Languages are in the Eye of the Beholder

Clemens Szyperski

Development Manager in Data Platform Group

Microsoft

Wirth 80 Symposium, ETH Zürich, February 2014

Driving your own car, anyone?

Having a chauffeur was more than a luxury. It was a necessity. So many things could go wrong, requiring a technician's skills.

And it limited who could afford to own and use a car.

Self-Service Revolution

"The worldwide demand for cars will not exceed one million – even if just for a scarcity of available chauffeurs."

Gottlieb Daimler, Inventor, 1901

Technology Revolution

As any technology matures, capabilities that required genius-level skills in one generation become common-place in the next.

"... all large scale applications of LSI* chips are by definition highly suspect. That does away with 'personal computing', 'home computers', 'the information society', and all that jazz."

Edsgar Dijkstra, 1978
(EWD691 "On improving the state of the art")

* LSI = Large Scale Integration

Sticking to the technology status quo ?

"There is no reason for any individual to have a computer in his home."

Ken Olson, Founder and CEO of Digital Equipment Corp, 1977 at Convention of the World Future Society

DIY !

At work and at home.

** Do It Yourself*

“A computer on every desk and in every home.”

Bill Gates and Paul Allen, Microsoft Vision Statement, 1977

Programmer

A person skilled in designing and developing programs.

The chauffeur of your computer!

- Programmers write solutions (programs) in a programming language.
  - Requires intersection of programming skills (how?) and domain knowledge (what?).
- Programming languages themselves are the subject of a design activity.
  - Facts and opinions abound: usability, expressiveness, correctness by construction, readability vs. writability, simplicity, style, …

Properties of Programming Languages

Tools need to match the problem space, the audience expected to use the tool, and the expectation space of the desired outcome when using the tool.

- Read-Only Languages
  - SQL (Structured Query Language) – many learn to read SQL, only a few can write non-trivial SQL
- Write-Only Languages
  - Pearl – many learn to write scripts, but most cannot even read what they wrote themselves a day ago
- Impedance Mismatch
  - "Ceremony" or lack of expressiveness force cumbersome formulations of solutions in a given problem domain
- Requirements Mismatch
  - Functionally good expressions end up failing expectations of performance, security, etc.

Law of the Instrument

"I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail."

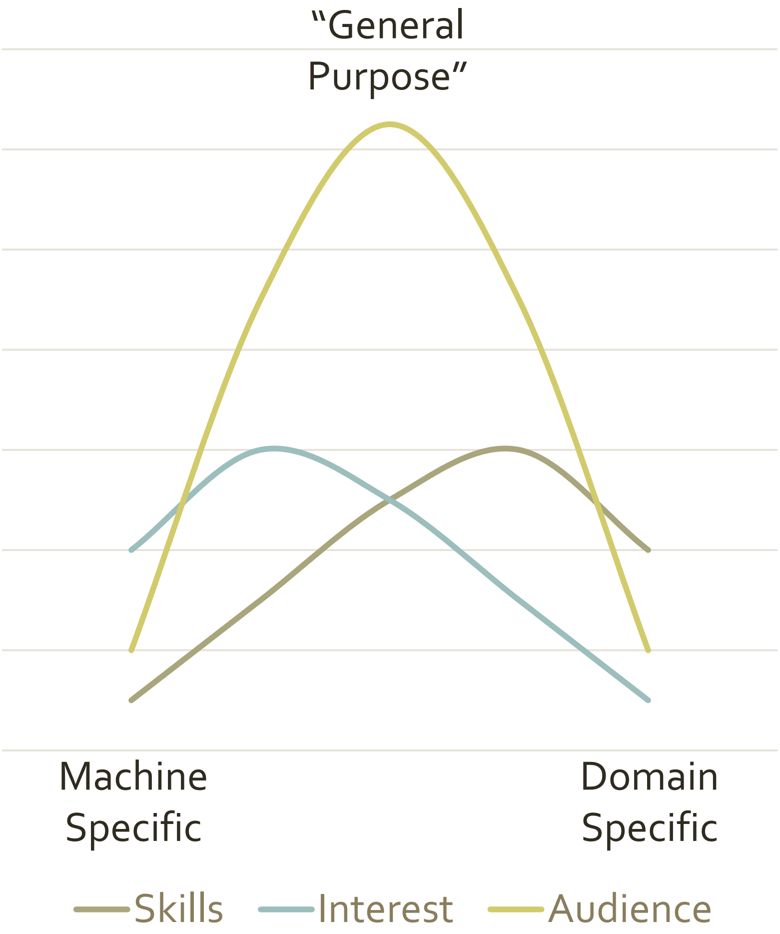
Abraham Maslow, Psychologist, 1966

Programming Languages

Given a computer with some primitive operations and a problem to solve.

Formulate a composition of instructions to the computer that solve the problem.

- Instructions can be very **low-level** (close to the machine's primitive operations)
- Instructions can be very **high-level** (close to the problem domain at hand)
- Most languages strike a **balance**
  - Too low-level (limited audience, limited target machines)
  - Too high-level (limited audience, limited problem domains)



Programmer

A person skilled in designing and developing programs.

The chauffeur of your computer!

- Why not "drive" your own computer to go where you want to go?
  - This is not about "using" a computer application, in the simple sense.
- Why not write the programs you need to get your job done, yourself?
  - This is not about "programming" a computer either, in the fullest sense.
- Why not master a programming language?
  - If the language is Abstract Algebra, you'll be in trouble. If it is Pidgin, you are in trouble too.

Self-Service Programming

Think of cars that most people can learn to drive.

Clearly not to the limit of what "cars" can be; think 18-wheeler trucks or F1.

- Query by Example

  Moshé M. Zloof, IBM Research, mid-1970s

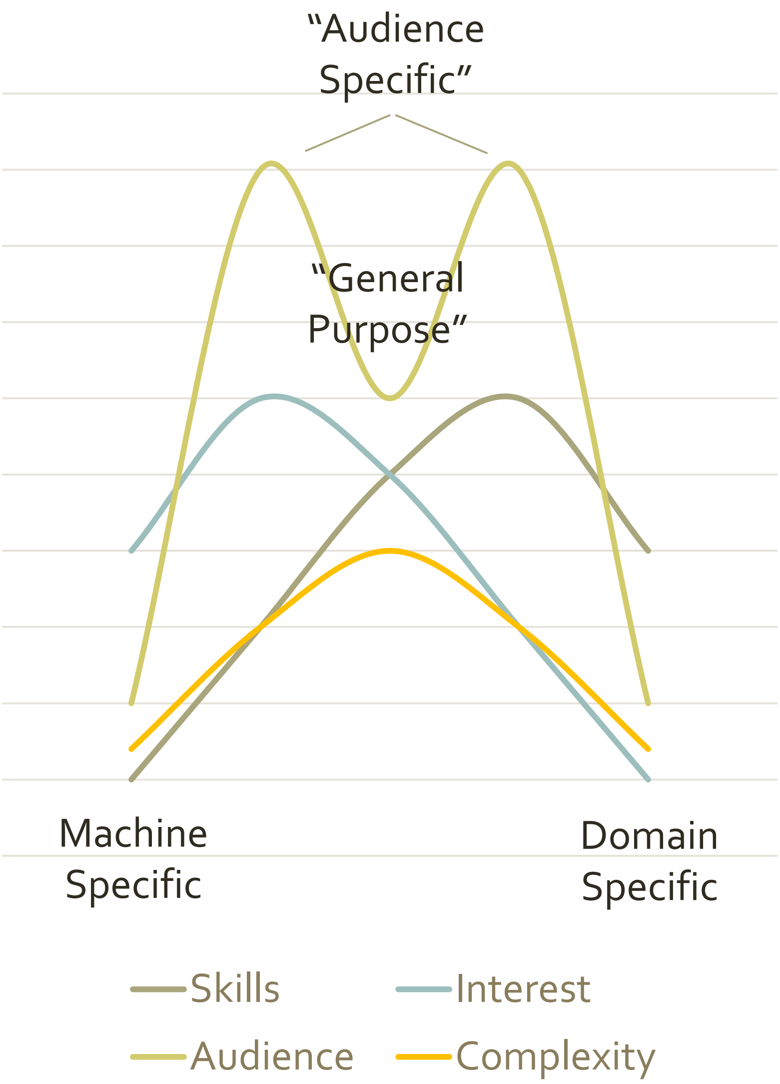
- Generalizes to Programming by Example
  - Using direct manipulation, change results of a program, causing the system to adjust that program.
- Users can watch the effect on the underlying program – and learn from that.
  - Some users pick up ways to change their programs directly, naturally learning the underlying programming language.
  - Requires uniform and simple languages.

Audience-Specific Programming Languages

Consider a variety of *personas* that characterize how groups of people get their tasks done.

Consider a set of personas that fall into comparable needs/skills categories. Call that an *audience*.

- Languages that strive to be "general purpose" end up being not quite right at most anything.
- To compensate, such languages develop a large arsenal of specialized but overlapping capabilities.
- The ideal maximized audience is subdued by complexity.
- Larger audiences can be served with simpler languages to either side of the "general purpose" point.



Anyone can drive a car

Downside: everyone does drive a car.

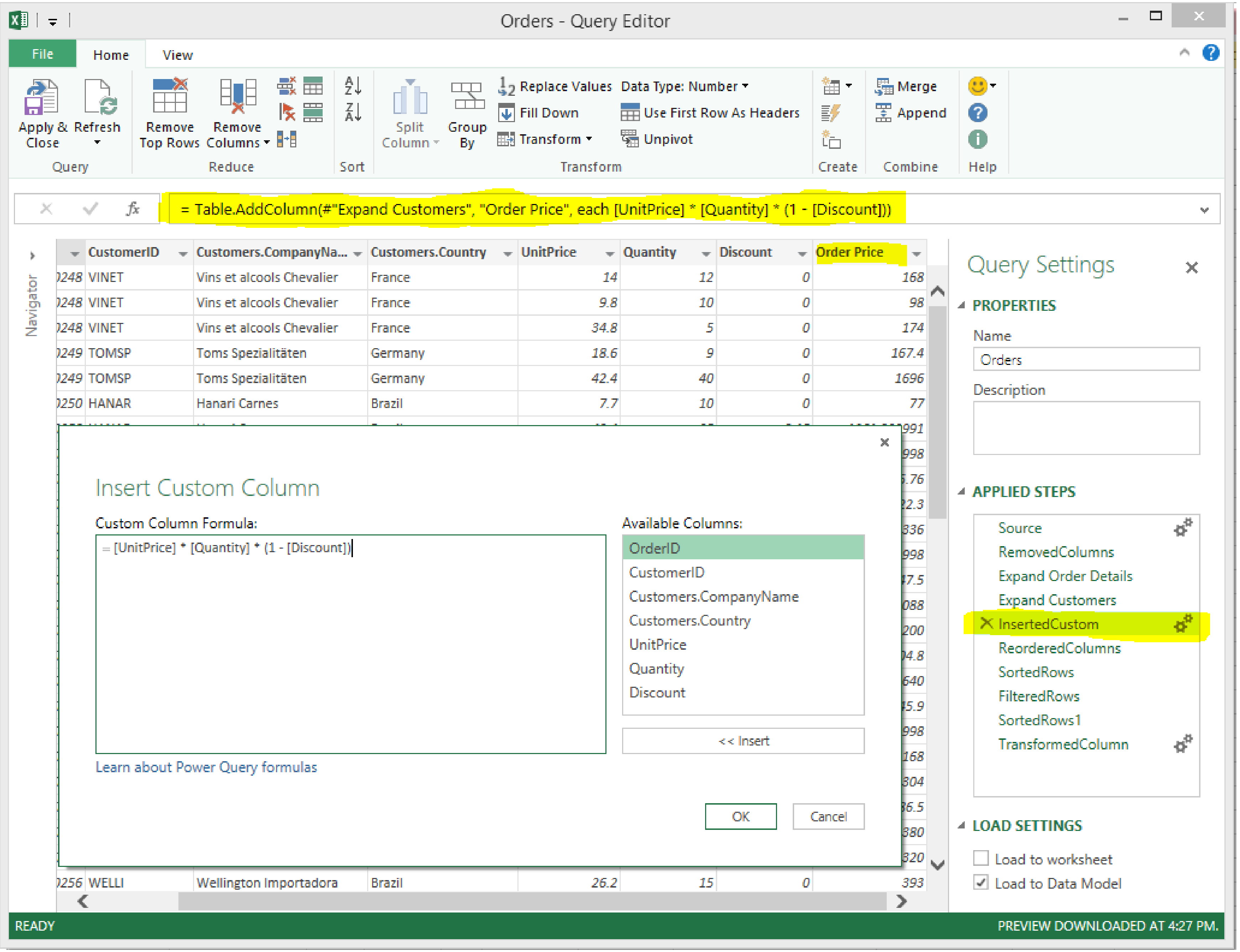
"The trouble with programmers is that you can never tell what a programmer is doing until it's too late."

Seymour Cray

Anyone can write a program

For a suitable set of domains and requirements.

Example: *Power Query*, a part of Microsoft Power BI, aims at Excel users that gather, combine, and analyze data from a wide variety of sources.



“M” - a simple programming language

Again, an example – the Power Query Expression Language (often referred to as “M” for short).

- Target audience is advanced Information Workers (Analysts etc.), Data Stewarts
  - Specifically, top 10% (ish) of Excel users
  - Litmus test: benefits from today’s Excel formulas
- For that audience, the language should be
  - Simple, easy to remember
  - Easy to read and write; limited syntax, little use of non-standard symbols
  - Powerful; no cliffs for advanced user
  - Wide range of “data models” (relational, hierarchical, semi-structured, etc.)

Uniform simple syntax

The *syntax* of a language defines the form a valid expression in that language takes.

It does not, as such, define the meaning of such an expression.

T-SQL

```
SELECT  Orders.OrderDate, Products.OrderID, Products.ProductSKU
FROM  Products
INNER JOIN  Orders  ON  Products.OrderID = Orders.OrderID
ORDER BY  ProductSKU ;
```

C# LINQ syntax

```
from p in Products
join o in Orders on p.OrderID equals o.OrderID
orderby p.ProductSKU
select new { o.OrderDate, p.OrderID, p.ProductSKU }
```

C# LINQ pattern

```
Products
    .Join(Orders,
        p => p.OrderID, o => o.OrderID,
        (p, o) => new { o.OrderDate, p.OrderID, p.ProductSKU } )
    .OrderBy( p => p.ProductSKU )
```

“M”

```
let Joined  = Table.Join( Products, "OrderID", Orders, "OrderID" ),
    Columns = Table.SelectColumns( Joined,
                    {"OrderDate", "OrderID", "ProductSKU"} ),
    Sorted  = Table.Sort( Columns, "ProductSKU" ),
in  Sorted
```

Semantics to meet expectations & requirements

The *semantics* of a language defines the meaning of an expression.

Semantics is defined relative to the syntax of a language.

For a language to be “simple”, its semantics should follow a few uniform principles.

- Dynamic
  - “M” programs only fail when reaching an invalid evaluation state
  - Static checking, beyond syntax, is an option for tools
- Functional (mostly)
  - Mostly deterministic: no direct side effects; mostly referentially transparent; once calculated, all values are immutable
  - External data is stream-processed (not necessarily buffered) and can be non-repeatable; error handling can expose non-determinism
- Higher-order
  - Functions, closures, and types are also values
  - Nested application and conditionals as only forms of “control flow”
- Optionally typed
  - Mostly optional yet expressive type system; very limited runtime checking of types

No control-flow primitives … Say again?

Control flow in a programming language directs the flow of program execution based on state observations.

Examples include constructs for looping (iteration), branching (case selection), and even jumping (“goto”).

- “M” discourages explicit control flow (even recursion!) and prefers higher-order application
  - Many library functions take functions as arguments

```
Table.SelectRows( table, (row) => row[Manager] = row[Buddy] )
```

Table.SelectRows is the name of a function. If applied to a table and a predicate, it returns a new table with rows that meet that predicate.

This function is *higher-order*; it takes a function as its argument.

The second argument is a function that takes a single row and determines whether that row should be selected (or dropped).

In the example, the predicate function is *anonymous*; it has no name and is defined right where it is needed.

Making the most common case simple

A common pattern is that higher-order functions take unary functions (single-parameter functions) as arguments.

Think items in a list, rows in a table, fields in a record.

- "M" discourages explicit control flow (even recursion!) and prefers higher-order application
  - Many library functions take functions as arguments

```
Table.SelectRows( table, (row) => row[Manager] = row[Buddy] )
```

- Often, those parameter functions are unary
  - A special syntactic form helps construct unary function values

```
Table.SelectRows( table, each _[Manager] = _[Buddy] )
```

- An 'each' expression is just shorthand for a unary function
  - The single parameter of an 'each' function is named _
- For conciseness, the _ can be omitted when accessing fields or columns (this is the *only* case of syntactic finesse in M)

```
Table.SelectRows( table, each [Manager] = [Buddy] )
```

Evaluation Model

The evaluation model of a language determines *how* expressions are evaluated.

This can be seen as a refinement of the language's semantics.

- Expressions evaluate to values in a context
  - The context binds names to values
- Function application is strict
  - No Excel-style if(*condition*, *true-expression*, *false-expression*)
    - "M" has an if-expression (the only admission to control flow)
- Evaluation is eager except for value construction
  - Construction of structured values (records, lists, tables) is lazy
  - Can deal with infinite lists and tables
  - Can deal with partial records and lists
    (values containing embedded errors only show when accessed)
- Evaluation 'fails fast' on hard errors
  - Simple model to raise and handle soft errors within M

Streaming

Evaluating data in a *streaming* fashion allows data to be processed as it arrives (instead of waiting for it to arrive completely).

Not all operations can be streamed. For example, sorting is a non-streaming operation.

- Resource adapters can expose data as streams
  - The world at large is not transactional
- Streams appear as lists or tables in M
  - Unlike regular values, streams are not necessarily repeatable
    - `List.Count(stream)` may not coincide with the number of items seen when exhausting the stream a second time, after counting it
  - `List.Buffer(stream)` and `Table.Buffer(stream)` functions take a stable snapshot of a stream
    - "Memoizes" a copy of all items in the stream into memory, as the underlying stream is enumerated

Overall “M” evaluation

The main purpose of the “M” system: Building a bridge from the natural expressions a user of “M” writes and the execution models that the diverse world of data stores and sources supports.

- Users build up expressions step-by-step, in their natural order
  - They draw on external resources when convenient
  - They apply functions in any order that seems appropriate
  - Copying external data entirely to local system is often unacceptable
- External resources support varying querying capabilities
  - Importer for text files (incl. CSV and log files) does simple things to avoid unnecessary string explosions
  - XML and HTML importers can handle certain path queries
  - Excel importer can handle simple framing queries
  - OData feeds support more or less complete OData queries
  - Access, SQL Server, Oracle, Teradata, etc. support SQL queries
    - Just not the same SQL!
  - LDAP queries over Active Directory, graph queries over Facebook, item queries over Exchange, …

Query Folding

Example

```
let Joined  = Table.Join( Products, "OrderID", Orders, "OrderID" ),
    Columns = Table.SelectColumns( Joined,
                      {"OrderDate", "OrderID", "ProductSKU"} ),
    Sorted  = Table.Sort( Columns, "ProductSKU" ),
in  Sorted
```

- User applies functions step-by-step
- System translates to external and efficient queries

```
SELECT  Orders.OrderDate, Products.OrderID, Products.ProductSKU
FROM  Products
INNER JOIN  Orders  ON  Products.OrderID = Orders.OrderID
ORDER BY  ProductSKU ;
```

Query Folding

By deferring the construction of result values, an "M" system can gather up operations until results are demanded.

Gathered-up operations can be translated ("folded") into external query expressions.

- Expressions are built in user-preferred order
- The "M" system performs runtime analyses to determine how to best break up ("fold") an expression into subqueries that can be federated to multiple resources
  - Takes into account multiple dimensions, including estimates of set sizes, statistics, connection latencies, query capabilities of heterogeneous resources
- To inject runtime analysis, lazy value-construction is used to aggregate expressions and defer evaluation of results until demand arrives
  - For individual lists and tables, this is similar to how LINQ works
  - Also done through arbitrary M-defined functions (unlike LINQ)
  - Streaming auto-adaptive join across multiple external sources

Power Query Data Sources

This list is continuously growing.

- Web page
- Excel or CSV/PSV/… file
- XML file, JSON file
- Text file
- Folder
- SQL Server database
- Windows Azure SQL database
- Access database
- Oracle database
- IBM DB2 database
- Sybase database
- Teradata database
- MySQL database
- PostgreSQL database
- SharePoint list
- OData feed
- Azure blob and table store
- Hadoop Distributed File System (HDFS)
- Windows Azure HDInsight (Azure Blob Store mapping of HDFS)
- Windows Azure Marketplace feeds and services
- Active Directory
- Facebook graphs
- Exchange
- SAP BOBJ soon

Key Takeaways

- Information Workers approach languages differently
  - Aligning with Excel's formula language is important
  - Aligning with C idioms (a.k.a. C++, C#, Java, JavaScript idioms ☺) is not a priority
  - Avoiding symbolic or syntactic overload commonly found in programming languages is important
- Information Workers need to solve their problems anyway
  - Embracing diversity in scale, schematization, even ill-formedness
  - Embracing "soft semantics" in transactional closure, repeatability, and edge-case handling
- Creating a powerful yet simple language for the user requires addressing some hard technical problems (ongoing ...)
  - Dynamic lazily-constructing language – how to deal with errors and diagnostics?
  - Runtime execution planning and federation – how to deal with "cliff" surprises?

Still need a driver, anyone?

Elevators and washing machines have an interesting thing in common: they no longer require a human operator.

And, yes, Google invented the self-driving car. Not.

Resources

- Power Query has shipped in two versions
  - Standalone (v1) shipped in July 2013
  - Corporate (v2) shipped in February 2014
    - Integrated part of Power BI offering, a subscription service aligned with Office 365
    - http://powerbi.com/
- Tutorials, samples, M language, and M library references
  - http://office.microsoft.com/en-us/excel-help/microsoft-power-query-for-excel-help-HA104003813.aspx