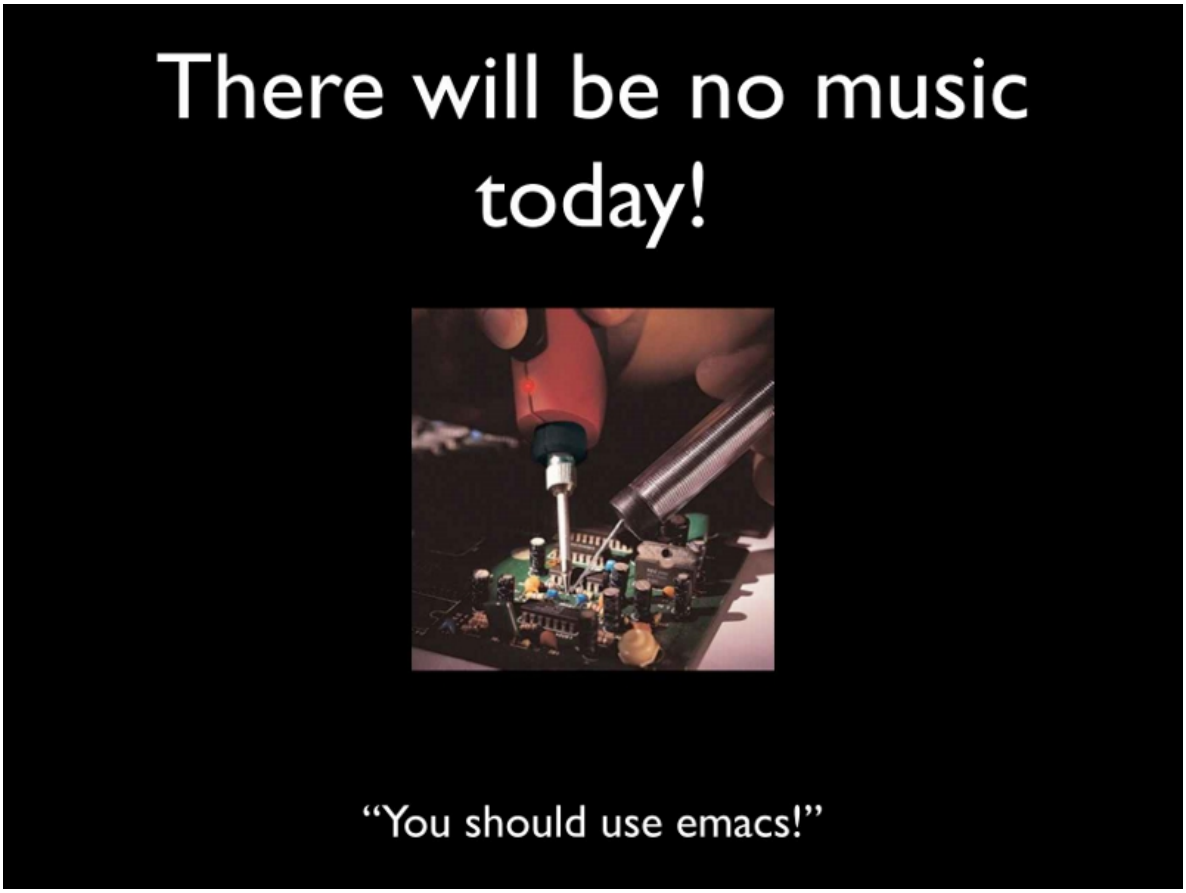

Figure 19: 00.46.52 no music today - build slide

And it is quite an important thing because, for him, he has different days. He has days when he is patching stuff together. Maybe he has days when he is playing this thing. And that is all fine, but days when he is soldering, he is not making music.

And this is what happens to us. This is what happens to us when we say, "you should use emacs". It is like somebody wanted to make music, and you gave them a soldering iron. It is like: here you go. Have at it. Start at the bottom.

[Time 0:47:05]

```
slide title: Code all the way down

+ In software, same mechanism at every layer
+ We all have soldering irons
+ Doesn't mean we can do filter design
+ Distraction, expansion
```

And why does that happen to us? The reason is because, for us, it is the same stuff all the way down. In that space, it is very different. Designing an analog filter is a pretty tricky thing just from a mathematics perspective, and then there is all the componentry associated with the electronics aspect of it. Then there is actually being able to solder and put it together on a circuit board.

And then somebody with a completely different skill set to go and say, "I can patch these things together, turn these knobs and listen, and understand what the architecture of these things is, and make a sound". And somebody else could walk up to that whole patch and say, "I could make a composition with this sound".

But for us, we have the same stuff at all the levels. It is code. The top level is code. The middle level is code. The bottom level is code. We can do it all. We have the same mechanism at every layer.

Essentially, we all do have soldering irons. It is like any time you want to, you can start soldering. You are supposed to be up here doing this, but you could just start soldering.

And just because we have the soldering iron does not mean we are capable of doing things at all layers, but we just do because we can. We have got the iron in hand.

And I think it leads to a lot of distraction and expansion of scope of things. It is like, I was working on this, and then I realized if I rewrote the driver, I could be 10% faster. And now I am doing something I should not be doing.

[Time 0:48:25]

```
slide:

+ The paralysis of choice
+ The impetus of constraint
+ _Constraint is a driver of creativity_
```

There is a sense in which having so much control over so many parts of the stack, it gives us this paralysis. There is so much we can do at every point. So what are we going to do?

And I think that we need to – of course, the problem space has some constraints, but we need to bring constraints of our own into play. We have to do this for ourselves, the same way composers do it for themselves, or choreographers or directors do it for themselves. They bring constraints in to help them move forward.

This is not a new idea. This is a very old idea, but it is one we have to keep remembering. Constraint drives creativity. When you do not have a lot of choices, you are forced to pick an answer and move on. We all have choices. You could just mull around about the choices all the time. So making your own constraints is a way to help you do that.

[Time 0:49:09]

```
slide title: Quit fidgeting

+ and agglomerating
+ and fiddling
+ and tweaking

[ Cartoon drawing of a person with an impossibly long neck gazing at
their own navel. ]
```

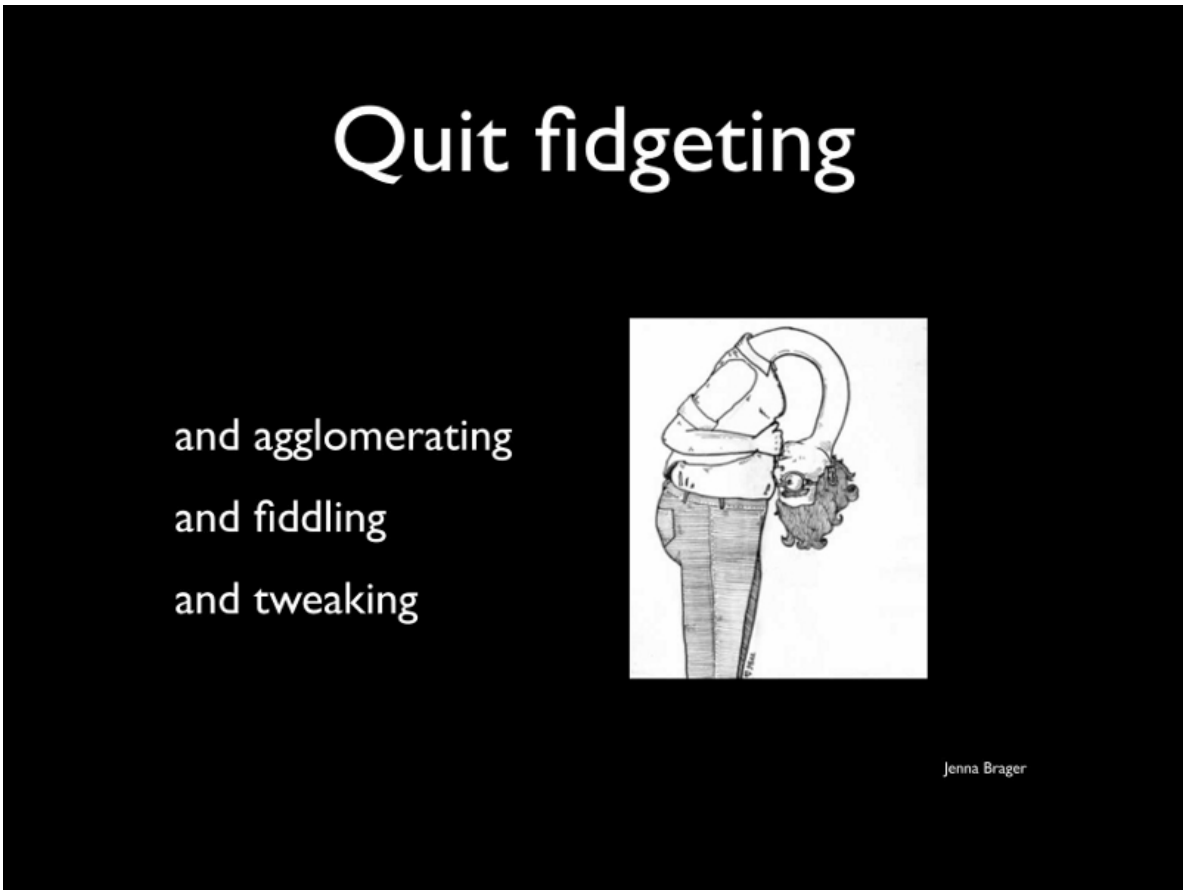

Figure 20: 00.49.09 Quit fidgeting

So I think we need to quit fidgeting, and glomming stuff on, and fiddling around with things and tweaking. I mean, oh, my God. As an industry, we spend an inordinate amount of time focused on ourselves: build tools, automating this and that, and just crazy, crazy, crazy stuff. And talking about it any everything else, and we should just be focusing on what we are doing.

Because what ends up happening is, when you keep fiddling with stuff, and when you have no limitations to scope, and no constraints, what happens?

[Time 0:49:38]

```
slide:
```

[Photo of electronic devices with a mess of wires interconnecting them.]

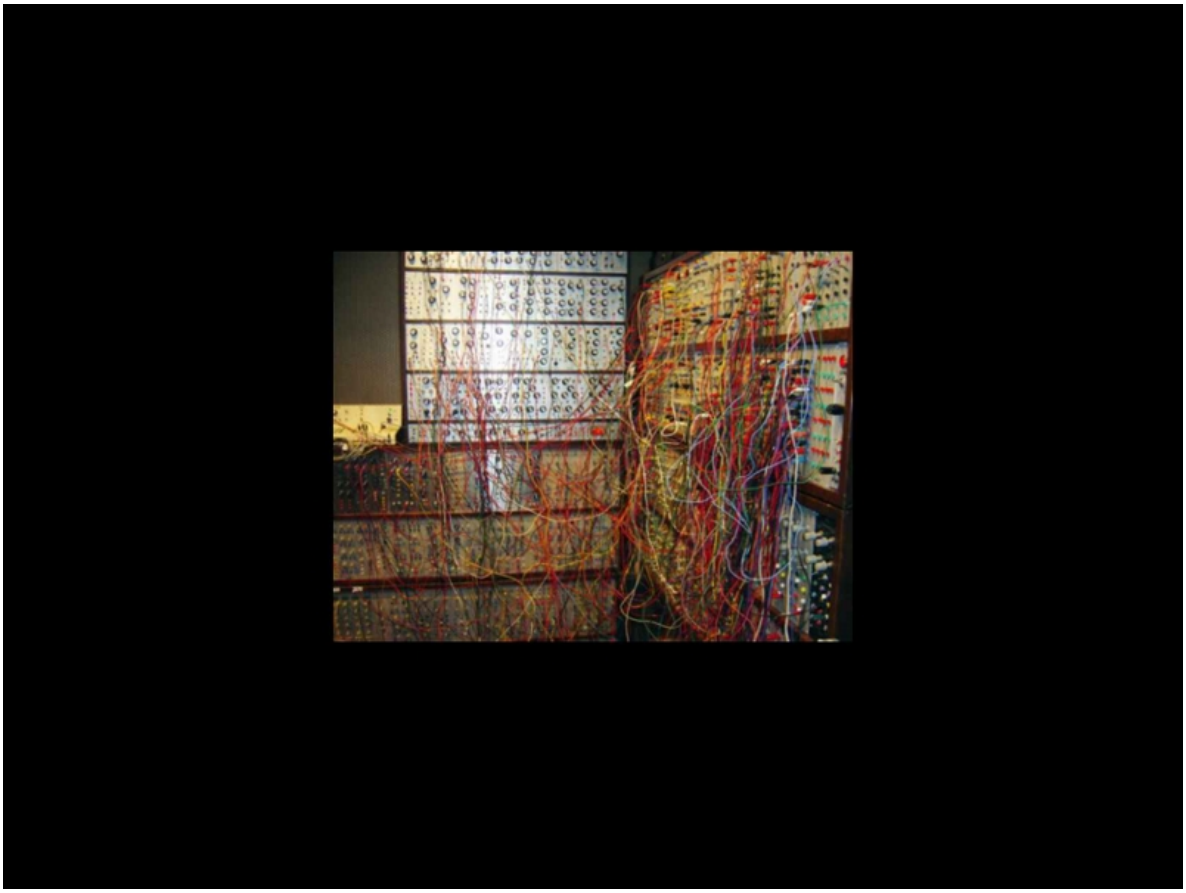

Figure 21: 00.49.38 - image -

This thing happens. And every one of those parts may be a good idea. They are probably all good ideas. But if you take every good idea, you end up with that.

I do not care if you configure this thing with Spring; it is not playable.

[Audience laughter]

No one wants to play this. And in fact, this particular one, no one does play. It plays itself.

[Audience laughter]

The actual patching of it is the composition, and it has got stochastic elements in it that cause it to generate novelty, and it plays itself in a museum. I mean, maybe we want programs like that, but maybe we do not.

[Time 0:50:28]

slide:

[Photo of a woman playing a Theremin.]

Carolina Eyck
https://www.youtube.com/watch?v=7l9YcewEumw

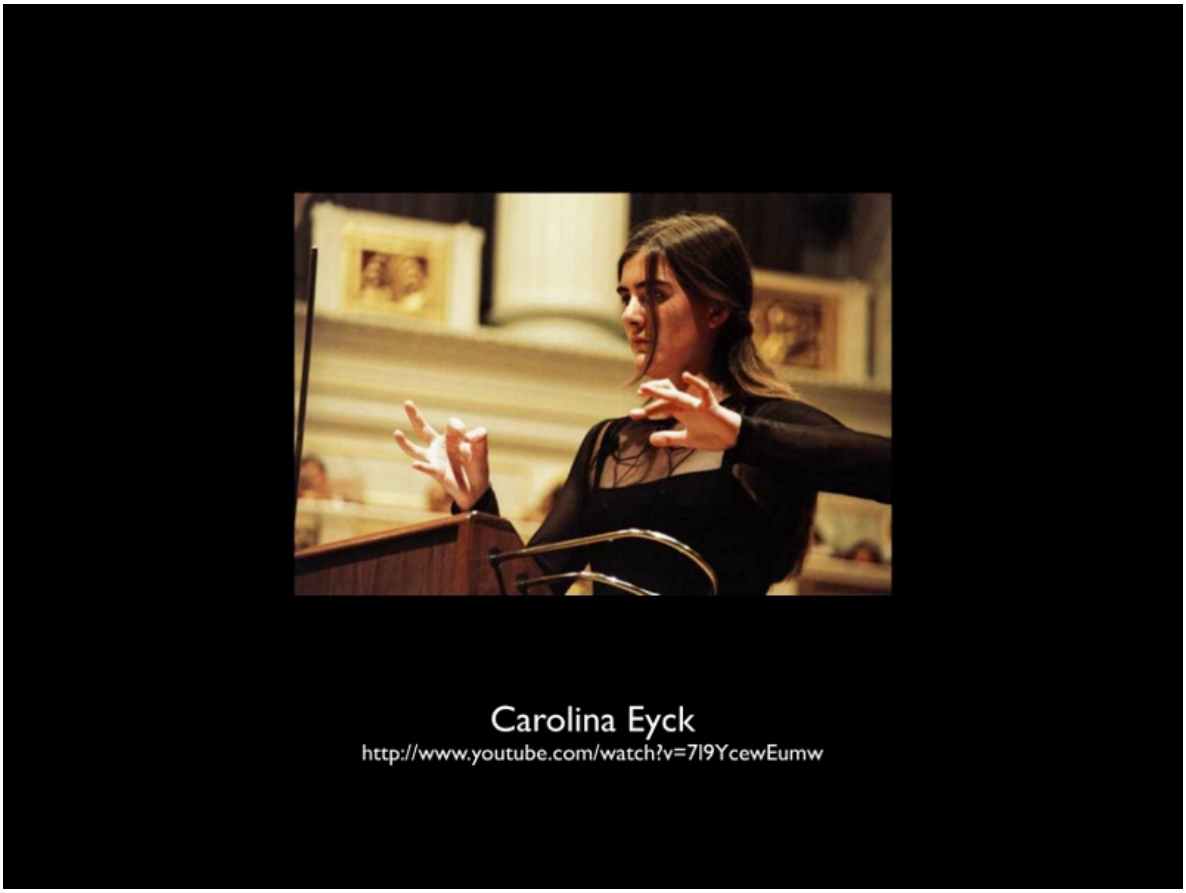

Figure 22: 00.50.28 Carolina

So we should push back, I think, especially in open source projects. There is this constant pressure. Take my good idea. Take my good idea. Take my good idea. And they are all good ideas, but whatever.

And we need to remember there are people who make music by waving their hands through the air. That is it. They do not need emacs or anything else. They can just do this. And I am telling you, if you have every tried to play the Theremin and have it sound like anything other than a siren or a spaceship, it is brutally difficult to do.

And so, I mean, I do not know if you can see it in her face, but she is not making a face like the other guys, but she is engaged. I think the reason why she is not making a face is because the pitch changes if you make a face. It is that sensitive.

[Audience laughter]

But at some point, go listen to that because it is beautiful.

[Time 0:51:18]

```
slide title: Design is imagining

+ Embrace the constraints
+ _Be optimistic_
+ Imagine a lot
```

So what is design? If we take a step back and sort of merge all of these things together. There is a sense in which design is imagining. If it is not just regurgitating something that is already happened before, you are facing some set of problems. You have to imagine potential solutions.

And the first thing you need to do is rush at the constraints. You do not want to be like, do not constrain me; I am trying to design. It is the opposite of that. You are like, give me, give me, give me the constraints. I want to know about everything. And, if you have not given me enough constraints, I am going to make up some, because I want this thing to work.

Of course, when you are facing all these constraints, it seems like negative. I cannot do this; I cannot do that. It must do this and this size and whatever. It is like, oh, you know. So you have to be – there is a sense in which designing is fundamentally an optimistic activity. You have to stay positive. In spite of all these constraints coming your way, you have to stay positive.

Remember, people do design that have no constraints, and pick constraints in order to get outcomes. So that optimism can be born of the fact that this works, and this is the way to make systems that work. And you want to imagine a ton of things.

[Time 0:52:26]

```
slide title: Design is making decisions

+ Admit little
+ _Value conveyed is in decisions made_
+ Leaving all options open is avoiding design
```

However, actually designing is about making decisions, which means you try to think up 100 times as many things as you actually use, way more things than you use.

You do not want to think of one thing and be like, OK, let us go do that. You want to think of ten things and then say, “this one is the one we want to do”.

So you want to admit very little.

You want to be able to say no, because the value that you convey in your design is strictly about the decisions you have made. When Yves helped make that synthesizer, he made a set of decisions. Are they perfect? No. Is it everything you want? No. Do I wish I could patch a wire from here to there? Yeah, sometimes I do. But you know what? I really appreciate the fact that this thing just works, and it sounds great, and I can do the next thing. I do not have to fiddle around with the inside of it.

So if you leave all the options open, you are not designing. That is not design. Everything configurable, that is not design. That is like: do your own thing.

[Time 0:53:23]

```
slide title: Performing is preparing

+ Practice
+ Study
+ _Developing design sensibilities you can deploy on the fly_
```

So performing is preparing. It is planning. You have to practice. You have to study.

And, in the end, what you want to try to do is develop sensibilities that you can apply when you are trying to write code. If writing code is the performing part, you have to have patterns, techniques, knowledge about what works and what does not, to apply to what you are going to do. You cannot just make it up as you go.

[Time 0:53:52]

```
slide title:

+ Take things apart
+ Design like Bartók
+ Code like Coltrane
+ Langs and libs like instruments
+ Pursue harmony
```

So design is taking things apart in order to be able to put them back together. And that is really all it is. Every time I encounter something, I can boil it back down to that. Every time I encounter something that I wish my design was better, I need to do more of this. It is over and over and over again. It is always this. I did not take it apart enough.

You want to design like Bartok. That is to say, you want to communicate very well. You want to be able to work at multiple levels.

And you want to code like Coltrane. You want to take preparedness and experience, real experience with doing things, not experience by doing the same thing over and over again. And bring them to bear in what feels like a more improvised thing. I am encountering a new scenario in a programming project. I am really not making it up. I am really bringing my background into play to solve that problem.

I think you want to find and choose languages and libraries that are like instruments in all the ways I talked about in terms of being simple, directed at one thing, oriented around people that know how to use them, and expressing and backing some fundamental excitation or idea. Those are going to be the most satisfying.

And in the end, pursue harmony in your own designs. Try to think about the nature of harmoniousness in software, what makes things work together, and apply that.

[Time 0:55:19]

```
slide title: Rock on

[ Photo of Pete Townshend of the band "The Who" jumping high in the
air, legs spread wide, playing his electric guitar in a flamboyant
fashion. ]
```

But thanks very much for listening, and I hope you enjoy the rest of the conference.

[Audience applause]

[Time 0:55:22]

Figure 23: 00.55.19 Rock on