Rules as a Control Structure

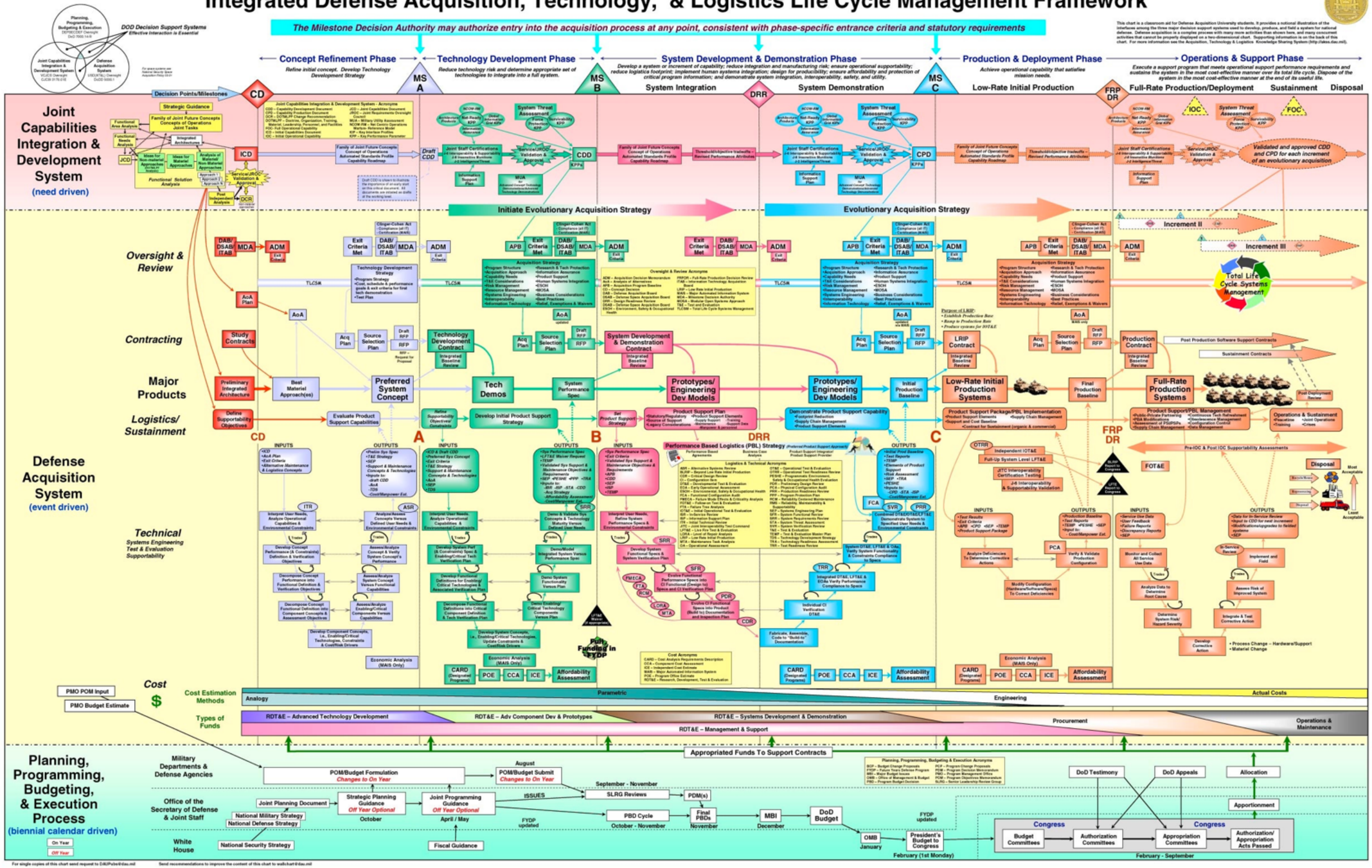
Ryan Brush
@ryanbrush

Midwest.io 2014

The story so far…

ver. 5.2. August 2005

Integrated Defense Acquisition, Technology, & Logistics Life Cycle Management Framework

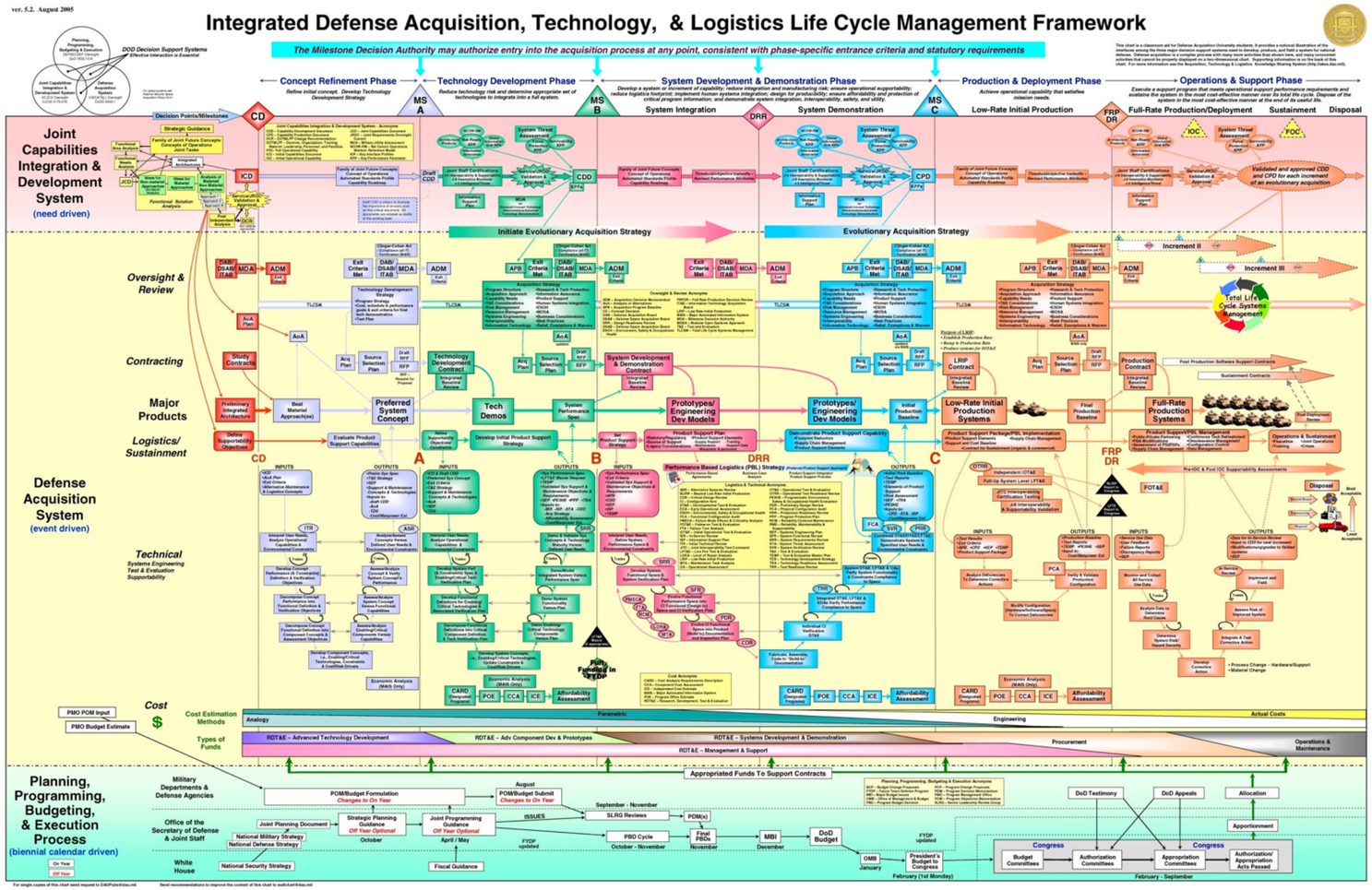


This chart is a classroom aid for Defense Acquisition University students. It provides a notional illustration of the interfaces among the three major decision support systems used to develop, produce, and field a system for national defense. Defense acquisition is a complex process with many more activities than shown here, and many concurrent activities that cannot be properly displayed on a two-dimensional chart. Supporting information is on the back of this chart. For more information see the Acquisition, Technology & Logistics Knowledge Sharing System (http://akss.dau.mil).

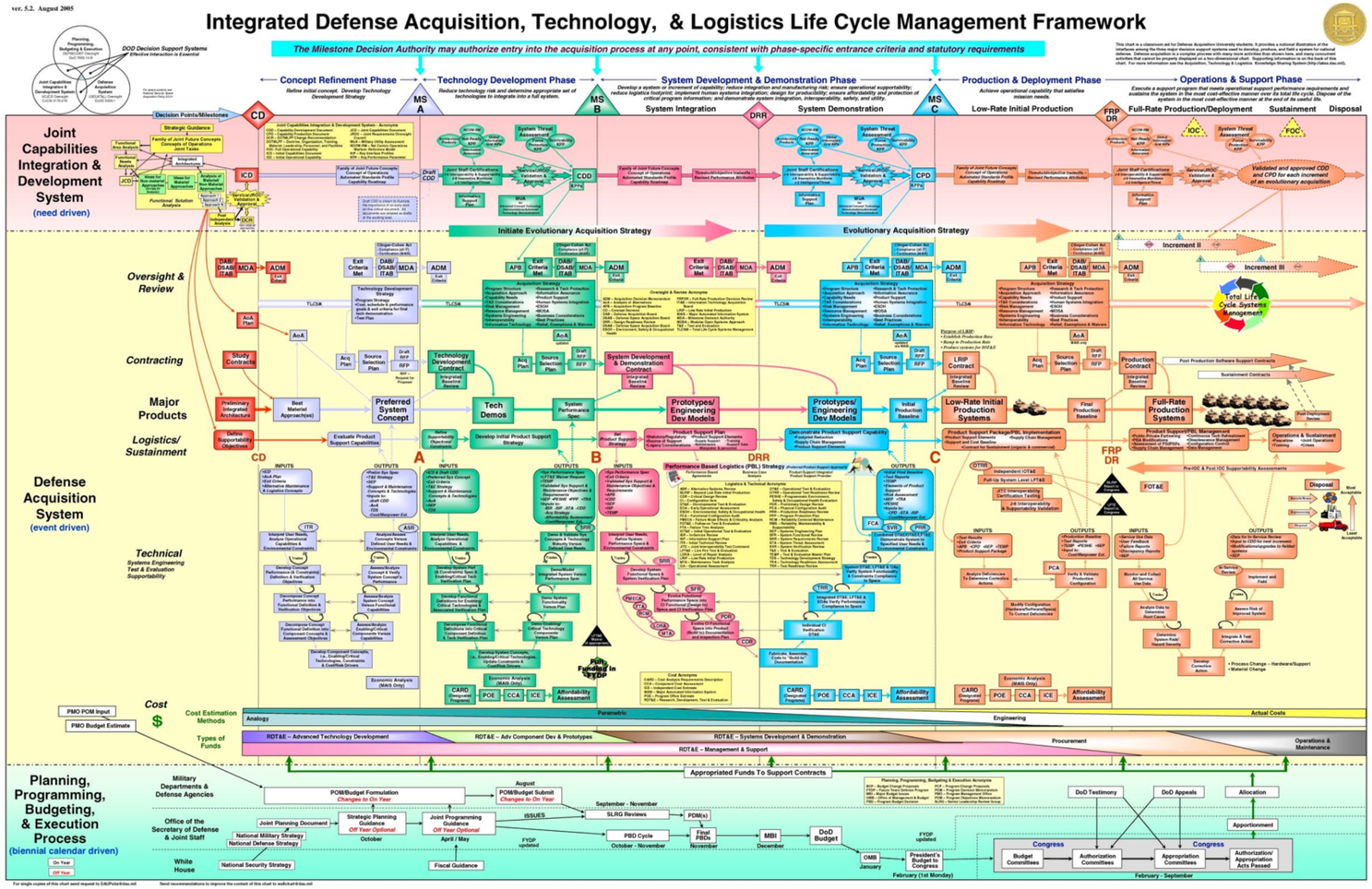
For single copies of this chart send request to DAUPubs@dau.mil

Send recommendations to improve the content of this chart to wallchart@dau.mil

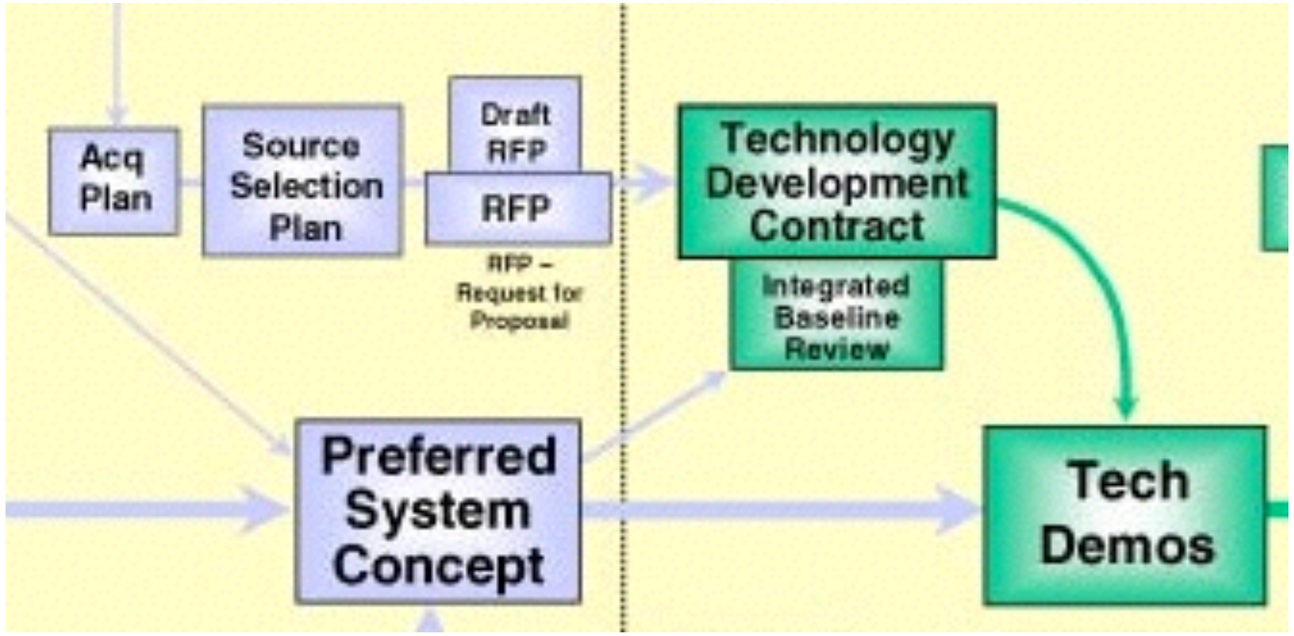
How do we approach this?



Users don’t think of everything at once



Break it down into pieces



Functional Decomposition?

Object models?

A B

f(A,B) => C

f(B,C) => D

f(A,C,D) => E

A B

f(A,B) => C

f(B,C) => D

f(A,C,D) => E

A = Travel Expenses
B = Contract Budget
C = Over Budget Expenses
D = Required justifications
E = Expense report summary

A B

f(A,B) => C

f(B,C) => D

f(A,C,D) => E

A = Customer Demographics
B = Order History
C = VIP status
D = Current Offerings
E = Sales Package

A B

f(A,B) => C

f(B,C) => D

f(A,C,D) => E

A = Diagnosis history
B = Lab test results
C = Complication risks
D = Interventions
E = Plan of Care

A B

f(A,B) => C

f(B,C) => D

f(A,C,D) => E

…or many other domains

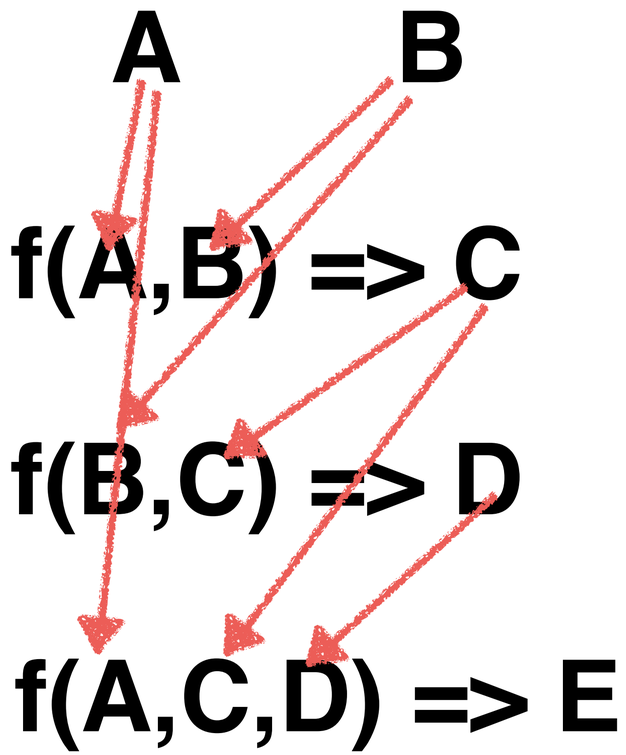
A B

f(A,B) => C

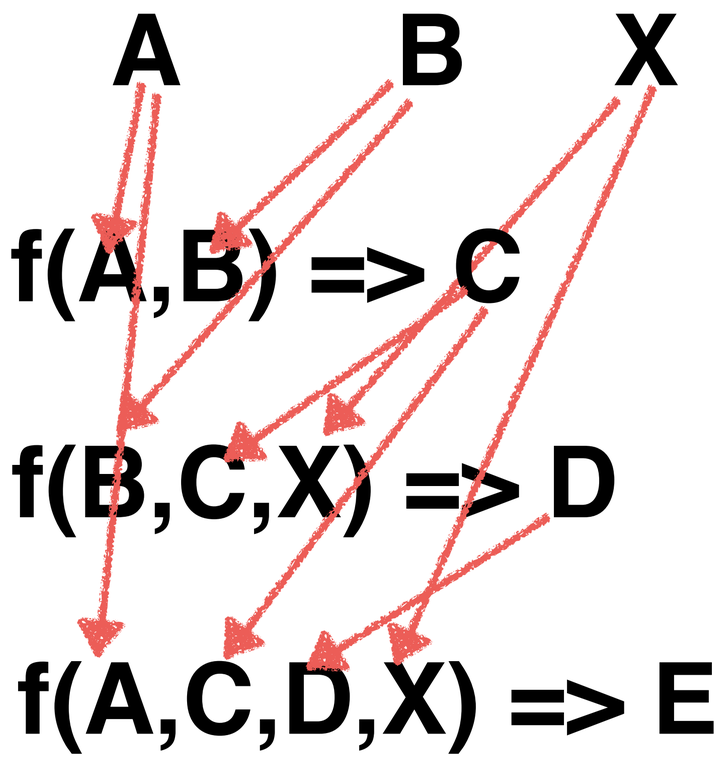
f(B,C) => D

f(A,C,D) => E

But we must wire things together!



But we must wire things together!



Sweeping changes from new inputs

Now try this with thousands of requirements from different sources

(Hint: it doesn’t end well.)

Explicit wiring is complex and error prone

So let’s wire them automatically

A **B**

f(A,B) => C

f(B,C) => D

f(A,C,D) => E

How do we automatically wire this?

A B

f(A,B) => C

f(B,C) => D

f(A,C,D) => E

unit-of-logic

Type[constraints]

Type[constraints]

Type[constraints]

=>

action

Looks like a rule engine

It’s just if-this-then-that!

(Until it’s not)

```
rule “My Rule”
    when:
      // Pile of Java code
    then:
      // Another pile of Java code
```

Make it a control structure (for real this time)

Expressiveness of a programming language

Nools (JavaScript)

Wongi (Ruby)

Clara (Clojure)

Nools (JavaScript)

Wongi (Ruby)

Clara (Clojure)

Why Clojure?

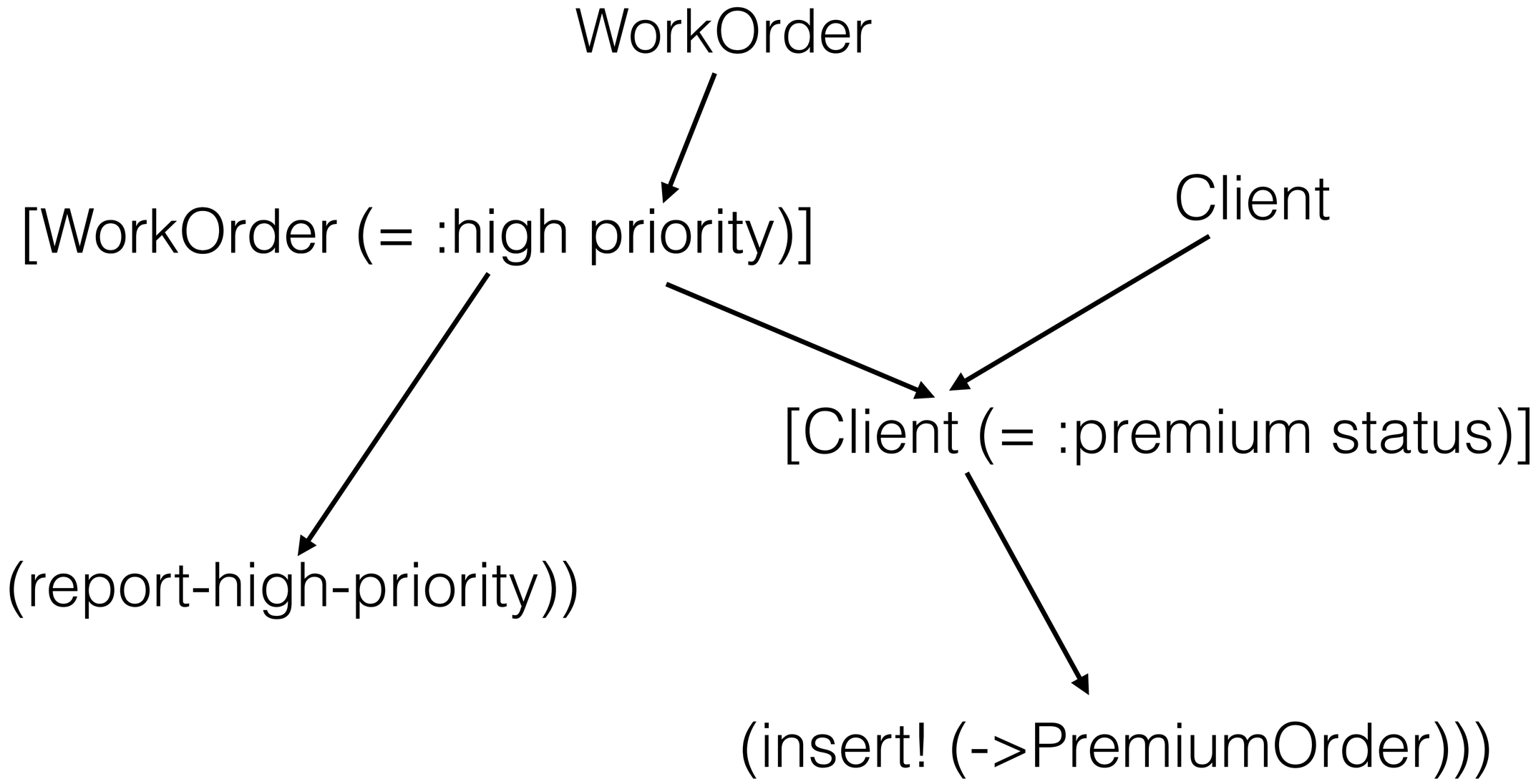
Opinionated core principles

Can grow the language

(Hello, macros!)

```
(defrule high-priority-report
  [WorkOrder (= :high priority)]
 =>
  (report-high-priority))
```

```
(defrule prioritize-order
  [WorkOrder (= :high priority)]
  [Client (= :premium status)]
 =>
  (insert! (->PrioritizeOrder)))
```

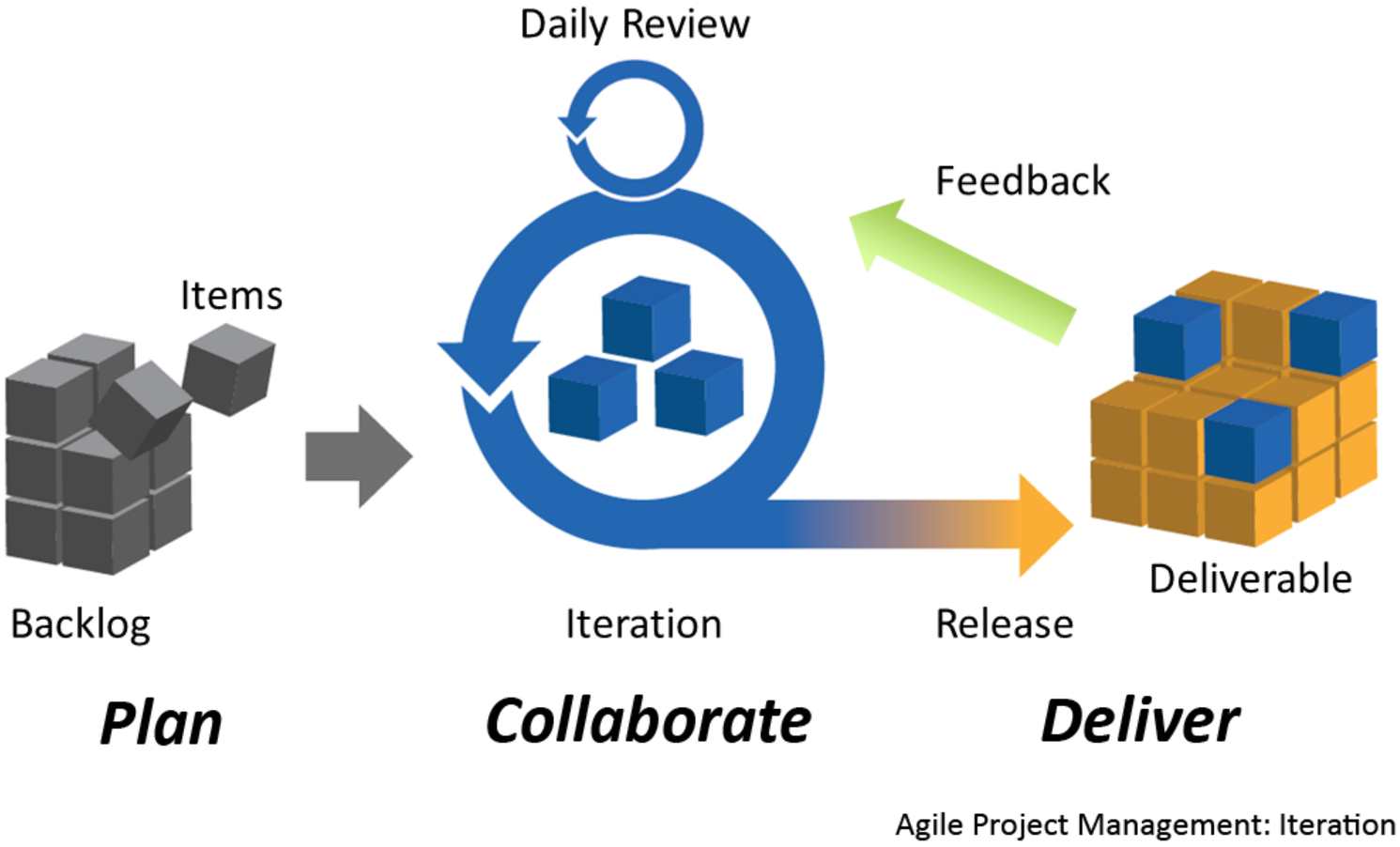

WorkOrder
[WorkOrder (= :high priority)]
Client
[Client (= :premium status)]
(report-high-priority))
(insert! (->PremiumOrder)))

Demo

Collaboration is our biggest challenge

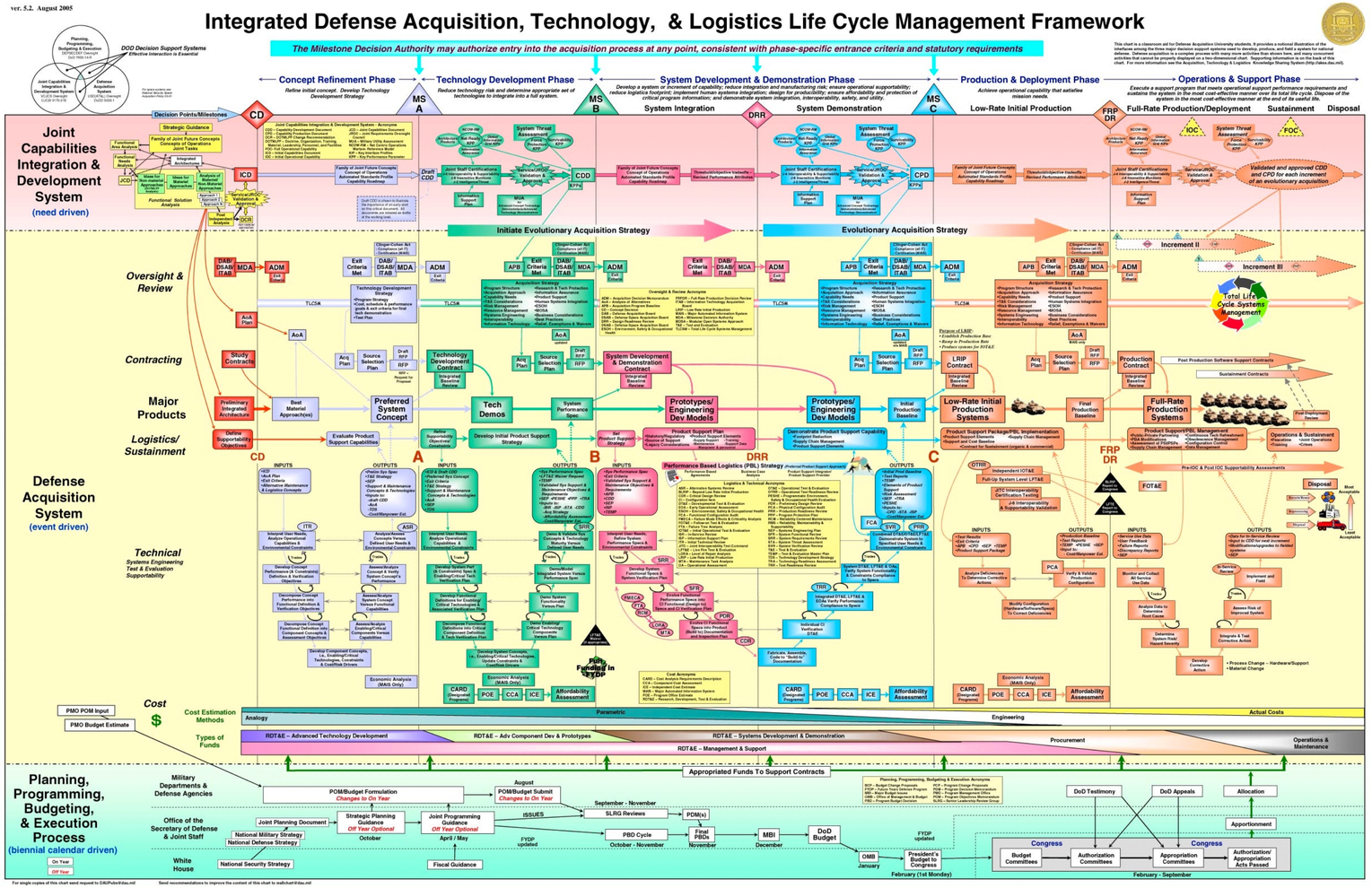

What users want
MIND THE GAP
What the system does

Agile Development!



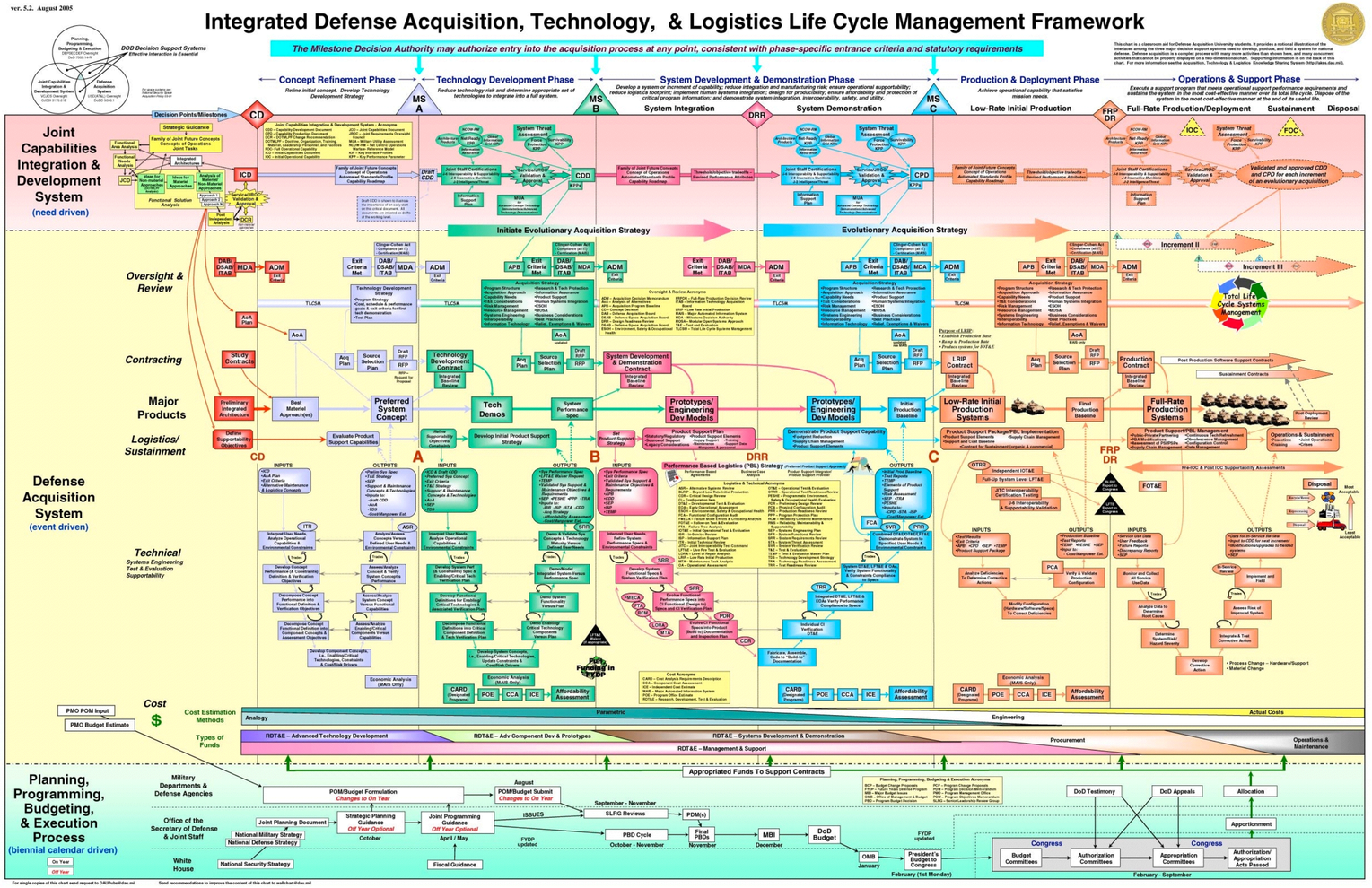
http://en.wikipedia.org/wiki/Iterative_and_incremental_development

Make this agile



I dare you.

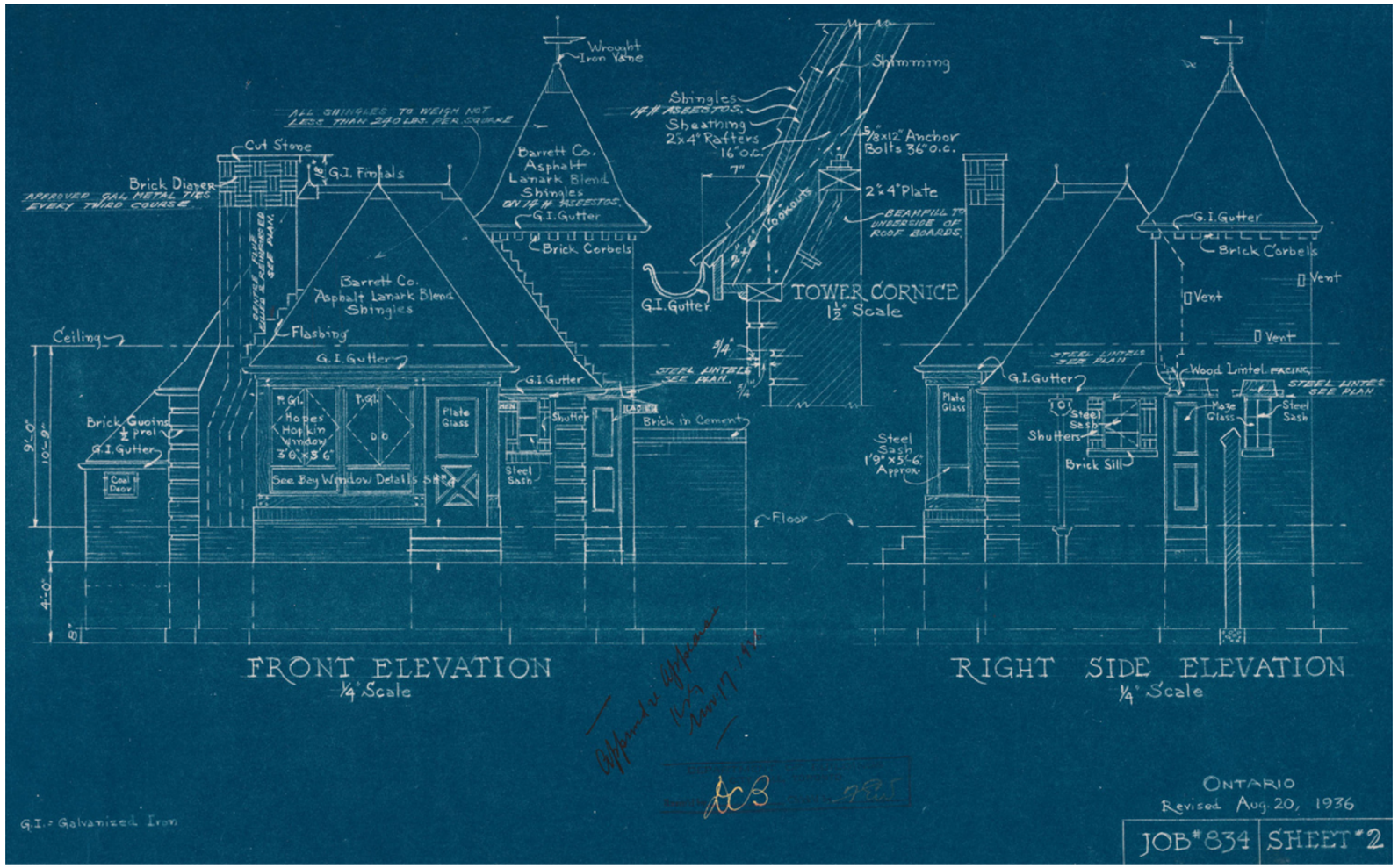
Deep complexity hinders feedback



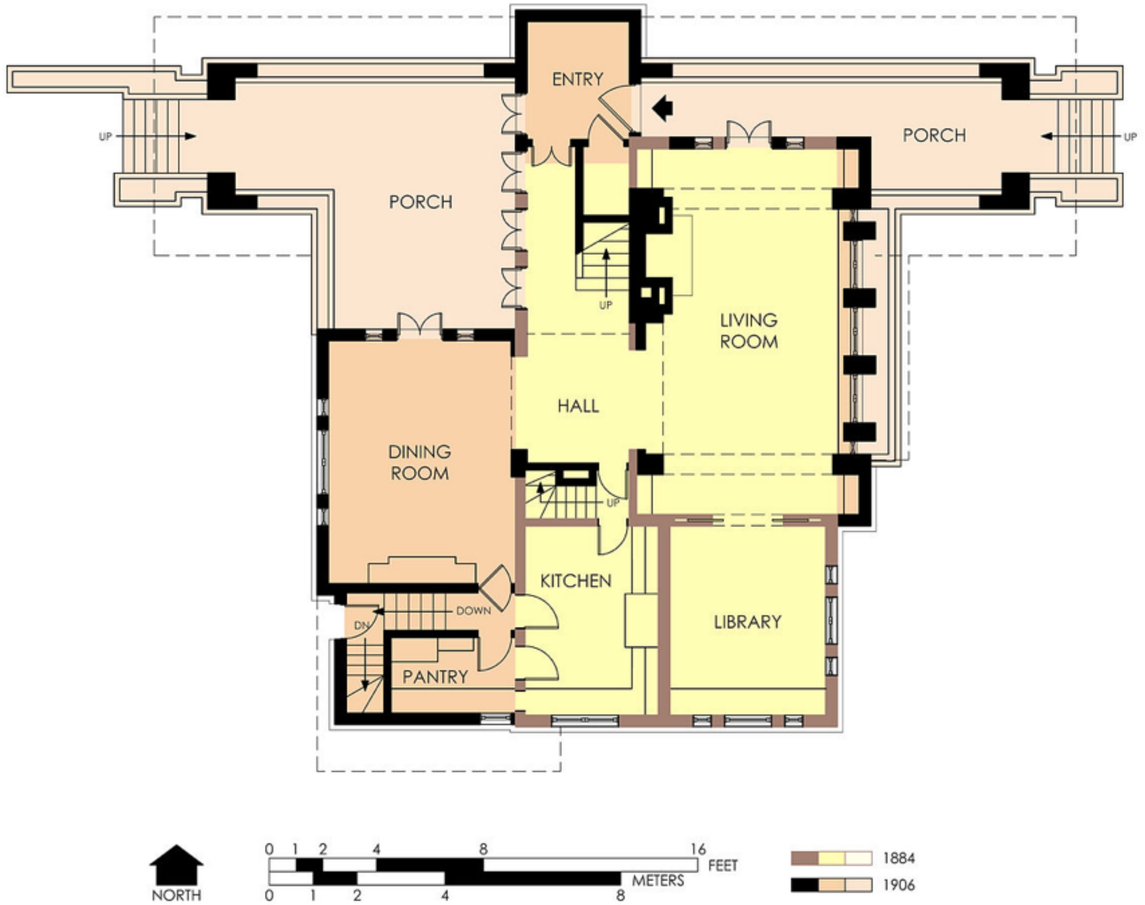
Unit tests exercise the developer’s understanding

Documents are ambiguous

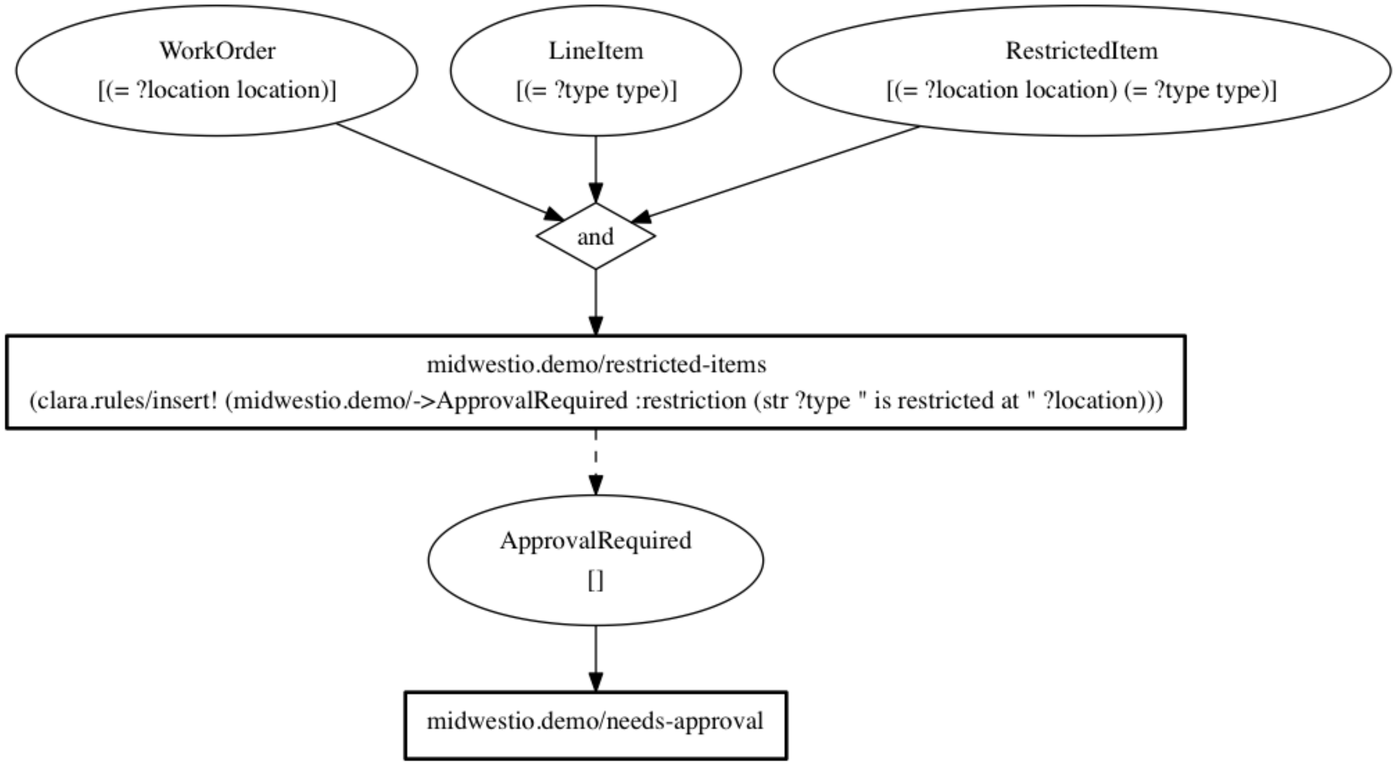
Code is our blueprint



We need a floor plan



We need a floor plan



Direct expression of requirements

Rules as data

Projections of logic

Leverage

Explainability

Query logic

Floor plans

Questions?

@ryanbrush

http://github.com/rbrush/clara-rules