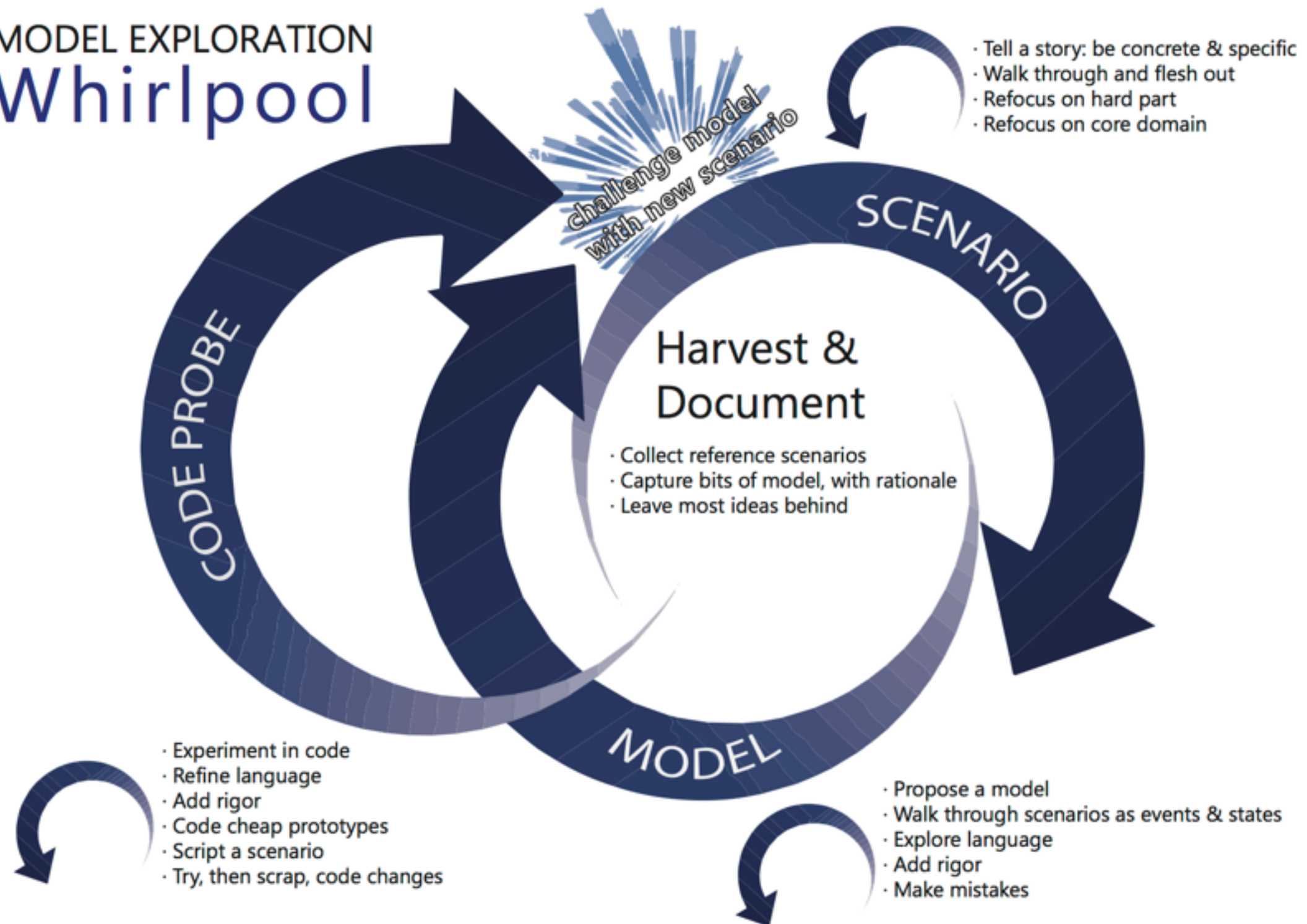


# MODEL EXPLORATION Whirlpool




[domainlanguage.com/ddd/whirlpool](http://domainlanguage.com/ddd/whirlpool)

[http://bit.ly/ddd\\_whirlpool](http://bit.ly/ddd_whirlpool)

# http://bit.ly/ddd\_code

Do this



 paulrayner / ddd\_code\_samples


Unwatch 1 Star 0 Fork 0


[Code](#) [Issues 0](#) [Pull requests 0](#) [Wiki](#) [Pulse](#) [Graphs](#) [Settings](#)

No description or website provided. — Edit

1 commit 2 branches 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

 paulrayner Add sample files. Latest commit 0a93782 on Jan 25

 src/warranty Add sample files. 5 months ago

Help people interested in this repository understand your project by adding a README. [Add a README](#)

# Limit of Liability

- Look at new `limitOfLiability()` method in `Contract`
- How has adding a `limitOfLiability()` method improved the maintainability and readability of the code?
- What is the connection between adding this method and growing our ubiquitous language? How has this improved our domain model?
- How did we separate the concept of Limit of Liability from the `Contract` lifecycle?

# Terms and Conditions

- What behavior does TermsAndConditions have?
- What things did we need to do to make TermsAndConditions a value object? How does the unit test for TermsAndConditions show this?
- How has making TermsAndConditions a value object improve testability? encapsulation? thread safety? code maintenance?
- What does the annuallyExtended() method do, and how is it used by the Contract entity?
- How does adding this new value object focus the Contract entity more tightly on identity and lifecycle?

[http://bit.ly/ddd\\_vo](http://bit.ly/ddd_vo)

Read “*Why Value Objects*”, &  
“*Value Objects and Immutability*”

## Leading by Design

Thinking about DDD, BDD, coding, software design and agile etc.

Twitter | GitHub | Google+ | LinkedIn | Facebook | About | Archive | RSS

Email Address  [Sign Up!](#) ← Subscribe to the *Leading By Design* newsletter

# Why Value Objects?

Jan 20, 2015 DDD |

Value objects are a key building block pattern in domain-driven design (DDD). Here are some of the reasons why. They...

- Give rich, expressive names to key concepts in our model
- Increase the conceptual cohesiveness of objects, readability of our code and the overall suppleness of our design
- Making value objects immutable reduces incidence of bugs, improves thread

[http://bit.ly/ddd\\_vo](http://bit.ly/ddd_vo)

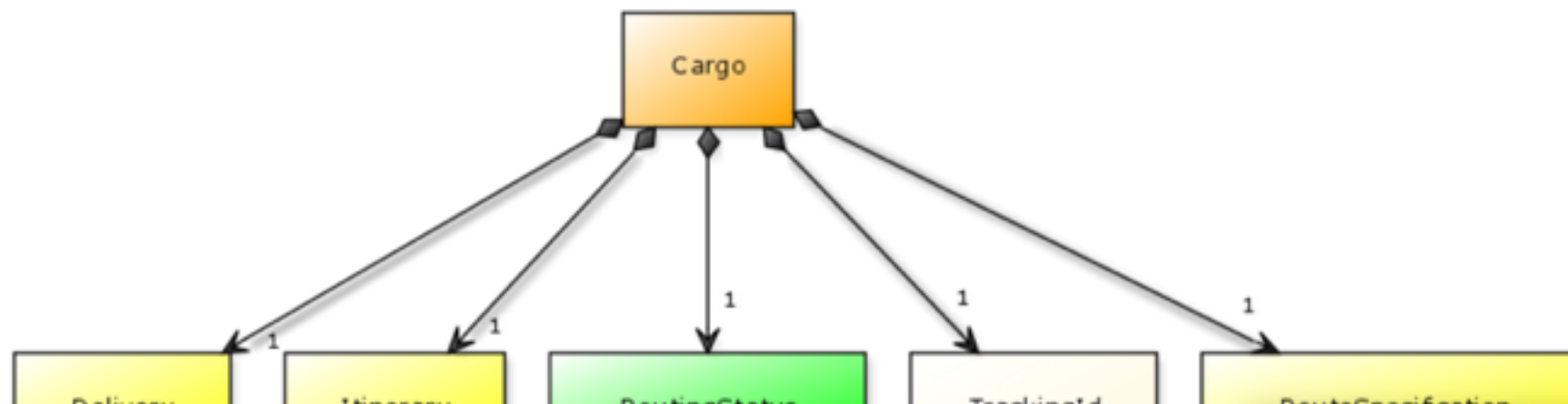
Read this post  
on persistence (3rd in the series)

## Persisting Value Objects

Jan 22, 2015 DDD |

It can be challenging sometimes to know how best to persist value objects to a data store, especially if you are using a RDBMS. There are a variety of options to choose from, however, depending on your needs and constraints.

Examples below are based on my [Ruby port of the DDD sample app](#). Here is a class diagram showing the `Cargo` aggregate, which consists of the `Cargo` entity (as the aggregate root) and a number of value objects, such as `Itinerary` and `RouteSpecification` that are also part of the `Cargo` aggregate. This is based on examples given in Eric Evans's [Domain-Driven Design](#) book.



# Domain-Driven Design Reference

---

Definitions and Pattern Summaries

Eric Evans  
Domain Language, Inc.



© 2015 Eric Evans

This work is licensed under the Creative Commons Attribution 4.0 International License.  
To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

[http://bit.ly/ddd\\_ref](http://bit.ly/ddd_ref)



For your strategic pattern, prepare a 3-min presentation on a flip-chart summarizing its *key details*, and *pros & cons*. Give a concrete example.

WRITE THE NAME OF YOUR PATTERN AT THE TOP OF YOUR  
FLIPCHART SHEET

- |                                   |                       |
|-----------------------------------|-----------------------|
| 1. Conformist                     | 5. Big Ball of Mud    |
| 2. Partnership                    | 6. Open Host Service  |
| 3. Anti-Corruption<br>Layer (ACL) | 7. Published Language |
| 4. Shared Kernel                  | 8. Customer/Supplier  |

**[http://bit.ly/ddd\\_ref](http://bit.ly/ddd_ref)**