The Value of Values

- **Speaker: Rich Hickey**
- **Conference: JaxConf 2012 - July 2012**
- **Video: https://www.youtube.com/watch?v=-6BsiVyC1kM**

So The Value of Values.

Who here works in I.T.? Everybody, I think. What does that mean? What does that stand for, I.T.?

[Time 0:00:24]

```
slide:

I.T.
```

Information Technology.

[Time 0:00:27]

```
slide title: Information

+ _Inform_
  + 'to convey knowledge via _facts_'
  + 'give shape to (the mind)'
+ _Information_
  + the facts
```

Yeah, or other things. So what do we mean when we say information?

As you may know from my other key notes, all I do to make a key note is I use my dictionary, because my dictionary has all the answers. And my dictionary says: information is based on the word "inform", which means to convey knowledge via facts, and the purpose of that is to give shape to the mind. And information is just those facts. That is what information is. Information is facts.

[Time 0:01:04]

```
slide title: What is a Fact?

+ _Place_ where specific information is _stored_
+ There is a place for every piece of information
+ Facts have _operations_, e.g. _get_ and _set_
+ Operations control how facts can _change_
+ To convey a fact, convey its _location_
```

So what is a fact? Well, a fact is a place where information is stored. And what is great about that is that there is a place for every piece of information. And facts have operations like get and set, and they may have other operations. Those operations control how facts can change.

The other great thing about facts is that it is easy to convey them. To convey a fact, we just convey its location.

How many people are uncomfortable right now?

[Time 0:01:45]

```
[ Now a large red circle with a red slash through it meaning "NO" has
been superimposed on the previous slide. ]
```

I am. This is not true. This is wrong. Everything that I just said was wrong. Dead wrong. And you *should* be uncomfortable. That is not what facts are.

[Time 0:01:59]

```
slide title: Place

+ 'A particular portion of space'
+ 'An area used for a particular purpose'
+ Memory address, disk sector
```

So are facts places? Well, what is a place?

Again, back to the dictionary. A place is a particular portion of space. And space is an important word. We are going to dig into it later.

In particular, the part about place that matters is the fact that it is sort of delimited. There is a specificity to the location named by a place or vice versa.

We know about places. We have good examples of places in our everyday work and programming. Memory addresses are places. Disk sectors are places. They have addresses, and we can go to them, and we can substitute their contents with other contents. And we are very familiar with this notion of a place.

[Time 0:02:51]

```
slide title: 'Information' Systems?

+ In memory
  + mutable objects are abstractions of places
  + objects have _methods_
+ In durable storage
  + tables / documents / records are places
  + DBs have _update_
```

But I think it is quite important to think about whether or not, with these memory addresses and disk locations, we are actually building information systems that are about information as we just defined it.

In particular, we use memory and we use objects typically, especially at this conference. And mutable objects are nothing more than abstractions over places. They are little abstractions that are over places, especially or in particular mutable objects. And they have methods, just like the thing I talked before about facts having methods. Objects have methods. This is the core thing we are building systems out of.

On the other side, on the durability side, we have the same kinds of notions. We have tables and documents and records, all of which are places, all of which have an update notion. The notion of going to the place and setting it to be a new value.

These are the underpinnings from which we build systems, but we are sort of making abstractions on top of them that do not hide what they are about.

[Time 0:04:05]

```
slide title: PLOP

+ PLace-Oriented Programming
+ New information _replaces_ old
+ Born of limitations of early computers
  + small RAM and disks
+ Those limitations are long gone
```

And I have a word for this, or a term for this, which I call "PLOP". PLace-Oriented Programming, which is what this is. And you can tell when it is going on, because any time new information replaces old information, you are doing Place-Oriented Programming.

There is a reason we do Place-Oriented Programming: because way back in the early days of computers, we *had* to do Place-Oriented Programming. I saw Guy Steele give a great talk where he was talking about building these systems on a computer that had four kilowords of memory.

And every piece of memory, you knew the address, you knew the even numbered addresses starting at 2000 were used for a jump table, and these other addresses over here were used for data, and other parts of the addresses were used for code. And sometimes they were used for more than one thing because you knew, like right now, no one is using this for code, so we can use it for data, and vice versa. You *had* to do it. There was not enough capacity to do anything else.

Computer memories were really small. Disks were very small. Everything was very expensive. So we adopted an approach to programming that was based around the manipulation of places. It totally made sense. And the key word there, the key aspect of that is: it made sense. It used to make sense.

Those limitations are gone. In the time that I have been doing programming, the capacity of these two things have increased a million-fold. A million-fold. What other thing in life has the same facts about it remain true when the size of something changes by a million-fold? Imagine if your car was a million times bigger than it is. What rules would still apply? What characteristics would still be true? Almost nothing. But yet, we are retaining decisions we made when things were much much smaller, and moving forward with it.

So why does PLOP still rule? That is the key question here.

[Time 0:06:10]

```
slide title: Memory and Records

+ We've co-opted
  + and believe our own myths
+ Mental memory is _associative_ and _open_
+ Real records are _enduring_
  + and _accreting_
  + not erase and overwrite
```

When we talk about Place-Oriented Programming, we often talk about these two things: memory and records. These words had meanings before we had computers, yeah? And we have taken them over. We have said, "Memory means addresses in RAM chips, and records mean slots in database tables."

And worse than just taking over these words, because obviously there is a limitation to a number of words, and the analogies roughly hold, right? The analogies between memory, and memory in your head, roughly hold. But the problem is, we have now been doing this for so long that we believe our own myths about what these things mean.

But we should go back to the facts of memory and records. The fact of memory is that it is an open system. If your friend gets a new email address, that does not go into your brain and find your friend's email address cells, and replace his email address in that part, in those neurons in your brain. That is not how brains work. That is not how memory works. Memory is essentially an open system, and it is an associative system. It is not an address-based system.

Same thing, record keeping. We used to keep records before we had computers. What did we do? We took out the stone tablets and we chiseled the thing, or we took out the papyrus and we wrote stuff down. When we had new information, what did we do? We did not go and smooth over the marble and chisel new stuff. We did not go to the papyrus and take out erasers and things like that.

And people built accounting systems before there were computers, and they did not use erasers either. What did they use? Double-entry accounting or ledger-based accounting. They only made correcting entries. They did *not* go back with erasers. That is not the way things worked before we had computers.

[Time 0:08:02]

```
slide title: Value

+ 'Relative worth'
+ 'A particular magnitude, number or amount'
+ 'Precise meaning or significance'
```

This talk is about values. It is another term we should define, and again, we just go back to the dictionary. And there are some very interesting definitions for value. First is "relative worth". And relative ends up being a very critical aspect of values, because the next thing is the one we are probably most familiar with, especially in computers, because this next definition is the one that applies to like 42, right? It is a magnitude. It is a number. It is something we use to measure something else. And that notion of a value, I think, is the one we are most readily able to adapt.

But again, the bigger notion of value is about meaning, and about comparability, and about relative worth. That is the bigger notion of value, because when you measure something, you have to measure it in terms of something else. There are no absolute measurements. The comparing part is important.

[Time 0:09:01]

```
slide title: Is a String a Vaue?

+ Is it immutable?
+ Equality, comparability are basis for logic
+ Who wants to go back to mutable Strings?
```

So the question right now is, "Is a string a value?" How many people think it is? I love doing the answer the questions by raising your hands because not everybody has a hand, apparently. How many people think a string is not a value? How many people think it depends? You *always* wait for "it depends," right? It is the best answer, so you hold out for it. What does it depend on?

Is it immutable? If the string is immutable, it is a value. If it is not immutable, it is not a value.

How many people worked in programming languages, or still do, where strings were mutable? How many people want to go back? No. We do not like this.

And it ends up that this equality notion of value ... Because right now, string does not measure something. It is not a magnitude or an amount. It does not correspond to that definition of value.

But it ends up that an immutable string is a comparable thing. And comparability is where we derive our ability to do logic, and to make decisions. Until we can say, "Is this the same as it was yesterday? Is it greater or less than what it was? Does this string label this thing correctly according to my understanding of it?" Or anything else we do with information. That comparability and equality test is sort of at the bottom, and that goes back to that other notion of value. So we really do not want to go back.

[Time 0:10:33]

```
slide title: Programming Values

+ Immutable
+ Are semantically transparent
```

So now we want to sort of talk about values, not from the dictionary definition, but very specifically in terms of what we do in programming. And there are lots of nuances to this, but for the purposes of this talk, given that it is a half hour, I am just going to focus on two. Values are immutable. And values are semantically transparent.

In other words, the purpose of a value is to expose itself to you so you can do that comparison and that equality test. Value is not about encapsulating something in methods, and doing things. A value is about saying, "Compare me to something else. I am telling you what my precise meaning or significance is." Right on the label, right on the outside. That is what a value is about.

[Time 0:11:21]

```
slide title: Value Propositions

+ Values can be shared
+ Reproducible results
+ Easy to fabricate
```

The reason to give this talk is to talk about sort of the value propositions of values. What makes values good? And originally, this talk was an hour long so this is going to go a little bit fast. But there are a bunch of characteristics of values that are valuable.

The first is that values can be shared. If you have a value, if you have an immutable thing, can you give that to somebody else and not worry? Yes, you do not have to worry.

Do they have to worry about you now, because they both now refer to the same value? Anybody have to worry? No. So values can be shared. That is very, very valuable.

Values support reproducible results. If you define a function in terms of values, every time that you call that function with the same values, will you get the same answer? Yes. If you define a function in terms of places, every time you are on that function, will you get the same answer? No. It depends what is in the place.

Reproducible results really matter. They allow you to run tests, and reproducible tests. So many people running tests now of places. They cannot tell you anything. They do not tell you anything. They are all conditional upon your ability to bring that place back to the same place, to the same value.

Another critical aspect of values is that they are easy to fabricate. You can make up a value from scratch in any programming language quite readily. Can you make a string in any programming

language? Yeah. Can you make a list of strings? Sure. A list of numbers? Yeah. A list of a list of numbers? Yeah. A map of a list of numbers? Yeah.

Can I make an instance of your fancy interface in any other language? No. It is not easy to fabricate that. It is not easy to make programs that write programs. It is not easy to make programs that write tests, if your programs are not based around values. So the fact that values are easy to fabricate is important.

[Time 0:13:16]

```
slide title: Value Propositions

+ Language independent
+ Generic
+ Values aggregate to values
```

All right. Values are language independent. So I just talked about that already. The notion of a list or a string or a number or a map. This has nothing to do with the programming language. Those things I just said. Nothing at all. Nor any of the aggregates of those things. It has nothing to do with that. They are generic. These ideas, the notions of values, are generic.

And I think it is something that we do not think about often enough in our programming designs and our systems – about the actual costs of specificity. We *love* specificity. We use Java. Every new idea gets a new class. Every new thing gets a new thing.

What does that cause to happen? You get this explosion of code. Just a tremendous explosion of code. Objects were supposed to support reuse. They have done the exact opposite thing, especially in typed languages. You get very little reuse, because you make a new thing every time. And what does more code mean? More code equals more bugs. Right away.

Another interesting thing about values is: values aggregate to values. And this is something I really want you to focus on. We talked about 42. We talked about a string. We talked about a sort of atomic things. But a list of immutable things is itself an immutable thing, and so on and so forth. So as you build aggregates, those bigger things are also values.

And I think that we tend to stop. We say, "Of course, strings should be immutable. But an immutable collection? Ugh! Oh, I cannot even comprehend it." But you *should*. It has all the same desirable attributes that that immutable string did. Nobody here wants to program with mutable strings anymore. Why do you want to program with mutable collections anymore? You should not. And there are really important benefits you get from doing this.

For instance, compare it to objects. If you have an object and you want to share it, what do you have to do? You have to define the object, define an interface for it, and everything else. But then you have to do what? What if it is in a concurrent environment? What do you have to have for that object? Some sort of locking policy, right? Very, very difficult thing. In fact, a lot of languages do not give you any resources for defining it well. It is like there is this napkin and on it, we have defined how we are going to use this object. But if you have done that ... And there is also other kinds of problems like: how do you copy it? What are the cloning semantics? Whatever.

Let us say you have done that work, and you have done that work for every one of your classes. And now, you build something that is a composite of those things. Do you get a lock policy from combining them? No. It all falls away. You have to do it all ... do over again. This composite of all these things, for which I have lock policies, now no longer has a lock policy. I have to come up with a new one. I have to come up with a new cloning policy, and everything else, over again. Values aggregate to values. All the benefits apply to aggregates.

[Time 0:16:15]

```
slide title: Value Propositions

+ Conveyance
+ Perception
+ Memory
```

There are other benefits you get from values, and we see these all the time. Values are easy to convey. If I have some piece of information I think is useful, I can send you that value and I know I have communicated to you what I was seeing.

If I see something interesting and I communicate to you the place where I saw it, what have I actually conveyed to you? Nothing, right? Not the information, that is for sure. Because you go look at that place and see something totally different from what I saw.

It works the other way as well. When I want to perceive something, if it is a value, I can take my time and look at it, especially if it is a set of values, if it is a composite thing.

If I want to perceive something that is based around places, how do I do that? There are a bunch of places I want to sort of look at it. What do I have to do? I have to stop the world. Please stop while I look at these places, because otherwise, I will look at one and as I turn my head to the other, something will have changed and, by the time I sort of perceived the whole thing, I do not know that I had anything consistent. I do not know that I am making a decision based on anything consistent.

And this also goes to memory, right? How do you remember anything? If you encounter an object during the course of your program running and you want to remember it, what do you do? Do you just store the reference. Not good enough, right? What do you have to do? Clone it. How do you do that? It depends. Exactly. It depends.

[Time 0:17:47]

```
slide title: Value Propositions

+ Values make the best interfaces
+ Reduce coordination
+ Location flexibility
```

So the other thing I want you to do is start thinking bigger, bigger than your box, bigger than your process. Because these value propositions extend to your systems. And in particular, values make the best interfaces.

Now here, I do not think we have any arguments. I think we are already doing this. What do you send around on the wires? Anybody still using CORBA or DCOM? No, no. They died for good reasons, right? We now use values. We send around JSON or XML, if you have to, but both are representations of values. And that allows us to build interfaces to things that are easy to change on both ends.

The other aspect of values that is very interesting, especially in the large, but it is also true in the small. We talked about how do I perceive something: I have to lock the world down. That applies in the large as well. How many people have ever heard of read transactions? Yeah. How many people like read transactions? No. The whole idea is counter-intuitive and violates physics. In physics, we just look at each other. We do not have to stop everything in order to look at things. So when we are programming with values and using values, especially in storage, we again have reduced coordination.

Another benefit we get is location flexibility. If in the small you build a system, and the system is defined in terms of processes that are communicating values, and you decide, “You know what? That part of the system, I want to re-write that in a faster language, or I want to run that on a different box.”

Is that straightforward to do if you are using objects and specific things related to your programming language? If I am passing Java interfaces, is it easy for me to re-write... or let us say Ruby interfaces, is it easy for me to re-write that in Java? No.

We know, we do not do this in the large. In the large, we do not do this. In the large, we communicate values. But in the small, in a programming language, we start doing icky things. And those icky things keep us from being able to move stuff. We cannot move it to another thread. We cannot move it to another language. We cannot move it to another box. But if we are using values, we can. That I would call location flexibility.

[Time 0:20:09]

```
slide title: Facts are Values

+ Not places
+ Don't facts change?
  + _No_ - they incorporate _time_
+ _Fact_ - 'an event or thing known to have happened or existed'
+ From: _factum_ - 'something done'
```

So let us get back to information technology, and back to facts. The first fact about facts is that facts are values. They are not places. That slide up front was a lie. And everybody is sitting there saying, "But don't facts change? Don't we have a president at one time, and then we have a new president?"

No! They don't, because facts incorporate time. How is that? What does that mean? Again, the dictionary knows everything because what does "fact" mean? Fact means something that happened, something known to have happened or existed. It comes from Latin. And it comes from a past participle in Latin that means: something that happened.

A fact is something that happened. It is not a place. It is not something you can change. Bill Clinton was president. The fact that he was president will *always* be a fact. We can have new presidents. That is a new fact, just like you can have new email addresses.

[Time 0:21:16]

```
slide title: Facts != Recent Facts

+ Knowledge is derived from facts
  + Comparing
  + Combining
+ Especially from different time points
+ _You cannot update a fact_
  + any more than you can change the past
```

The other thing about facts is that it is very important when you consider facts that it is insufficient for you to consider recent facts. And again, we will go back to the whole point of information. Information is to inform, so that people can convey knowledge. But knowledge is derived from facts. When we try to make decisions, we compare times, we compare two different things, we combine facts.

And especially we use different time points. Imagine if you only knew the present value of any property or attribute in the world. How good would your decision making capability be? It would be awful. It would be absolutely terrible. And yet, we are building systems that only know the most recent set of facts. They do not know anything else. But we are supposed to be making information and decision making support systems.

So the bottom line is that you cannot update a fact. And the reason why you cannot update a fact is because you cannot change the past. And that is what facts are. They are the documentation of the past.

[Time 0:22:27]

```
slide title: Information Systems

+ Are fundamentally about _facts_
  + Maintaining, manipulating
+ To give users _leverage_
  + Making decisions
+ Systems should be _value-oriented_
  + Don't use _process_ constructs for information
+ _Place has no role in an information model_
```

So let us go back and sort of revisit: what would it mean to build information systems that are about information? It would mean that the systems would be fundamentally about facts. It would be about maintaining facts, manipulating facts. And presenting facts to users to give them leverage so they can make decisions. Now, we think we are doing this. When we are in information technology, we think we are building systems that are decision support systems. But we are not using infrastructure that is fact-oriented.

In the most bottom level notion, before you even get to the temporal aspect of facts, we should certainly be building systems that are value-oriented. Now, that is not to say that programming languages should not have process-oriented constructs. Of course, they should.

But we do not distinguish them. If you look at any program, it is going to have two different parts. There is going to be the part of your program that is sort of like a machine. It has got a loading dock and stuff comes in, and you put on a conveyer belt, and it moves, and then it gets sorted. It gets split, and some stuff goes over here, and some other stuff goes over there.

All programs have this aspect to themselves which is process-oriented, and it is sort of like a machine. And we build programming languages that allow us to use constructs that are analogous to little machines. They do stuff.

The problem is: we apply that technology to everything, including to information. And information is not a little machine. It is not at all like a machine, and so you have to separate out these process constructs.

And in particular, one of the big take-aways should be that place, that all of your constructs related to place, have *no role at all* in an information model. They are artifacts of the way computers work. They have nothing to do with what your software is supposed to be accomplishing, if your software is supposed to be accomplishing information management and decision support.

[Time 0:24:27]

```
slide title: Decision Making

+ We know what it takes to support our own decision making (hint: _information_)
+ Compare present to past
+ Spot trends, rates
+ Aggregates
+ Often requires _time_
```

So one of the great things about this talk is: I think you all already know this. Because we do decision making. We know what it takes to support our own decision making. We are constantly comparing the present to the past. We are trying to spot trends. We are trying to see the rate of change. We need to add stuff up that has happened over time. We almost always need a temporal aspect.

[Time 0:24:54]

```
slide title: Programmer I.T.

+ Source Control
  + Update in place? - No
  + Timestamps - of course!
+ Logs
  + Update in place? - No
  + Timestamps - of course!
```

How can I tell you I know this is true? Well, it is really straightforward. We are programmers. We have stuff to do. We have our own little businesses. We make stuff. What do we make? What is our concrete artifact? We make source code. And then, we have an operations aspect to what we do. What do we do? We run our programs. And we maintain information about both these things. Let us look at programmer I.T. Let us look at our own I.T. systems, the ones we build for ourselves.

Source control. Is it update in place? How many people use a directory on the file system for source control? And when you have a new edit to your program, you just put it in the directory? You really do not want to raise your hand. [Laughs]

No, of course not. It is not update in place. Anybody use a source control system that does not have dates and times on stuff? No, of course not. How would we possibly know what we were doing? How can we possibly make the decisions we need to do to run our little software business if we did not have this information? No way.

OK, what about our operations I.T.? We run our systems, what do we do with our running systems? We log. We log and log. We keep track of everything that is happening. Anybody use an update in place log? Anyone use a log that says: the last latency was five? Yeah? No. Anybody use a log that does not put time stamps on every entry? No, of course not. How could we possibly understand what our systems were doing if we did not have this information? How could we make decisions? We know there is no way. No way. So we do not do this.

[Time 0:26:36]

```
slide title: Big Data

+ Business to programmers:
  + _"I like your database better than the one you gave me"_
+ Logs have all the information
  + and timestamps
+ We are reactive here
  + mining logs, seriously?
+ Not delivering leverage
```

So big data. It is all connected. What is big data? I will contend that a certain percentage of big data is this. It is businesses telling programmers: That database you have, I like better than the one you gave me. Because the one that you gave me only remembers the last thing I told it. And your

database, your logs, they know everything that happened in our business. You know everything. You have times for everything. Let us go mine that, find some good business decision making information. Because my database, it does not have it. It does not remember anything other than the last thing. The logs have all the information. The logs have the time stamps.

But I think it is really sort of a shame here, because we are being reactive in this area. Mining logs? We know much better ways to represent information. We know how to store information in ways that is leverage-able. Logs are not that. MapReduce over logs are not that. Logs are full of crap, too, right? They are full of stuff about the program operation itself. There is business value information in there. We have not built databases to store that yet, but that is sort of our fault.

[Time 0:27:46]

```
slide title: The Space Age

+ _Space_
  + 'The unlimited expanse in which all things are located, and all events occur'
+ If _new_ never fails ...
  + you are effectively running in space
+ If _S3_ never fills up ...
  + it is not the _cloud_, but space
```

I think we are moving into what I would like to call The Space Age. One more definition: space, the unlimited expanse in which all things are located and all events occur.

What is really interesting about the definition of space is that it has always, if you go back to the oldest definitions in the oldest languages, the definition of space has always incorporated both place and time. It has never been something that applied only to one or the other of those two things. It has always connected to two, and there is a certain physics aspect to that.

So if "new" never fails, if you can call "new" day after day, 24/7, over and over and over again, you are not running in a place. You are running in space. You are not seeing the edges of things. You are not delimited. You need new stuff? You get it.

If S3 never fills up, that is not the cloud, that is space. If every time you want to put a file on S3, it says, "Sure!"

[Time 0:29:00]

```
slide title: New Facts, New Space

+ The end of PLOP
+ If you can afford this, why do anything else?
+ You can afford this
  + _(there will be garbage)_
```

Are you worried about the edge of it? I mean, most of the times, we do not worry about the edge of space. I mean, maybe you do, but I do not.

So information systems should have a different approach. They should say that new facts require new space. This should be the end of Place-Oriented Programming. We have had the facilities to do this for a very long time.

If you could afford to do this, why would you do anything else? Does anybody feel like, "I wish I could log but I cannot? Because my disk is only 5 megabytes." Anybody ever have a 5 megabyte disk that cost thousands of dollars? That is not that way anymore. So you can afford to do this.

Lots of interesting things will happen when you take this approach. A lot of the things that happen in memory with garbage collection are going to happen with storage. There will be garbage.

[Time 0:29:53]

```
slide title: Summary

+ We continue to use place-oriented programming languages and databases
  + _and make new ones!_
  + long after rationale is _gone_
+ We are missing out on the value of values
  + which we recognize
+ We need to deliver _information_ systems
  + demand is clear, resources available
```

But that is OK. These are things that we are learning about how to manage.

So to summarize, unfortunately, I think we continue to use Place-Oriented Programming, and the rationale is gone. And the sadder thing is, we continue to make *new* things that are like this, brand new programming languages and brand new databases, that still take this approach, still use a place orientation that has been invalid for at least a decade, but certainly for the last five years. The rationale is gone.

We are missing out on all those benefits I listed before. And I could talk to you about any one of those benefits for an hour. There are a *huge* number of benefits to using values. We recognize them. Look at our information systems that we use for ourselves. We are proving we already know this. We do not overwrite our logs. We do not overwrite our source control. We are already... we are in the space age for ourselves, but we need to be in the space age for the businesses we are supporting.

There is definitely demand for this, this whole big data push and mining logs and everything else and tracking everything and keeping the time of everything. Businesses know there is tremendous value here. They want it. The demand is quite obvious. We have the resources to do it, and I think that the challenge is to make sure that we do. What I would recommend is that you try to take an information-oriented approach to the way you build your programs.

Thanks.

[Audience applause]

[Time 0:31:30]