Step Away from the Computer or Hammock-driven Development

- **Speaker: Rich Hickey**
- **Conference: Clojure/Conj 2010 - Oct 2010**
- **Video: https://www.youtube.com/watch?v=f84n5oFoZBc**

[Time 0:00:15]

```
slide title: Step Away from the Computer

             Rich Hickey
```

Hi. As I said yesterday, I swapped the talks around so this is the more philosophical talk. There are multiple titles.

[Time 0:00:35]

```
slide title: or, Hammock-driven Development
```

I could not decide. Some of them were semi-cranky.

[Audience laughter]

[Time 0:00:40]

```
slide title: or, If We're Not Solving Problems,
               What are We Doing?
```

[Time 0:00:45]

```
slide title: or, What Rich Does
              When He's Not
              Writing Tests
```

[Audience laughter]

In those moments.

And then Mark gave me a great Phaedo [tbd] line. So this is about step one.

[Time 0:00:59]

```
slide title: A Rant

      In any case
```

This is just an experience report.

[Time 0:01:01]

```
slide title: An Experience Report

+ i.e., not cognitive science, social science,
  computer science or any other kind of science
+ YMMV etc
```

It is not advocacy. There is little methodology here or science or anything else.

[Time 0:01:11]

```
slide title: When was the last time you ...

+ Thought hard about something for an hour
  + a day?
  + over the course of a month?
    + a year?
+ Felt confident implementing something for the first time?
```

So I would like you to think about when was the last time you thought about something for an entire hour? Like nobody bothered you, and you had an idea, and you sat for an hour and thought about it. How about for a whole day? Does everybody remember the last time they sat and thought about something for a whole day?

How about over a course of a month? You had something you were working on and obviously, not spending all the time every day when you started thinking for a month.

Or a year? These are tremendously valuable moments if you get to have them at all. I consider myself extremely lucky to have had the ability to think about probably three different things for a year or more. One of them was Clojure. And there is nothing I prize more than that kind of time.

The other thing I would ask is: when was the last time you felt confident trying to do something you had never done before? And what do you think it takes to become confident doing something you had never done before?

Obviously, as software developers, a lot of times we are doing the umpteenth application that takes something out of a database and puts it on the web, but the luckier you are, the more likely you are to encounter problems you have never done before. And how do you start doing that? And not feel incredibly at risk?

[Time 0:02:55]

```
slide title: Bugs

+ Most expensive in production
+ Expensive in QA
+ Cheaper to fix in dev
+ Least expensive to avoid in design
+ Most of the biggest problems in software
  are problems of _misconception_
  + Not implementation
  + Testing and type systems are of limited use here
```

So I will start by talking about some software development things we all know to be true. We hate bugs in our programs. We are trying to write quality programs. And we know, if we let the programs reach the field, it is incredibly expensive to fix inadequacies and inaccuracies in the program.

Right, so we say we will have a big testing process in Quality Assurance. And even that, we know, is not so great because it sort of has this removal, this distance from the development effort, which is not good. So now, we know what to do in this area. We fix bugs while we are coding by testing in development, and this is the best way to avoid bugs in our applications, right?

[Audience members: No.]

No.

Did you notice that I learned how to make each bullet happen one at a time?

[Audience laughter]

So now, you do not know what I am going to say next.

[Audience laughter]

It is awesome. It has got this menu item for it.

No, definitely, absolutely, positively not. The least expensive place to fix bugs is when you are designing your software. Which everybody does, yes?

[Audience members say yes.]

Mm-hmm. I will contend of all the things I am saying here, which are very very extremely fuzzy, that without a doubt, most of the *big* problems we have with software are problems of misconception. We do not have a good idea of what we are doing before we do it. And then, **go**, go, go, go and we do everything. We have practices and all kinds of stuff, and we feel really good about ourselves, after that point. But if you mess it up, as Mark said, in step one, it is not going to turn out well.

They are not problems of implementation. There *are* problems of implementation, obviously. And testing and other things help with those. But problems of misconception are not generally addressed by testing, or type systems, or the things we use to correct defects in implementation. There are not really type systems that can tell us if we have got a good idea, or what we are doing addresses that idea.

[Time 0:05:16]

```
slide title: Analysis and Design

+ I know, so '90s
+ Got dinged as if about making pictures
  + not UML, nor all-encompassing requirements or plan
+ We are suffering greatly due to a lack of attention to these
+ Simplify - A & D are about making sure we understand, and are
  solving, problems
```

I am going to talk a little bit about Analysis and Design. I know that is so '90s, and ugly, and was rightfully criticized, and really dropped. Because people considered it to be about process, and drawing pictures, and knowing everything about everything, and making comprehensive plans in the Waterfall Model, and there was amazing amounts of stuff that was *terrible* about this.

But that *doesn't* mean that the step before "go do it" is not an important step. I think we do not spend enough time and energy, or make enough time, or get allocated time. It may not be a matter of our choice. We could say: oh, we would like to spend some time thinking about it, but we have to ship something next week. But we are definitely suffering in quality because we do not spend the time here.

So what I would like to do is sort of just whatever you think Analysis and Design is, I would like you to just forget for the moment, and let us try to make a really simple definition. Analysis and Design is about two things: identifying some problem that we are trying to solve, and assessing our proposed solution in terms of whether or not it solves that problem. That is *really* what it is about, not about anything else. We should be solving problems. We should not be building features.

[Time 0:06:49]

```
slide title: Solve Problems!

+ Features are results, not objectives
  + No guarantee that a sum of features is going
    to solve anyone's problem
  + or not introduce new problems
+ Don't let users specify answers
+ Avoiding problems != solving them
+ Solving problems requires understanding,
  recognizing and stating them
```

There is nothing about feature . . . What is feature? Feature is just an attribute of something. It is the shiny chrome knob on something. It is not the purpose of the car. There is no guarantee, if you put together a feature list, even if it comes from the customer, that it is going to solve their own problem, or that it solves *any* problem, or that the features, when you put them together, do not introduce a whole ton of other problems. So programming and writing software is not about completing lists of features.

In particular, features provided by users, in spite of their best efforts to satisfy themselves, are often really not good ideas. You have got to dig underneath it and figure out what problem they have, and what is the best solution to it, and then reconcile it with whatever they ask for.

We also have a tendency, because we are all smart, and we love being smart and sort of figuring out how to make things go, that figuring out how to make something go is good, no matter what it took to do it. If we can find a way to get around a problem, we are like, "Whoo! That is great."

And it is not great. Avoiding problems, which we are all capable of doing, very capable of doing, is not the same as solving them. So we should really try to work on solving problems.

The thing that I am going to talk about today is really that there is a bunch of technique and skill to solving problems, and the first one is just to make an effort to understand the problem you are working on, to recognize, identify it, put it somewhere and talk about it.

[Time 0:08:36]

```
slide title: Problem Solving is a Skill

[ cover of book "How to Solve It" by George Polya ]
```

http://en.wikipedia.org/wiki/How_to_Solve_It

So problem solving is definitely a skill. I think you should not take away from this talk that there is a certain kind of person who is good at problem solving, and they get to do this part of the job, and we can practice these other things. You can practice this part.

Polya wrote this amazing book called, "How to Solve It" in 1945 or something, which is about how to practice *how to practice*, and what are the techniques of solving math problems in this case. It is a

terrific book, full of great insight. And if you have never read it, go onto Amazon right after my talk and order yourself a copy.

One of the things that is not so great about the book is that it *is* in the math space. In that space, there is this really nice thing that happens. When you are done, and you think you have an answer, you have all the techniques of mathematical proof to determine if you actually have.

Whereas as software developers, we do not have that. There is no way to prove that you have a good solution to somebody's e-commerce site problem. There is no mathematical techniques, and there is not going to be any anytime soon, that will let us do that.

[Time 0:09:48]

```
slide title: Practice

+ Humans get good at what they practice
+ Practice problem solving
  + get good at problem solving
+ Practice methodology X
  + get good at methodology X
+ Where's the greater leverage?
```

But it *is* a skill, and it is something you can practice, something you can learn about. And it is worth doing, because as human beings, we get good at what we practice. It does not matter what it is. There is amazing examples of people practicing things that they seem to have no potential hope to become good at it, and they get good at it because they practiced it.

If you practice problem solving, really practice problem solving, you will get good at it. If you practice methodology X, you will get good at that. I would like you to ask yourself, where do you think there is more leverage? I do not care what X is. Pick any X you want. Would you rather be good at it? Or the general skill of solving problems?

[Time 0:10:38]

```
slide title: State the Problem

+ Say it somehow
  + out loud in conversation
  + or (heaven forbid) write it down
```

So what do we need to do? If we are going to work on solving problems, what is the activity like?

The first thing is to actually say, "I am solving this problem. This problem is this. Blah, blah, blah, blah, blah and therefore, blah." I have seen so much software made where no one ever said that. No one ever wrote that down. Then we have a whole system, and no one said what problem it is supposed to solve.

If we are not solving problems, I have no idea why we are in this room. We absolutely should be working on solving problems, which means we should be enumerating what they are.

And then from the mental standpoint, which I will talk about a little bit later, it is actually important to say them out loud. As the person who is trying to solve a problem, say it. Have a conversation with somebody in your group and say, "We need to solve this. The problem is blah, blah, blah." Rant or talk and have a little conversation, or write it down. But just like you use the practice of repeating

somebody's name when you are introduced to them as a mnemonic to help you remember their name. It is the same thing. This is the seed of solving the problem, is stating it.

[Time 0:11:45]

```
slide title: Understand the Problem

+ What do you know?
  + facts
  + context
  + constraints
+ What don't you know?
+ Are there related problems?
+ Write it all down
```

The next part, which is definitely trickier – and Polya's book is great. It has got a lot of practical things. Many of these overlap what he said – is to understand the problem.

So we said, "We have this problem. I think we need a NoSQL database." There is something missing, right?

[Audience laughter]

"We have this problem. We need a NoSQL database." We have not actually said why. What are the characteristics of this problem that lead us to this solution space? And this is where all of the interesting work is, I think, in software development.

And so the first step is, what do you know about what you are trying to do? There is definitely going to be a bunch of facts. There will be customer requirements. There will be other things. There will be context: the system has to run on this kind of box, has to run for this long. It cannot consume more than this many watts, or has to support 10 million users. Whatever it is. There are those kinds of things, and constraints. All the stuff are facts you know about what you are suppose to do.

There will be things that right away you know you don't know. "I wonder where we are going to get the data as an input to this thing?" Or: "What are we going to do when our main data source for it is not available? Do we have a secondary thing?" There will be things like that.

Of course, there will be things that you don't know you don't know. Well, that is fair, but if there are things you don't know, you should think about them now.

The other thing to do is to say, you know, everybody says, "Oh, do X. I have this great idea for X." As if you are the only person in the world that ever had this problem to solve. That is very, very unlikely, so go find some other solutions to similar problems. Are there any others that you know about? What can you find out about them?

Because looking at other solutions to the same problem is *the* number one way to get up to speed really quickly, and start working ahead of the best known solutions in this space. Because what you will have to do then will just be an incremental step above what the last guy did. But if you are ignoring what the last guy did, you are starting from scratch, so you definitely want to look around in the space.

Now, I am not advocating a methodology or anything, but if you are going to bother to do all this work, you should write it down somehow, in some way. I do not care how.

[Time 0:14:17]

```
slide title: Be Discerning
```

```
+ Find the problems in your solutions
  + Solve those too
+ No tradeoffs?
  + Not likely
+ No questions?
  + Who knows everything?
+ Write these down too
```

The other thing you have to do is: you have to be discerning. You have to be critical. We are sort of in this world because there is all of this community stuff. It is like, I just hear "awesome". I would like awesome to happen. I just hear it 50 times a day. Not everything is awesome.

[Audience laughter]

So it is hard to talk about other people's stuff not being awesome. So just, mainly, focus on your own stuff. In particular, as you are finding solutions, as you are trying to get on your way to a solution to a problem, look for defects in your own solution.

And of course you could have a whole talk about this, because there will be technical errors. There will be errors in logic. There will also be errors of taste and judgement, and abstraction, and all those kinds of things. It all feeds into this, and there is an entire talk in this area. But whatever issues you can find in your own solutions, try to solve those too, right away, up front.

So the other thing you see is, "Oh, we are going to do this." "Oh, we use a NoSQL database." "That is great. It has these 10 attributes. It is awesome." It is really easy to get excited about the good parts of what you do, but you should be looking for tradeoffs. The chances of there being no tradeoffs in any solution are slim.

The other thing is just this, "what don't you know?" thing. If there is stuff you know you don't know, there are questions you should be asking in order to find out what you don't know. You don't know everything, so there should be question marks on this. Whatever it is that you want to use, that you are going to write all this stuff down. There should be question marks on that page. If there are no question marks, you are missing this step.

[Time 0:16:12]

```
slide title: More Input, Better Output

+ You can't connect things you don't know about
+ Read in and around your space
+ Look (critically!) at other solutions
```

The other thing is you have to think about: none of us are born knowing how to write software. None of us are born knowing about SQL, or the characteristics of the web, or protocols, or anything else.

And if you are trying to solve a problem, especially in a space where you have not done it before, you are going to have a very limited ability to come up with a solution if you do not have a lot of input. You are going to need to get a lot of different inputs so that you can let your brain go around between them and say, "Oh, yeah. This idea and that idea are connected to each other and therefore, I can do this other thing."

If you only take a really narrow slice of, "I see exactly what I am doing right now, right this second to deliver next week", you are not going to have enough inputs to make decisions. So you want to read about the kind of space that you are in, widely, very specifically. Ooh, there are other people trying to do exactly the same thing; and then broadly, there are these other characteristic problems; and maybe even, if you want, go try to find research papers that are kind of in the same space.

It is amazing the cool things you can find by searching something like ACM for papers about the kind of, it is like, "Oh, I wonder if we can get a certain kind of hash code that does whatever." You go into Google and type, "hash code that does whatever." Enter. And if there is some scholarly, and ACM references, grab those papers. Even if you only understand like a tiny fraction of the paper, it is likely to contribute to your ability to think about your problem.

The other thing is: even if you are not going to tell the other guy, when you are looking at other solutions, be *extremely* critical. I cannot tell you how often you are going to find the next best idea by completely crucifying the last guy's idea. At least, in your own head. Take it apart, because when you take it apart, you are going to find a couple of things maybe they did not write down when they were doing it.

[Time 0:18:07]

```
slide title: Tradeoffs

+ Everyone says design is about tradeoffs
+ but...
  + You need to enumerate two or more possible solutions...
    + and the attributes and deficits of each
    + ... in order to make a tradeoff
+ You might want to write these down also
```

So everybody says design is about tradeoffs. Everybody knows this. Usually when they talk about tradeoffs in their software, they are talking about the parts of their software that suck. I had to make these tradeoffs. That is not what a tradeoff is, right?

You have to look at, at least, two solutions to your problem. At least two. And you have to figure out what is good and bad about those things before you can say, "I made a tradeoff." So I really recommend that you do that. And when you do it, you might want to write that down somewhere.

[Time 0:18:55]

```
slide title: Focus

+ On the hammock, no one knows you're not sleeping
+ The computer is a prime source of distraction
+ You can't do everything
  + balls will be dropped
+ Let loved ones know you are going to be 'gone'
```

Okay. So let us talk a little bit more about practice. A big part of trying to do this work is maintaining your focus. We had a really nice talk yesterday about flow, and that is a kind of a focused-related concept. And when you are trying to do design work, you also need, I think, some of the most extreme focus you are going to ever need.

And so there is some cool aspects to the hammock. One of the cool aspects to a hammock is that you can go on a hammock and you can close your eyes, and no one knows that you are *not* sleeping, but they will not bother you be cause they think you might be sleeping. So it is very cool.

Computers are bad, bad sources of distraction. They are so bad, especially for people like us. It is just like, uh... something else besides what I am trying to think about. You desperately need to get away from the computer if you are trying to focus. It is almost impossible to focus sitting at a computer.

The other thing about focus is that you are going to be making tradeoffs when you try to focus really intensely. You are going to drop balls. You are going to miss calling people back and responding to emails, and doing your slides for conferences until the airport on the way there and things like that. That is just the game.

The one thing though is that you should communicate to people that you care about, about this process. And the fact that when you are doing it, you are going to seem pretty far away. And that is not a comment about the person that you care about. It is just the nature of doing this kind of work, so it is important to sort of do it.

A lot of people will not get time to do this all day every day, or over the course of an entire week, but if you are going to get some focus time, define what that is. Everybody knows about time-out time for little kids. Well, programmers need this focus time. So, like little kids, they need to go sit on the hammock and have nobody bother them.

[Time 0:21:10]

```
slide title: Waking Mind

+ Good at critical thinking
  + analysis
  + tactics
+ Prone to finding local maxima
+ Use waking mind time to feed work to background mind
  + and to analyze its products
```

So for me, personally, I think that the process involves two parts of your mind, and this is stuff that you are seeing. There are books written about this and whatever. I have not read them, but they seem to correspond to my personal experience, which is that you sort of have this waking mind and background mind.

And your waking mind is really good at that criticizing part. It is extremely analytical, and it is very, very good at tactics. Like right now, we need to make a decision. The lion is chasing after us. Jump left. We are really good at that. That is what our waking mind is about: keeping us alive and making short-term decisions, and looking at the immediately present information, and doing something about it.

However, if you think you are going to sit down and look at a problem for the first time, and stare at your computer, and do whatever, and have a conversation for 10 minutes, and make a really great decision. I do not think so. I know I cannot do that. Definitely not.

The problem with this kind of thing is it tends to push uphill. "Oh, I see this. Oh, I see that. Oh, I see... okay, here I have a choice of left and right. Okay, I can go right. That is more up. Left or right, right. You know, left. That is more up. More up."

This part of your thinking is really good at finding the local maximum, but it is not very good at getting off the track it is on, and finding the fact that there is another hill over there that really takes you higher.

But there is a very, very critical activity that you have to engage in, I think, if you want to use your entire brain and become very good at problem-solving. And that is, to think about using your waking time to assign tasks to your background mind, to actually think hard about something and create work for your background mind. That really is the point of the hammock, and all this listing, and all this work that we are going to talk about you are going to do when you are awake, is actually to give the other half of you stuff to do.

The other good thing about your waking mind is: when you do think that you have a great idea that you have come up with in your background mind, your waking mind is good at picking it apart, saying, "Ah. You thought you woke up with this brilliant idea but now, I am seeing this characteristic of it that seems not so brilliant."

[Time 0:23:37]

```
slide title: Background Mind

+ Good at making connections
  + synthesis
  + strategy
  + abstracts, analogizes
+ Solver of most non-trivial problems
  + Unfortunately you can only feed it
    + not direct it
```

So let us talk about the background mind. I am not going to directly equate it with the sleeping mind, but the sleeping mind is the number one instance of the background mind. You can find access to your background mind during the day while you are awake, but it is tricky.

It is good at making connections. The kind of thing like, "If I make a hut out of mud and it rains hard, it will disintegrate" is not necessarily the kind of thing that you can tactfully figure out. Your background mind is going to know aspects of all those different components and make the connections and synthesize them.

Even when you think you are really hot at making decisions on the fly, you are almost always just regurgitating something your background mind has already figured out. So the background mind is good at synthesizing things. It is good about strategy.

And so when Mark talks about abstractions and things like that, abstractions are software strategy. Because the idea there is you are making some super global decision that is going to need to be correct in a whole bunch of contexts in which you cannot make tactical decisions yet.

What does it mean to make an abstraction you are going to derive libraries from? What does it mean to put something in a programming language where I had no idea what you guys are going to do with it? It is a more strategic kind of thing. You do not build programming languages and say, "How will this programming language deal with HTTP requests?" What you want to do is give Mark something that he can use when he has got a tactical decision to make about HTTP requests.

And that is a strategic kind of thinking, and your background mind is good at strategic thinking. If you want to do abstraction, you have to find time to do this thinking, because that is the part of your brain it comes from. It does abstraction. It draws analogies.

I think this is where you solve most non-trivial problems. You can make good decisions in the moment otherwise, but if you are really trying to solve something hard, you have got to engage the other half of your head.

[Time 0:25:48]

```
slide title: Scientific American said:

+ As we snooze, our brain is busily processing the information
  we have learned during the day.
```

```
+ Sleep makes memories stronger, and it even appears to weed out
  irrelevant details and background information so that only the
  important pieces remain.

+ Our brain also works during slumber to find hidden relations
  among memories and to solve problems we were working on
  while awake.
```

So I am not just saying this. Scientific American

[Audience laughter]

said that when we are sleeping, we process the information during the day because that is pretty obvious. But that sleep reinforces memory, which is good. I mean, it is important to remember what you are working on.

But more importantly, it is a great sorter out of things. So we had this whole... I just advocated taking a lot of input. Taking a lot of input, doing all this analysis of the requirements in the space, doing all the reading, looking at competitive solutions and tearing them apart. There is this ton of stuff.

When are you going to decide what about that is important, and what is not? When you are asleep. That is what happens. Evolution has solved this problem for us, and that is the solution it came up with. We cannot ignore it. We have to use it.

But the most critical thing is this one: finding hidden relations, and solving problems we were working on. So imagine somebody says, "I have this problem of this, that, that, that, that, that." And you look at it for 10 minutes and say, "Okay, I am going to go out to the movies and do something else or whatever." Then you go to sleep. Are you going to solve that problem in your sleep?

[Audience member: Sure.]

No.

[Audience laughter]

And you did not think about it, did you? No, you did not think about it. You did not think about it hard enough while you were awake for it to become important to your mind when you are asleep. And this goes back to that feeding your background mind thing. You really do have to work hard: just think, not typing it in. Just thinking about a problem during the day, so that it becomes an agenda item for your background mind. That is how it works.

It is when people are out there and they are like, "Oh my God. How am I going to find food, and this is happening there, but I know I saw elk over there, and they seem to be by the water sometimes, or whatever." That is when you wake up as a caveman and say, "let us go hunt for the animals by the water." It is not a logical deduction. It seems like that when your foreground mind is analyzing it, but there is no logic for that necessarily. It is really a process of this very parallel kind of thinking. So this is very important.

[Time 0:28:19]

```
slide title: Loading It Up

+ 7 +/- 2 problem
  + Many designs > 9 components
+ Write proposed solution down
  + Yes, with pictures maybe
```

```
+ Go over and over, different orders
+ Hammock time is important _mind's eye_ time
```

So we have a problem, in general, because we are just being asked to write software that is more and more complex as time goes by. And we know there is a 7 plus or minus 2 sort of working memory limit. And as smart as any of us are, we all suffer from the same limit.

But the problems that we are called upon to solve are much bigger than that, normally. So what do we do? If we cannot fit the whole thing in our head at the same time, how can we work on a problem with more than nine components?

What I am going to recommend is that you write all the bits down. Especially now, you have written a lot down about the problem. You know what the problem is. You know a lot of facts about it. You know the constraints about it, where it runs. You know what you don't know. You have asked yourself those questions. You wrote them down. "I wish I knew blah." You looked at competitive things and said, "That works great over here, but that part of that competitive thing sucks. I hate that. I wish that was not there."

You gave this huge agenda to your background mind. And when you are trying to load it up, you need to survey it, and that is the point of writing it all down before. If you have written all the stuff down, including some sketch of how you want to solve the problem, you can go and just sort of jump around and look at that. And it is sort of like, how many balls can you juggle? Well, you can only juggle so many. Well, I can't juggle at all. But if we look at the 7 plus or minus 2 thing, we are going to say we can juggle seven to nine balls.

But if you can imagine having an assistant who every now and then can take one of those out and put a different color in, then you could juggle balls of 20 different colors at the same time as long as there are only nine in the air at any one point in time.

And that is what you are doing, you are going to sort of look around at all these pieces, and shift arbitrary shapes of seven into your head at different points in time. Maybe you will draw pictures. Don't use a UML tool to do this.

[Audience laughter]

There. It is not a methodology. So go over and over. But then, you must again step away from the computer.

[Time 0:30:36]

There is another really important part of doing this, which is to go and sit somewhere and have no input, and close your eyes, and not go to sleep. Close your eyes, because we have this other thing. Everybody knows what it is. It is really hard to describe it. Does everybody have a concept of their own mind's eye? What you see when you close your eyes and you start thinking about something.

It is this weird, I mean, it is not actually technically visual, though some people are really vibrantly visual. I know, for me, it is... I do not know. I cannot describe it but it is not very realistic.

But you need to do that. That part is important for your brain, because at that point, you are switching out of sort of an input reception mode. If you are just look at your list, you are sort of in the mode of, "I am getting input." But when you are sitting and contemplating something, and hashing it over in your head, you don't have any other input, which means you are exercising the recall. I have looked at those 20 points. Let us say it was just 20. I have looked at those 20 points over and over and over again. And I jumped around, with input, between them.

Now, I close my eyes and I am trying to recall them and think about them a little bit more in my head, and you are going to find if you have done the last step, going over and over, you will actually

be able to sit on a hammock and pull all the different parts of a fairly large problem in and out of your head, admittedly maybe one at a time, and think about them that way.

That exercise is really, really important. I do not know why, it just is, because it forces this recall thing that definitely makes those things agenda items for your background mind. So we will call that "mind's eye time".

[Time 0:32:22]

```
slide title: Wait for it ...

+ At least overnight
  + sleep sober for best results
+ Sometimes over months
  + so, work on more than one thing
    + not interleaved within a day, but over the course of time
  + Switch around when stuck
```

Now you are done. Cake is in the oven. You just have to wait. It is *so* good.

[Audience laughter]

One of the things I would say is: at least wait overnight. No matter how, you and your buddies talked about it, and you just feel like such a hotshot today, "I have *got* this thing." Sleep on it at least one night. At least, if it is an important decision.

[Audience laughter, probably in reaction to Rich revealing the line "sleep sober for best results" on the slide]

Now, how many people woke up this morning with the answer to a hard problem?

[Audience laughter]

You see? It is science. Science at work. No, it is really kind of an unfortunate thing. If you are not thinking about this, you think, "What happened? I worked hard all day." Like, "I am done working. Time to relax."

Unfortunately, if you believe in what I am saying today, you are actually doing something kind of important when you are sleeping. So occasionally, you really have to give your brain a chance to do that other part of the job. If you always deny it, I do not think you are going to have the best results.

Unfortunately, sometimes overnight is not enough. Some big problems, especially finding really good abstractions, or finding answers to things that satisfy a bunch of simultaneous constraints, take a long time. It just does. And I know, everybody has to ship and everything else. In that case, a lot of what I am saying does not apply. And like I said before, I consider it a *huge* opportunity when I get an extended amount of time to think about a problem, because I know I will come up with a better answer.

But one of the ways that you can deal with this and not get stymied by, "Well, let me just think about that for 3 months." – because most managers are not incredibly receptive to that sentence

[Audience laughter]

– is to just work on more than one thing, *not* inside one day. Try to work on *one* thing each day.

But if over the course of time, you have like three projects, it is quite possible to load one up and work on it for three days. And find that you are not finding answers to any of your question mark items, or able to enumerate new possibilities, so you are kind of stuck a little bit. Just switch to another project and do that for a few days.

You have to amortize the loading up time. It can take between an hour and an entire day to load up something, so once you have done that, you try to get at least the rest of the day, or three days, or more on it. But do not get hung up about the stuck thing. Just switch. Don't stay stuck. Switch. Or get more input. Talk about it more. Keep stimulating the pathway. Do not stay stuck on it.

[Time 0:35:22]

```
slide title: Wake up Working

+ Eureka!
+ Sometimes with answer to something else
  + that has to be ok
  + be ready to capture that
  + or even switch over for the day
```

But yeah. Then eventually, the cake comes out of the oven. You wake up and you have a great idea. You think you know the answer to your problem, or you have a good idea for a solution.

Unfortunately, sometimes you have an answer to not the problem you were working on. You were working on three projects and you loaded up Project C, and you woke up with the answer to Project A. That has to be okay. Just switch and take advantage of it. At least capture it. If you wake up to an answer with some other thing that you cannot work on that day, capture the results of this background process. They are really useful.

[Time 0:36:02]

```
slide title: Try It

+ Eventually, coding is required
+ Type sparingly
  + with confidence
+ Feedback loop is important
  + but don't lean on it
```

Finally, you do have to take your great ideas and figure out if they are actually great by either analyzing them more, which is certainly important, but sometimes, you have to write them and type them into your computer. Actually, we all have to do this. So you do, eventually, have to code.

It is fun, Stu has this great – he has seen some of my design sheets [motions of folding something open]. What does he call it, "document of despair" or something, he calls it. It seemed to be all like, we can't do this. This does not work like that. Question marks, blah, blahs. This other thing was just like ... It is all negative.

But it is all challenges to the problem solving process. It is not despairing. It is positive. It is saying, "I know what my challenges are, and therefore I can work on them." But you spit this thing out, now, you have something so you try to ... Try to avoid a lot of typing. I know I do. Because if I think that I have got an answer and the answer is small, that is one of the most telling attributes that it is probably good.

And what I would hope from doing this whole process is that you gain confidence in it after you have seen it work for you. So that you say, "You know what, I have never done this before but I really have thought about it, and this solution I came up with overnight feels awesome, and wooh! Let us go."

It is important to look at what you did and to run it and see, and find out new things about the solution and say, "Oh, you know what I had this supposition. It is not correct. I thought it would

have this characteristic. It does not, etc., etc." I am not advocating the Waterfall Model. You are going to try stuff and go back. That is fine. But do not lean on this. Test-driven dentistry, I do not think I could come up with a better thing.

[Audience laughter]

We cannot really do that.

[Time 0:38:04]

```
slide title: You Will Be Wrong

+ You will think of better ideas
+ "When the facts change, I change my mind.
  What do you do, sir?"
  + John Maynard Keynes
+ Oops
+ That's all ok
  + Don't be afraid
```

The last thing is: you are going to be wrong. I am frequently wrong. That is part of the game. You are going to think of better ideas. I think that is one of the most exciting things. I think no matter what I have ever thought of, the fact that I know I am going to think of something better – as much as it will suck a little bit because I know I delivered something that was not the best – means that I am still going. It is still working. So you will think of better ideas.

Also, the facts change. It can change because of two reasons, right? One, you missed some of them early on. So they are new to you, because you skipped them. What else do we have? Changing requirements. It is just, we all know this.

The facts will change. When the facts change, do not dig in. Do it over again and see if your answer is still valid in the context of the new requirements, the new facts. And if it is not, change your mind, and do not apologize.

Sometimes, you will just make mistakes, errors in logic, or you just get it wrong. That is fine. If I could advocate anything, it is "Do not be afraid" especially, "do not be afraid of being wrong."

[Time 0:39:28]

```
slide title: Summary

+ It's a rant!
  + it doesn't have a summary
```

So in summary, this was a rant. There is no summary.

[Audience laughter, then applause]

[Time 0:39:32]