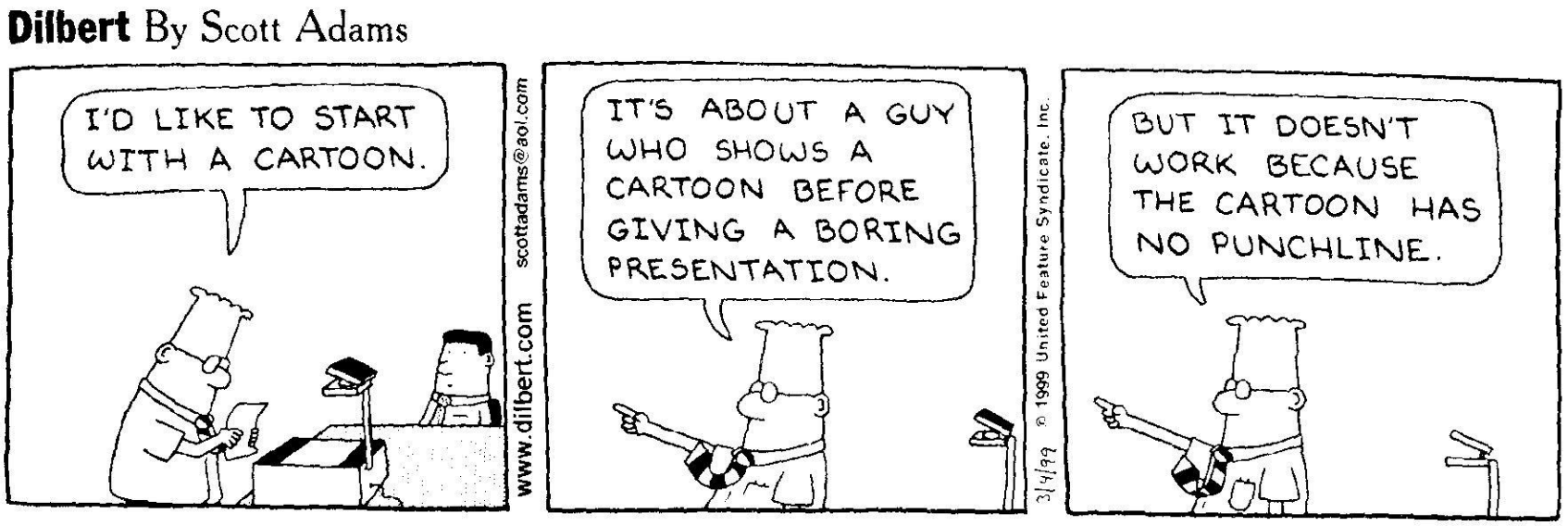
CERN School of Computing

Large Projects & Software Engineering



With thanks to Bob Jones for ideas and illustrations

CERN School of Computing

Why spend so much time talking about "Software Process"?

How do you create software?

- **Lots of parts: Writing, documenting, testing, sharing, fixing, ….**
- **Usually done by lots of people**

"Process" is just a big word for how they do this

- **Exists whether you talk about it or not**

"Why do we have to formalize this?"

CERN School of Computing

Scale and process: Building a dog house

- Can be built by one person
- Minimal plans
- Simple process
- Simple tools
- Little risk

Rational Software Corporation

CERN School of Computing

Scale and process: Building a family house

- Built by a team
- Models
- Simple plans, evolving to blueprints
- Well-defined process
- Architect
- Planning permission
- Time-tabling and Scheduling
- ...
- Power tools
- Considerable risk

Rational Software Corporation

CERN School of Computing

Scale and process: Building a skyscraper

- Built by many companies
- Modeling
- Simple plans, evolving to blueprints
- Scale models
- Engineering plans
- Well-defined process
- Architectural team
- Political planning
- Infrastructure planning
- Time-tabling and scheduling
- Selling space
- Heavy equipment
- Major risks

Rational Software Corporation

CERN School of Computing

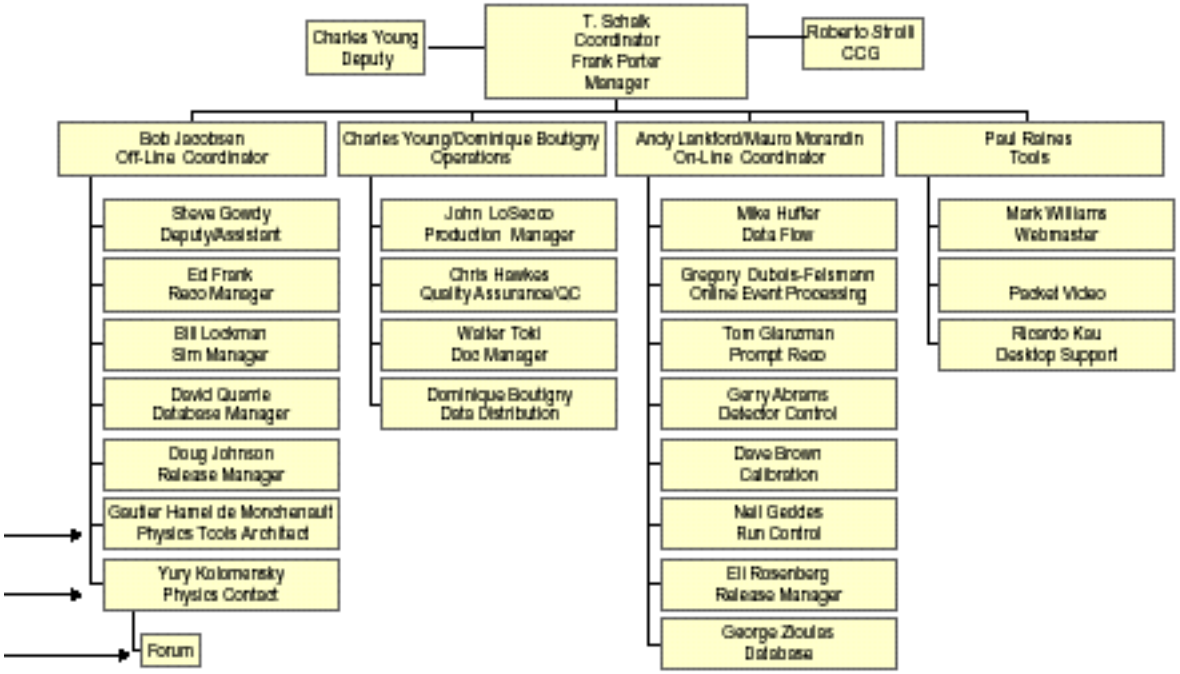
Why do software projects fail?

Even if you do produce the code it does not guarantee that the project will be a success

There are many other factors (both internal and external) that can affect the success of a project...

Communication explosion

More people means *more* time communicating which means more misunderstandings and *less* time for the software



Why software projects fail...

Undefined responsibilities

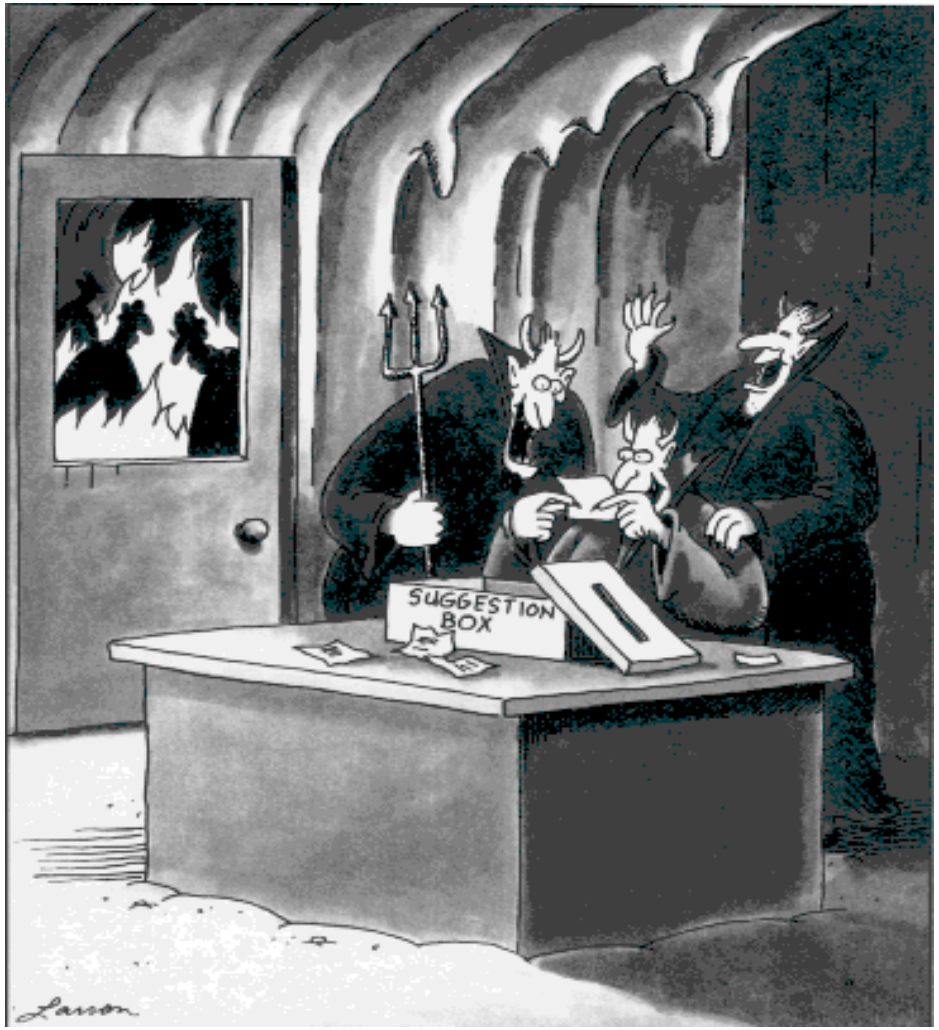
"Hey... this could be the chief"

Too little responsibility can cause a lot of confusion & embarrassing mistakes

Why software projects fail...

Missed user requirements

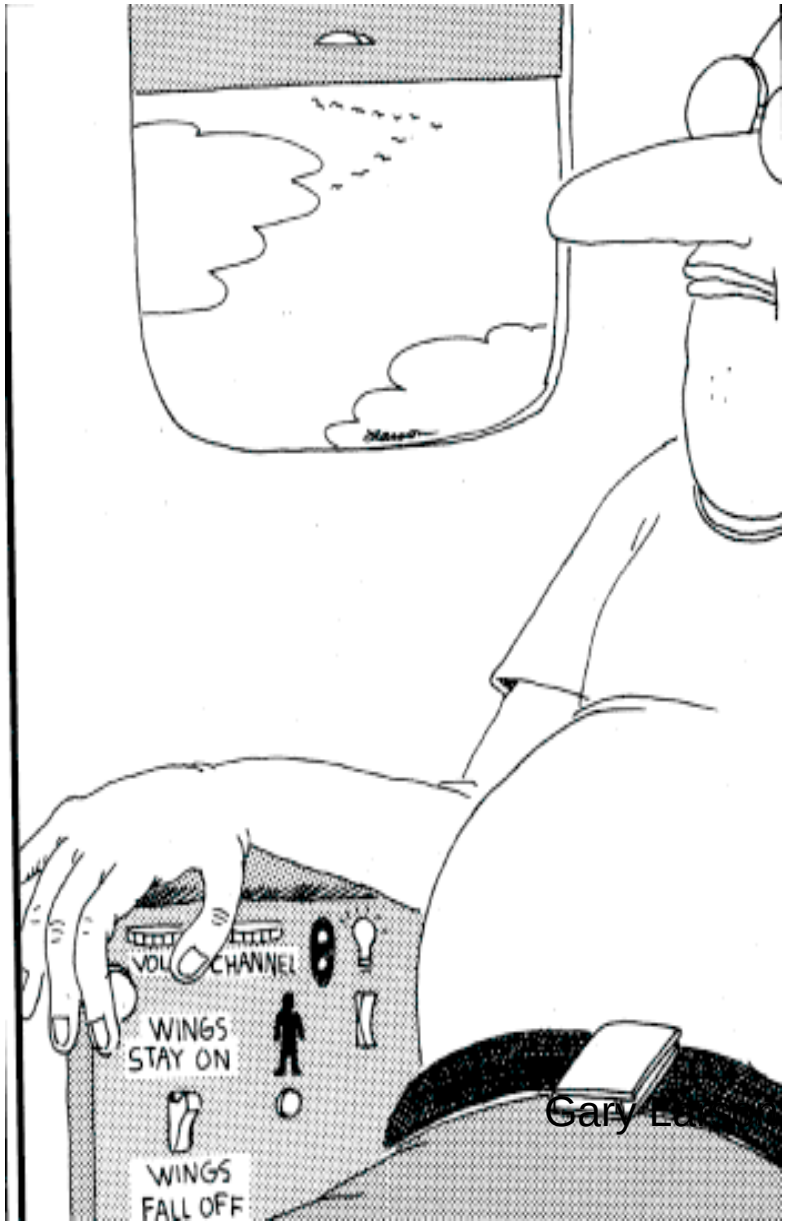
We're not smart enough to know everything people want the system to do; we need to ask!



Why software projects fail...

Badly defined interfaces

Fumbling for his recline button, Bob unwittingly instigates a disaster



Spend the time to design and test good interfaces

Why software projects fail...

Creeping featurism

"No, no… Not this one. Too many bells and whistles"

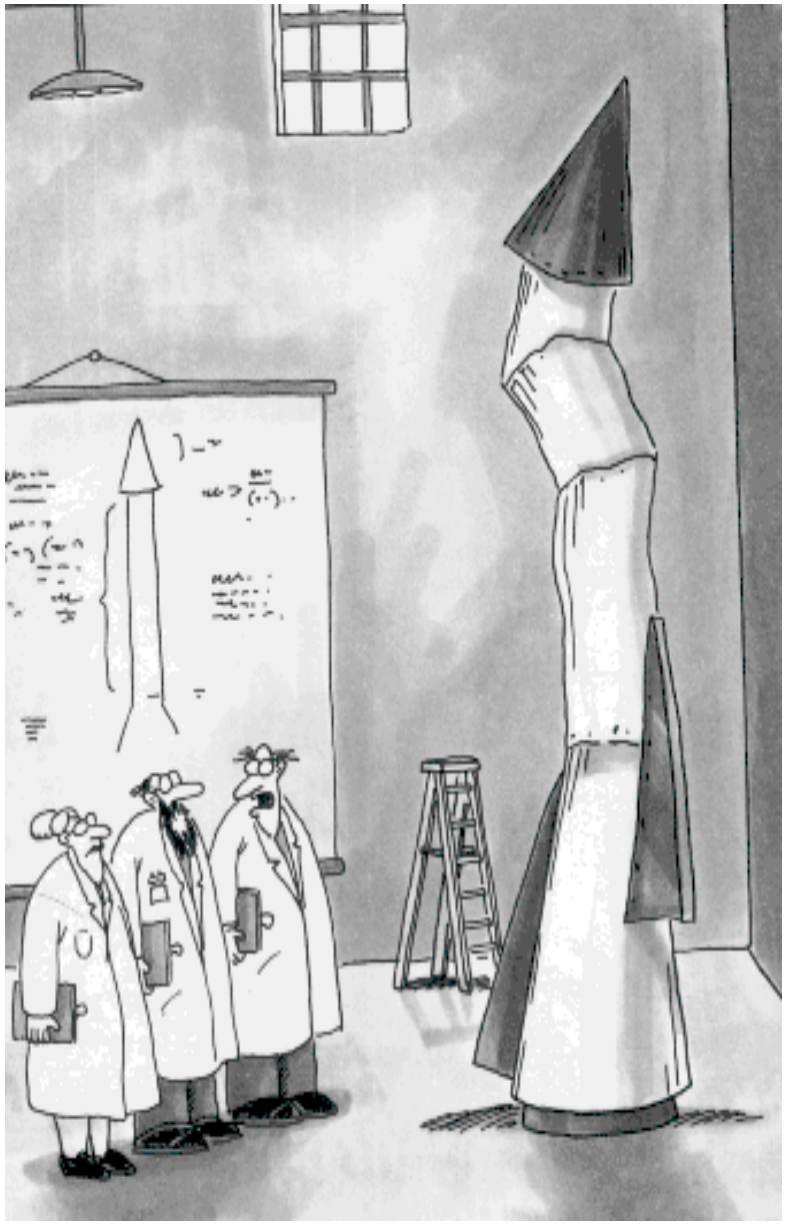
Gary Larson

Focus on what the users are asking for, not what the developers think is cool

Why software projects fail...

Unrealistic goals

"It's time we face reality, my friends…
We're not exactly rocket scientists"



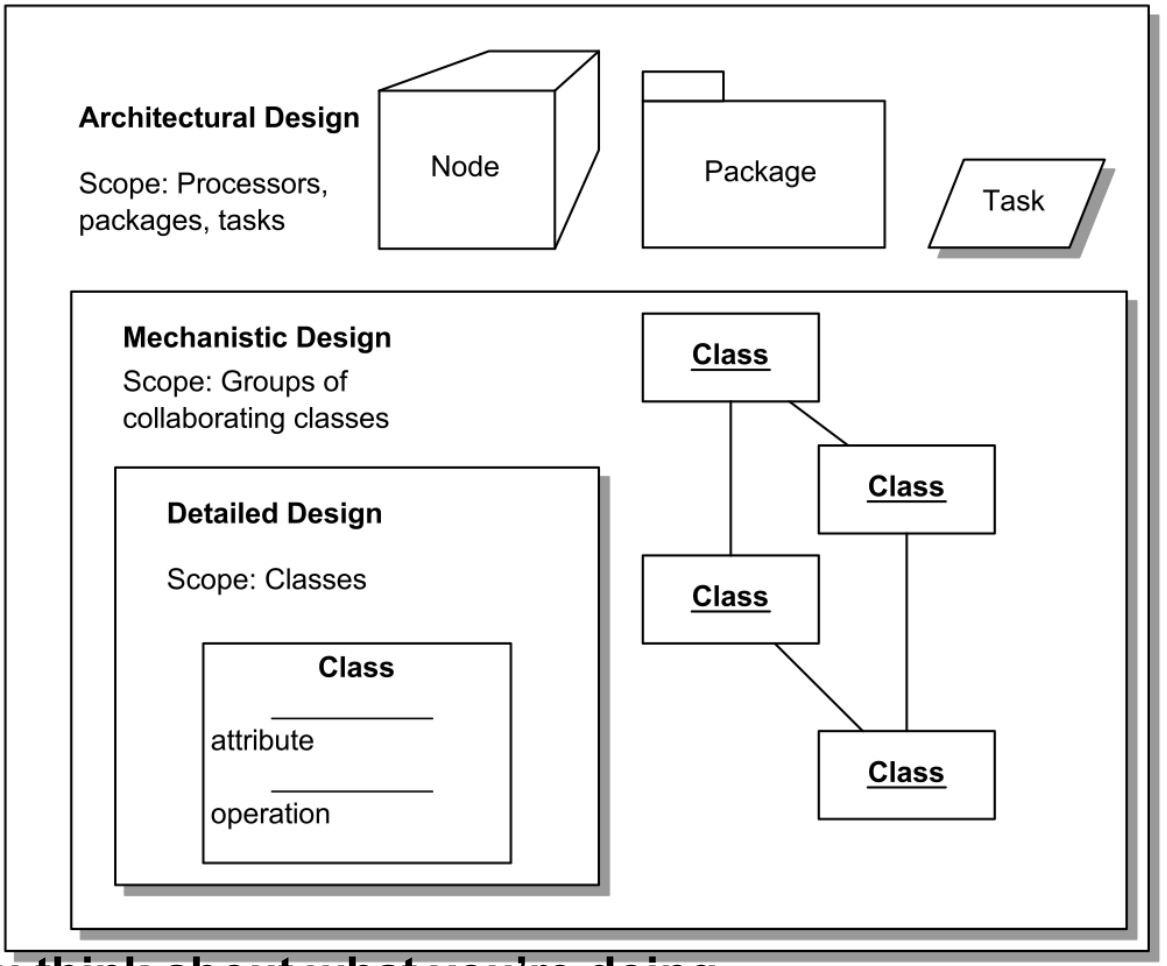
Analysis and design would make it clear the project is not feasible

Design

System architecture

Individual project

Specific task



"Design" is how you think about what you're doing

Design Levels: an analogy

Imagine the project is not to build software but to go on an inter-planetary journey...

Architectural design

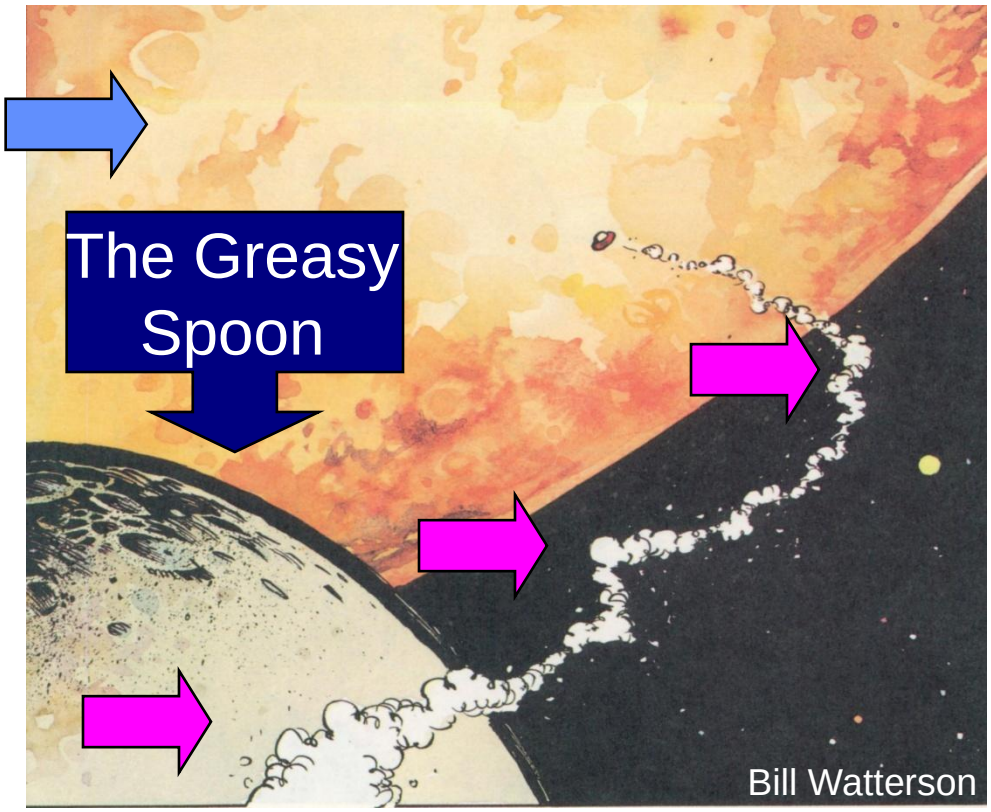
decide which planet to fly to

Mechanistic design

select the flight path

Detailed design

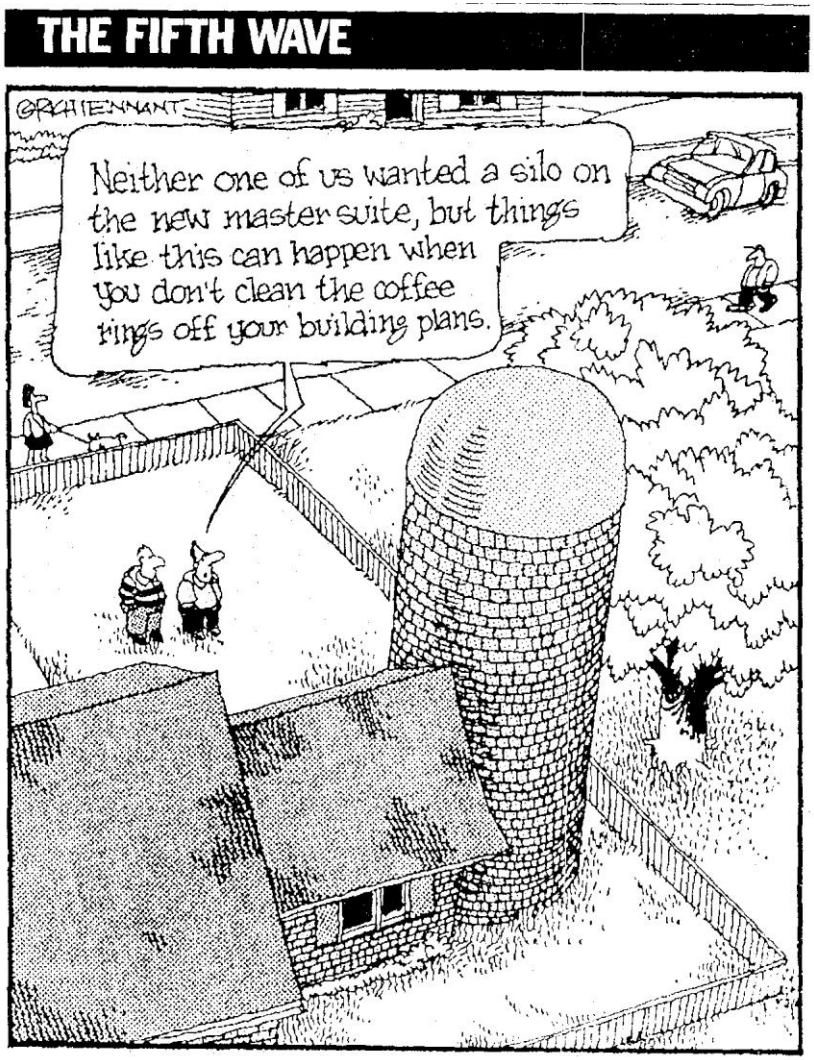
choose where to have lunch



Architectural design

Goals

- Capture major interfaces between subsystems and packages early
- Be able to visualize and reason about the design in a common notation
- Be able to break work into smaller pieces that can be developed by different teams (concurrently)
- Acquire an understanding of non-functional constraints
  - programming languages and operating systems
  - technologies: distribution, concurrency, database, GUIs
  - component reuse



Architectural Design Qualities

A well designed architecture has certain qualities:

- **layered subsystems**
- **low inter-subsystem coupling**
- **robust, resilient and scalable**
- **high degree of reusable components**
- **clear interfaces**
- **driven by the most important and risky use cases**
- **EASY TO UNDERSTAND**

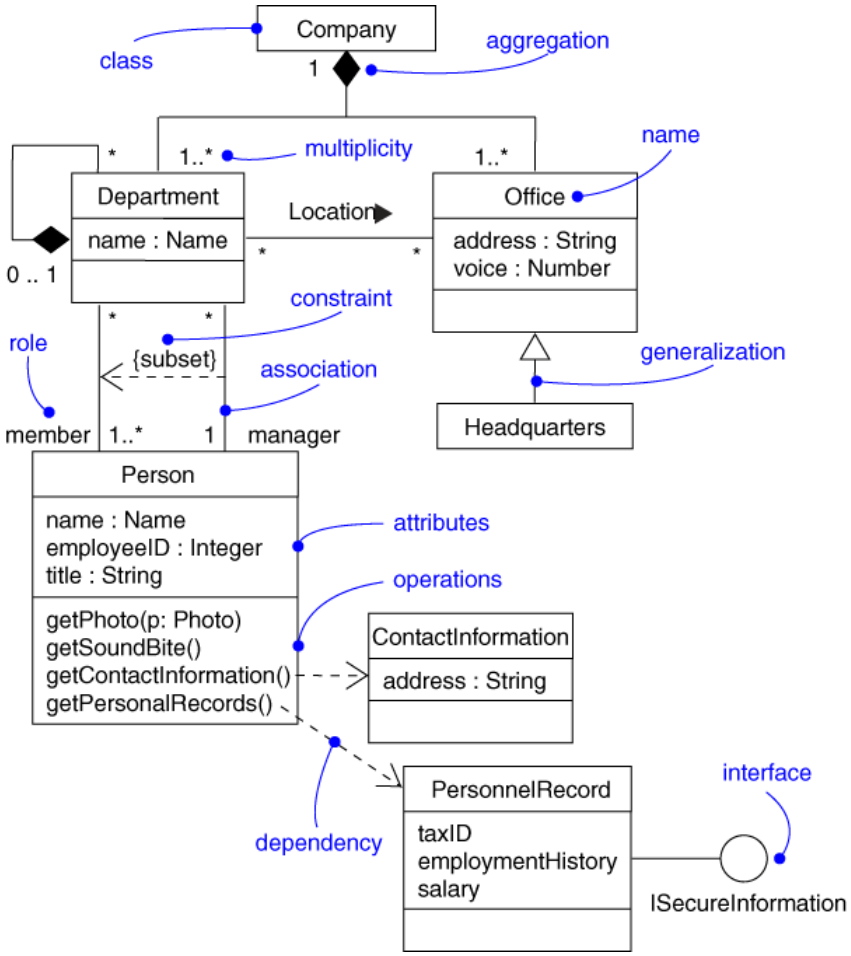
Mechanistic Design

Specify the details of inter-object collaboration *mechanisms*

- **Determine the *structure* of classes and their associations**
  - **Class diagram**
- **Determine the *behavior* of classes**
  - **Interaction diagrams**
    - **Collaboration**
    - **Sequence**
- **Target: The people working together**
  - **Over time & space**
  - **You can't do everything!**

Class Diagram

Describes the types of objects in the system and the various kinds of static relationships that exist between them



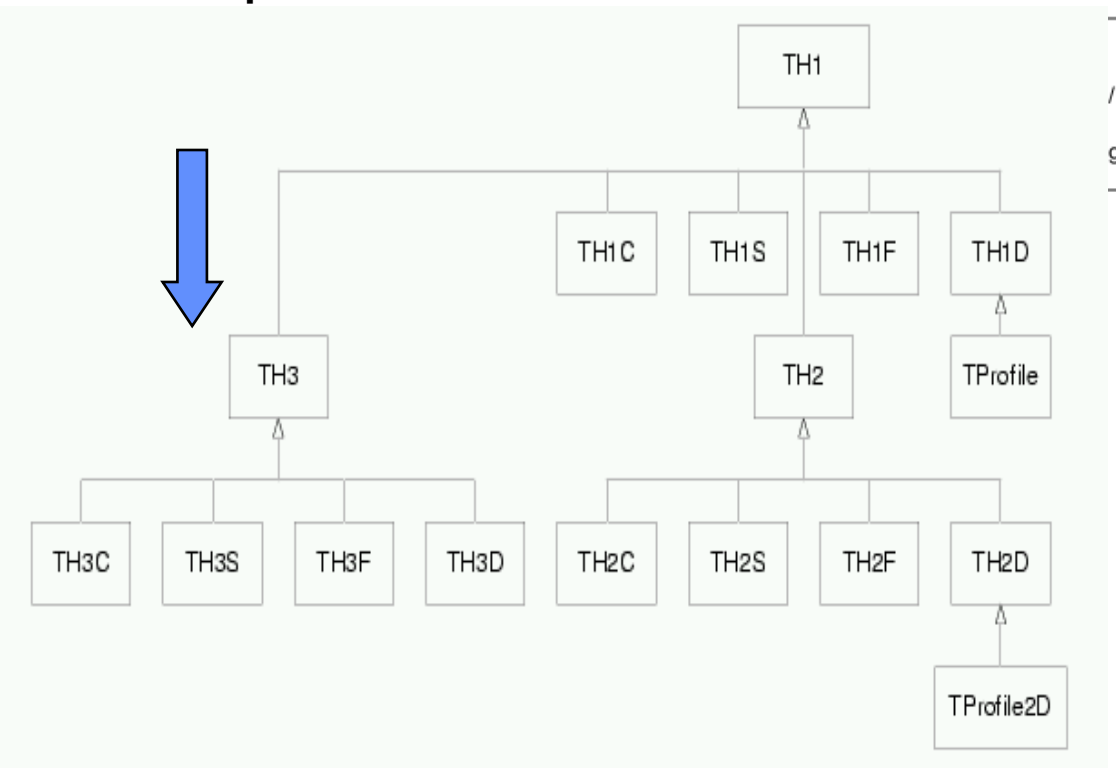
Rational Software Corporation

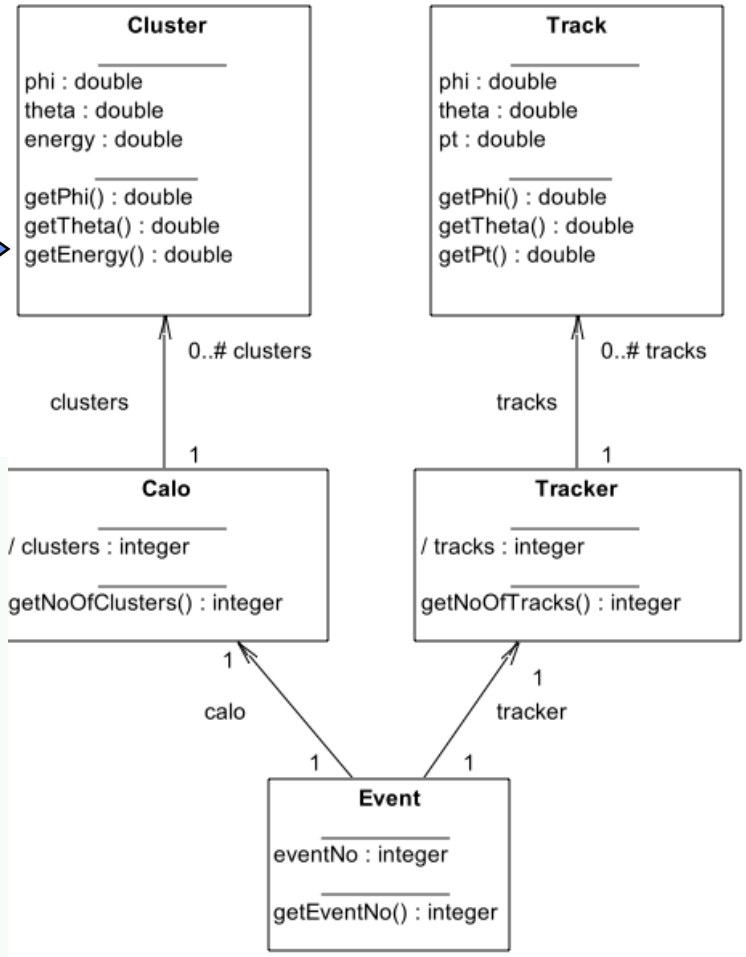
Example Class Diagrams

There are many possible designs

Goal: Allow you to reason about the strengths and weaknesses of a particular choice

Communicate through time and space



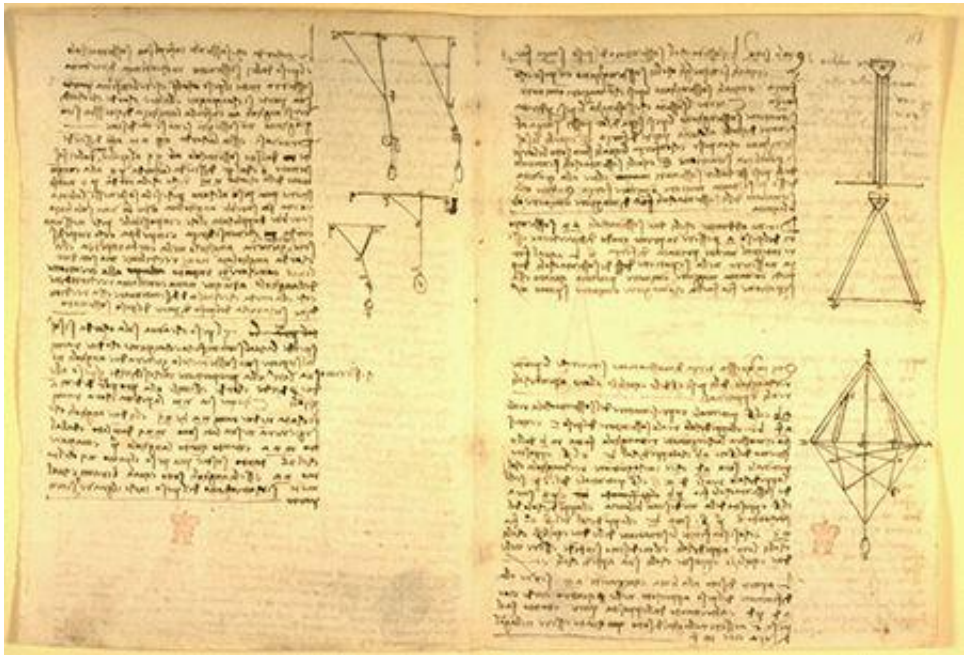


Design

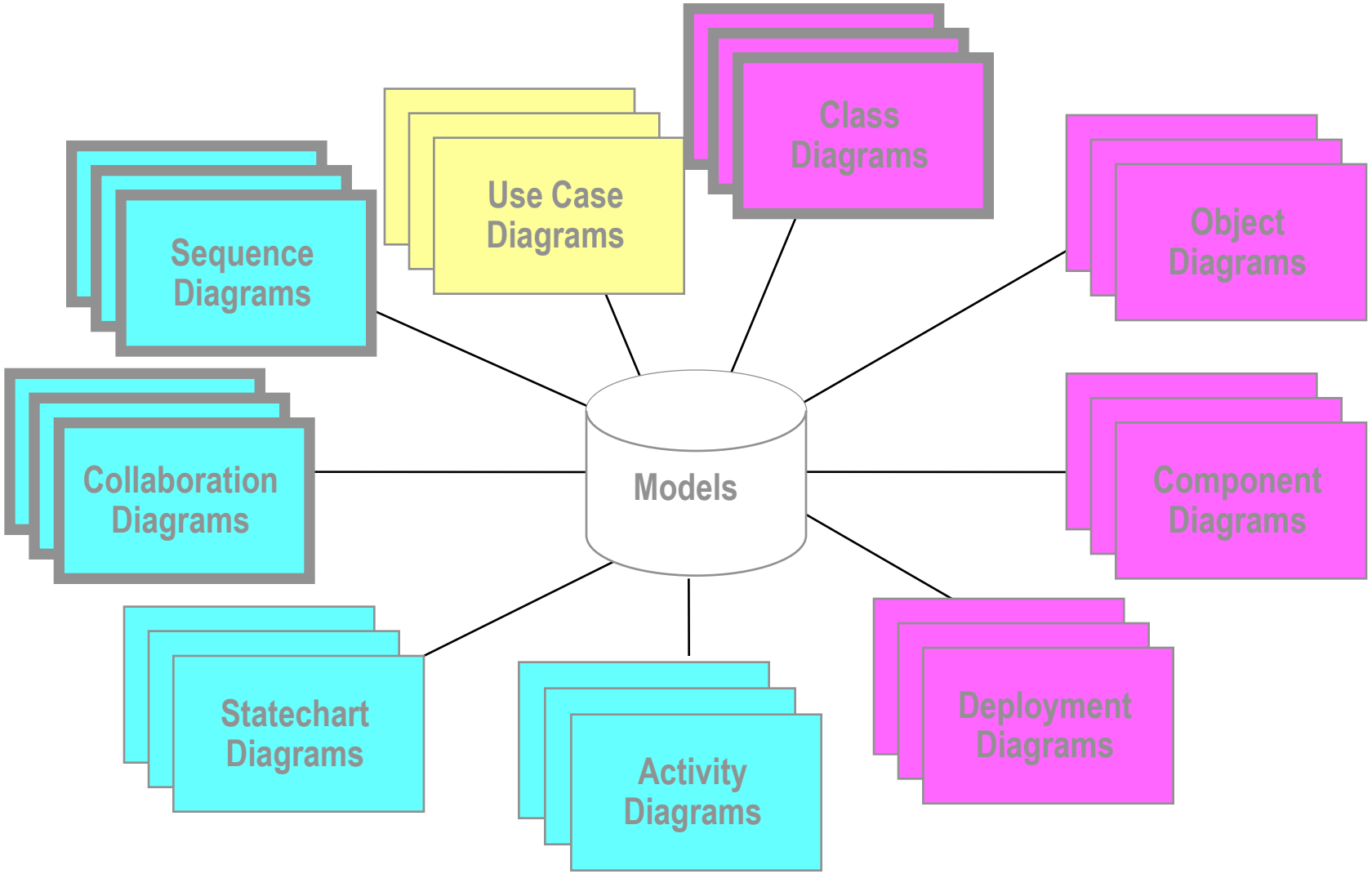
Specify the details of inter-object collaboration *mechanisms*

- **Determine the *structure* of classes and their associations**
  - **Relationships of access, ownership, authority**
- **Determine the *behavior* of classes**
  - **E.g. Interactions with other objects**
    - **Collaboration**
    - **Sequence**

How do we record and communicate this?

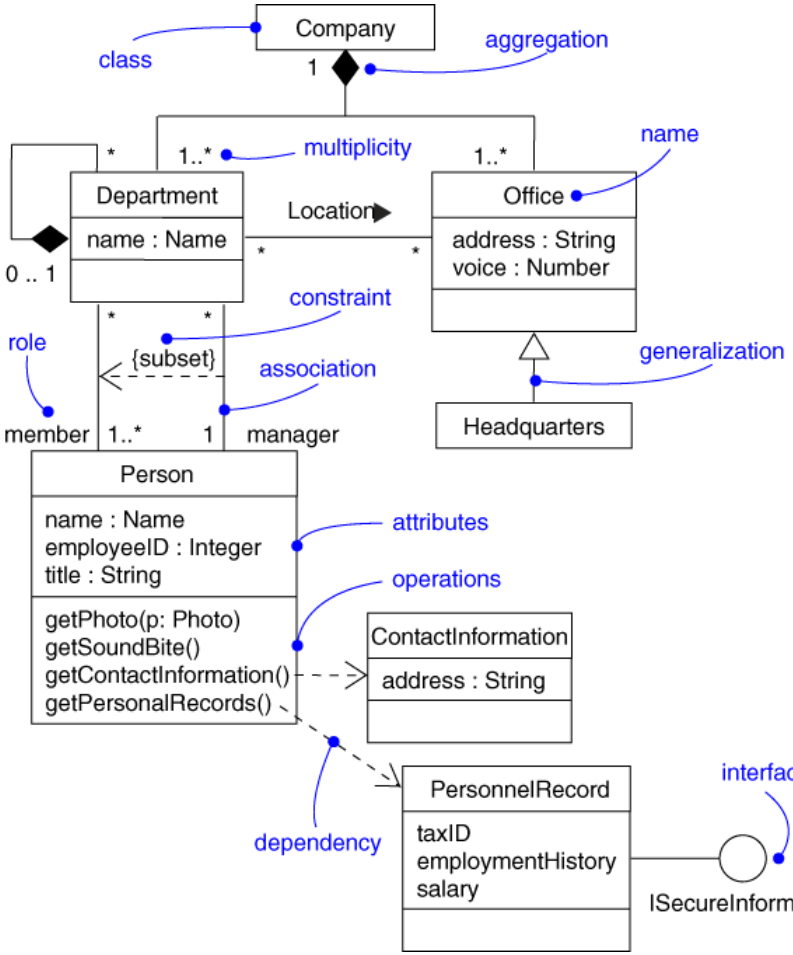


UML Diagrams



Class Diagram

Describes the types of objects in the system and the various kinds of static relationships that exist between them

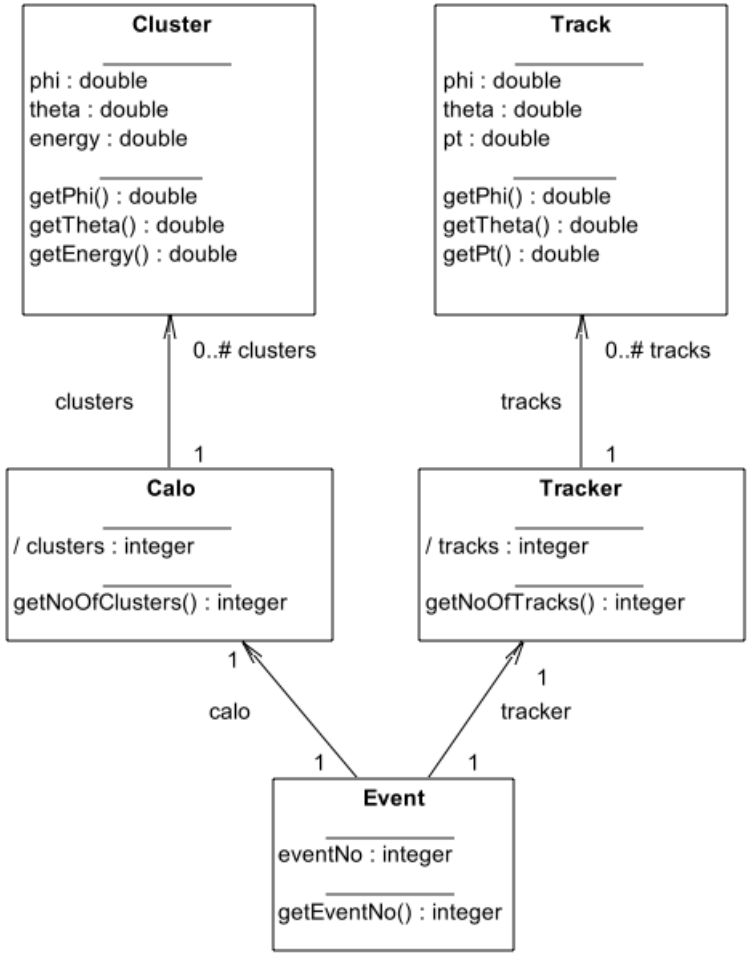


Rational Software Corporation

Example Class Diagrams

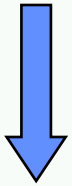
LHC++/Anaphe:

Event structure as defined in DDL file for populateDb exercise



ROOT:

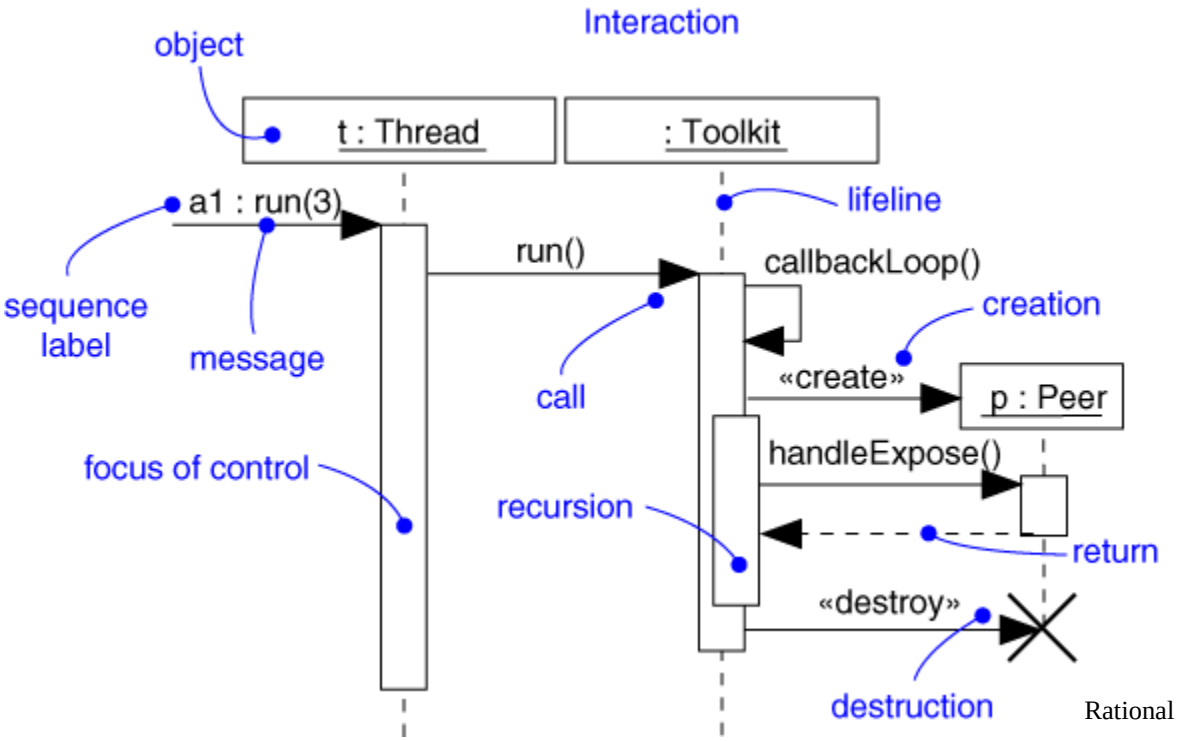
Histogram classes



Sequence Diagram

Captures dynamic behavior (time-oriented)

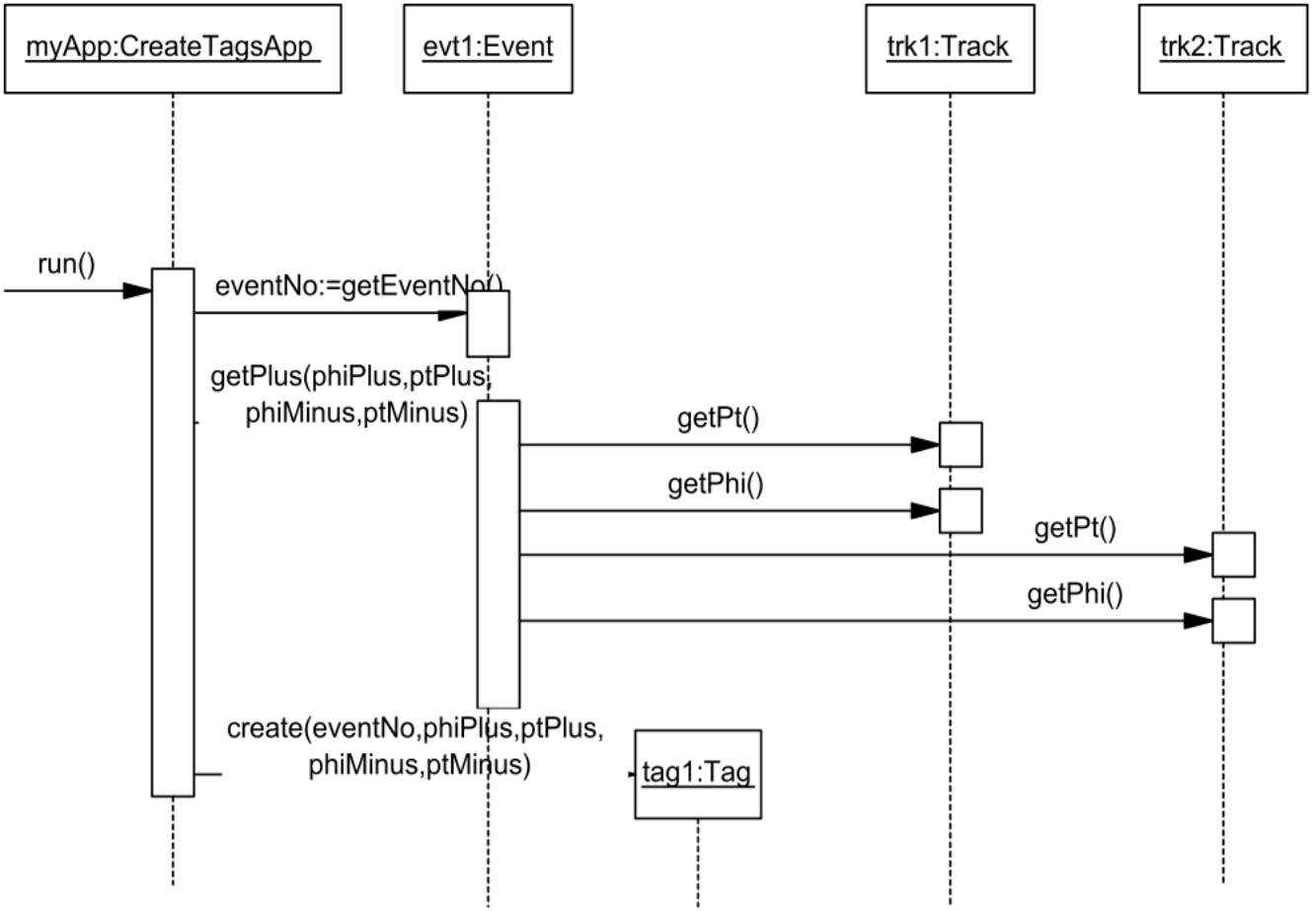
- **Model flow of control**
- **Illustrate typical scenarios**



Rational Software Corporation

Example Sequence diagram

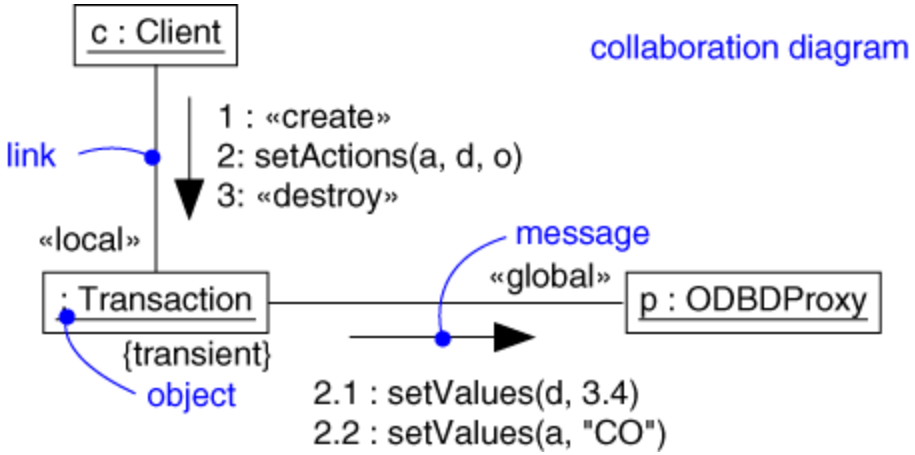
LHC++/Anaphe: scenario for createTag exercise with 1 event and 2 tracks



Collaboration Diagram

Captures dynamic behavior (message-oriented)

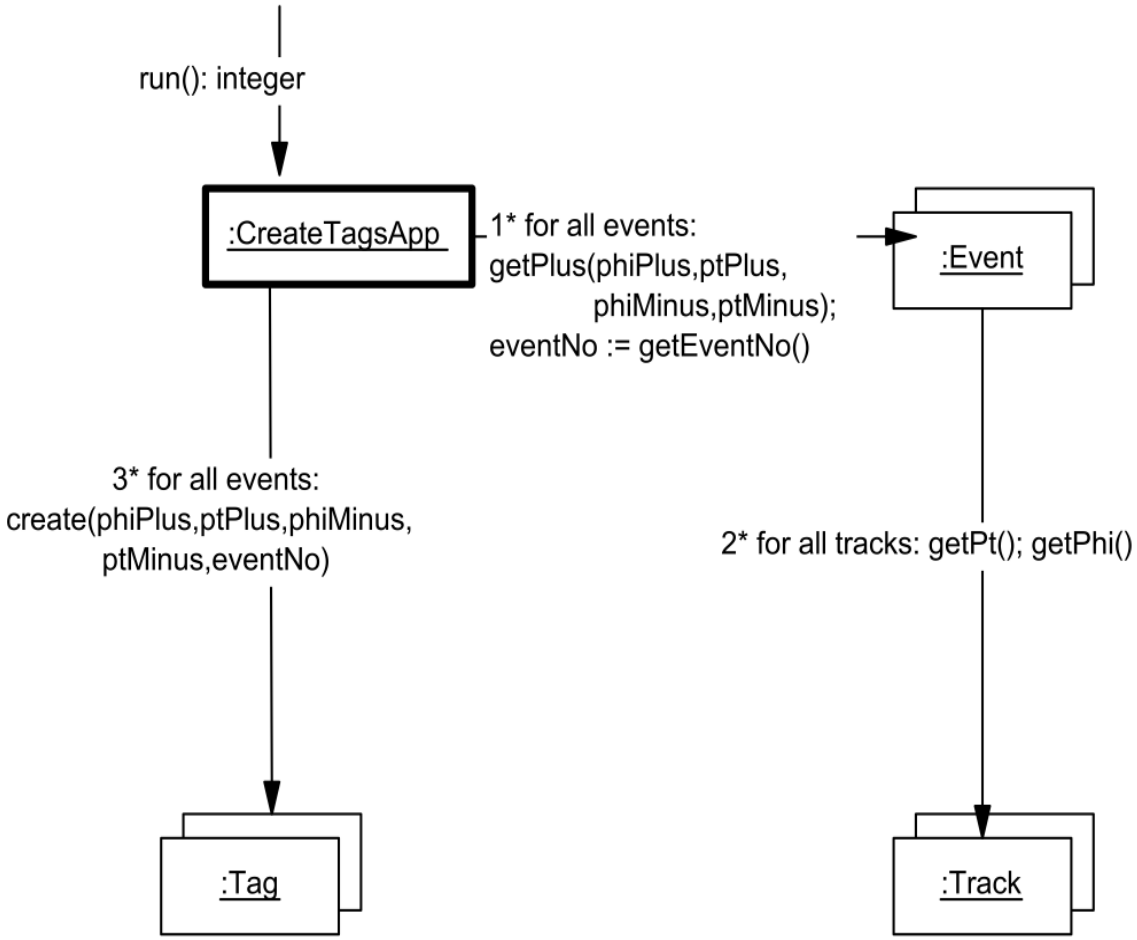
- **Model flow of control**
- **Illustrate coordination of object structure and control**



Rational Software Corporation

Example Collaboration Diagram

LHC++/Anaphe: messages between classes for CreateTag exercise



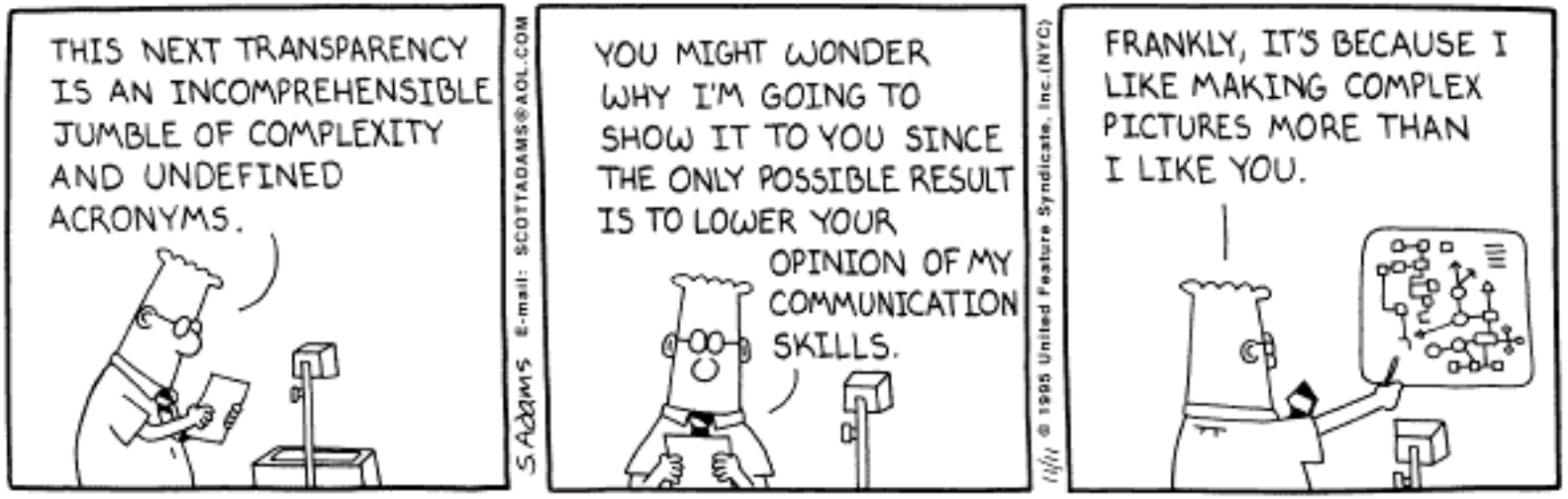
"These are complicated"

"So is field theory"

•Which is physicist-speak for "I don't get it either, so I'll call it 'trivial'"

"It's just notation"

•The notation is complicated because it's representing a complicated thing



"Yes, and how do we know they're right?"

- **That's the key question.**

Example: Linear Algebra

Physics code contains lots of linear algebra: A*X+B

- **Where A, X and B are more than just numbers: vectors, matrices**

Complicated operations:

- **Only some operations are OK**
  - **Can't add, dot-product vectors of different sizes**
  - **Dimensions must agree for vector-matrix multiplication**
- **But within those rules, users don't want to care about restrictions**
  - **A measurement might be a 1D, 2D or 3D constraint, but same formula to use it**

What are the trade-offs for a "linear algebra library"? For users?

- **Time & space of the linear algebra code**
- **Ease of use**
- **Time & space of the using code**
- **Correctness of answers**

Implementation: Vector3, Matrix32

```
class Vector3 {
float values[3];
float dotWith(Vector3 v) {...}
Vector3 add(Vector3 v) {...}
...
}
```

```
class Matrix33 {
float values[3,3];
Vector3 multiply(Vector3 v) {...}
Matrix33 add(Matrix33 v) {...}
...
}
```

Does the job

- Once you've created one of these, you can just string together operations
- Code to implement each method is quite simple

- Almost can't resist operator overloading to A*X+B

Does the job

- **Once you've created one of these, you can just string together operations**
- **Code to implement each method is quite simple**

But needs lots and lots and lots of methods

- **Vector3 can multiply Matrix32, Matrix33, Matrix34, Matrix35, Matrix36, ...**
- **Similar numbers for matrix multiplication**
- **Large amount of duplicated code to make a general library**

Can we get smarter with inheritance?

- **Matrix class, with Matrix32, Matrix33, Matrix34 as subclasses**
- **Methods then take and return Matrix objects**

Problem: Implementation of methods still needs to know

- **Methods require size information, access to individual elements**

  Different size internal arrays need to be accessed, compiler wants to know

- **Lots of work to get those**
- **And methods still need to call "new Matrix32" vs "new**

General Implementation: Vector, Matrix

```
class Vector {
int dim;
float *values[dim];
float dotWith(Vector v) {...}
Vector add(Vector v) {...}
...
}
```

```
class Matrix {
int dim1, dim2;
float *values[dim1, dim2];
Vector multiply(Vector v) {...}
Matrix add(Matrix v) {...}
...
}
```

Again does the job

- Once you've created one of these, you can just string together operations
- Code to implement each method is almost as simple
  - "Just has to" keep track of index dimensions, and do one indirection
  - Return types are fixed, so only need to handle one "new"

Strong, general approach for a library, but at what cost?

Costs:

Type checking at runtime, not compile time

Memory allocation from heap (“new”) always, not stack or static

Extra indirection to access any element

...

It’s an experimental question whether these matter!

Tradeoffs:

Direct structure

Minimal memory use:

Compiler handles limits, allocates data as part of object

Fast allocate/deallocate:

Vector[5] is just one long allocation & 5 ctor calls

More complicated user code:

You have to explicitly specify classes for intermediate variables, etc; can't pass common super-types

Indirect structure

More memory needed:

Virtual table pointer

Length values

Pointer to memory

Allocate/deallocate is more work:

Vector[5] is one allocation, 5 ctor calls, then 5 more allocations

User code simple, general:

All objects are same basic type

Code can be written without reference to specific sizes

When there's no perfect answer, you're in the realm of tradeoffs

Start with the general, and replace with specific when needed?