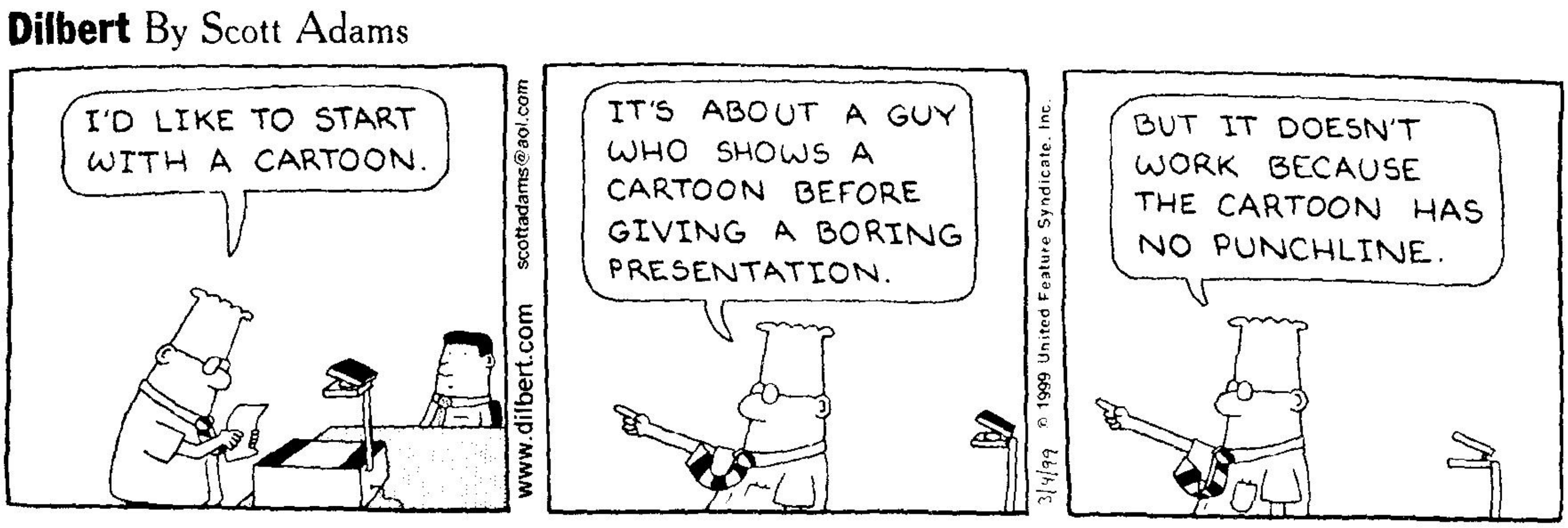
Introduction To Software Engineering



With thanks to Bob Jones for ideas and illustrations

Two recurring terms: "Processes" and "Models"

A Process:

- A set of partially ordered steps intended to reach a goal
- In software engineering the goal is to build or enhance a software product
- Defines **who** is doing **what**, **when** and **how** to reach a certain goal - *Ivor Jacobson*

A Model:

- A model is a description of a system from a particular perspective
- Models are created as part of the defined process

"Why do we have to formalize these?"

Scale and process: Building a dog house

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

Rational Software Corporation

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

Scale and process: Building a skyscraper

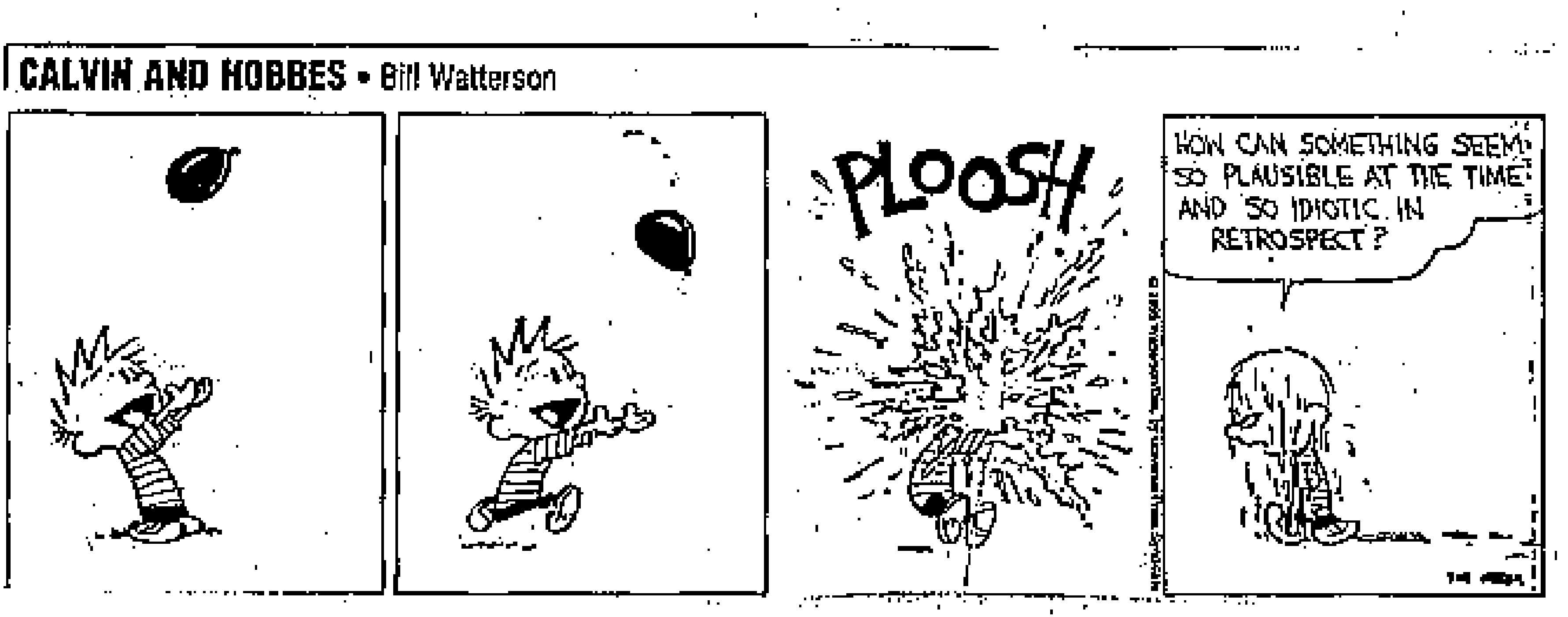
- Built by many companies
- Modeling
  - Simple plans, evolving to blueprints
  - Scale models
  - Engineering plans
- Well-defined process
  - Architectural team
  - Political planning
  - Infrastructure planning
  - Time-tabling and scheduling
  - Selling space
- Heavy equipment
- Major risks

Rational Software Corporation

Why do software projects fail?

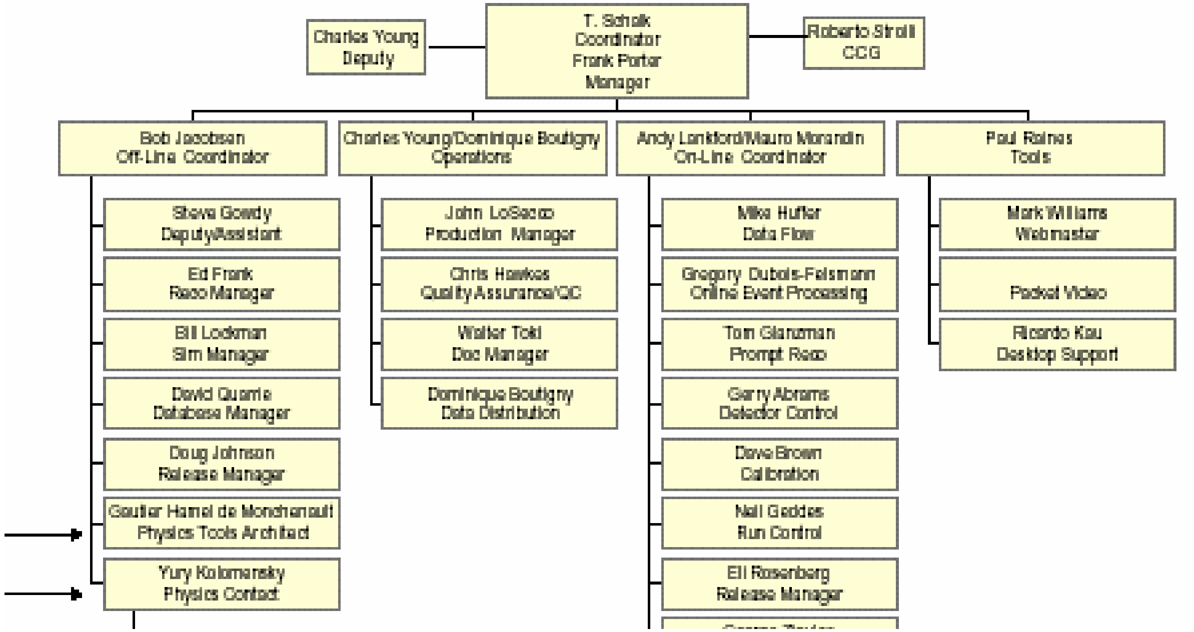
Even if you do produce the code it does not guarantee that the project will be a success

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



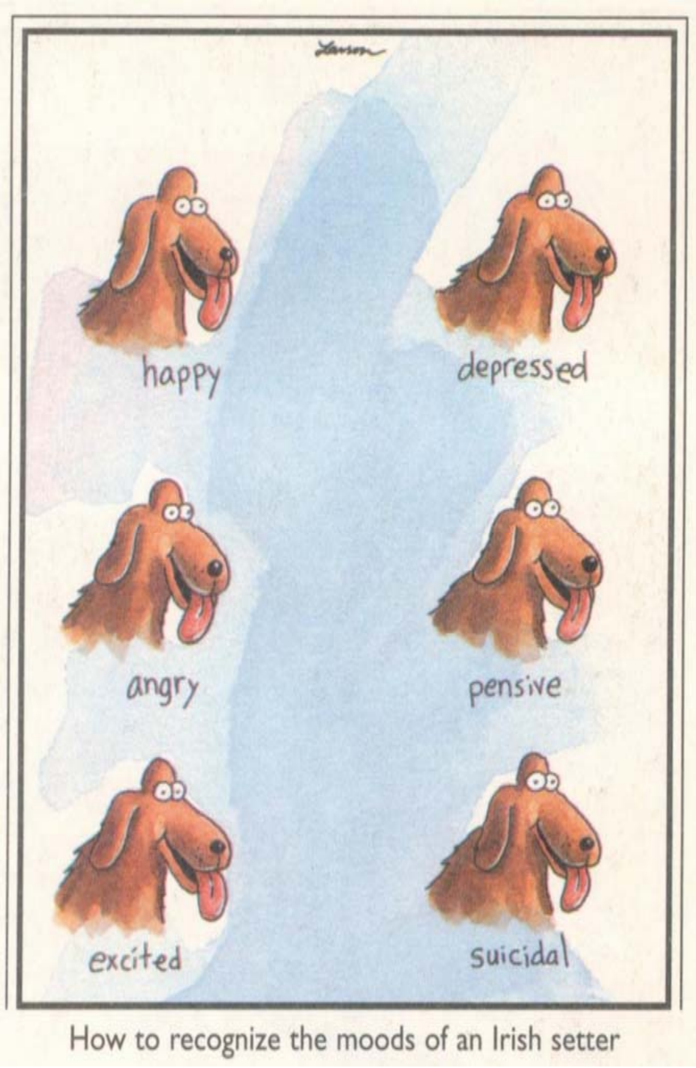
Communication explosion

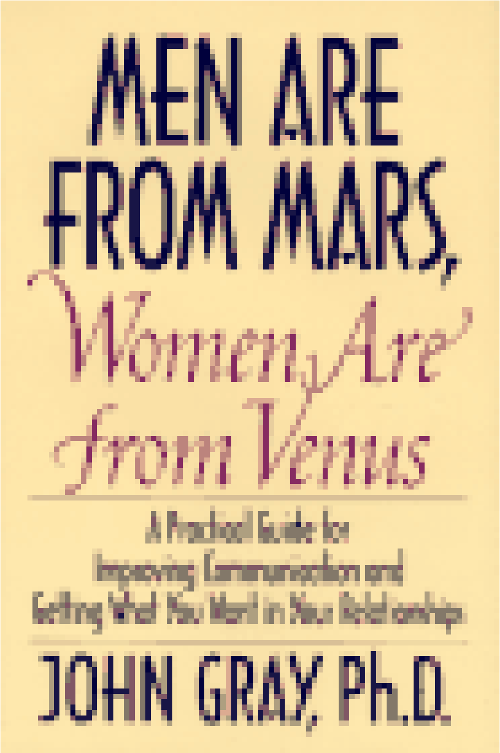
More people means *more* time communicating which means more misunderstandings and *less* time for the software



Why software projects fail...

Misunderstandings between users/developers/sponsors





Analysis and design can catch such misunderstandings

Why software projects fail...

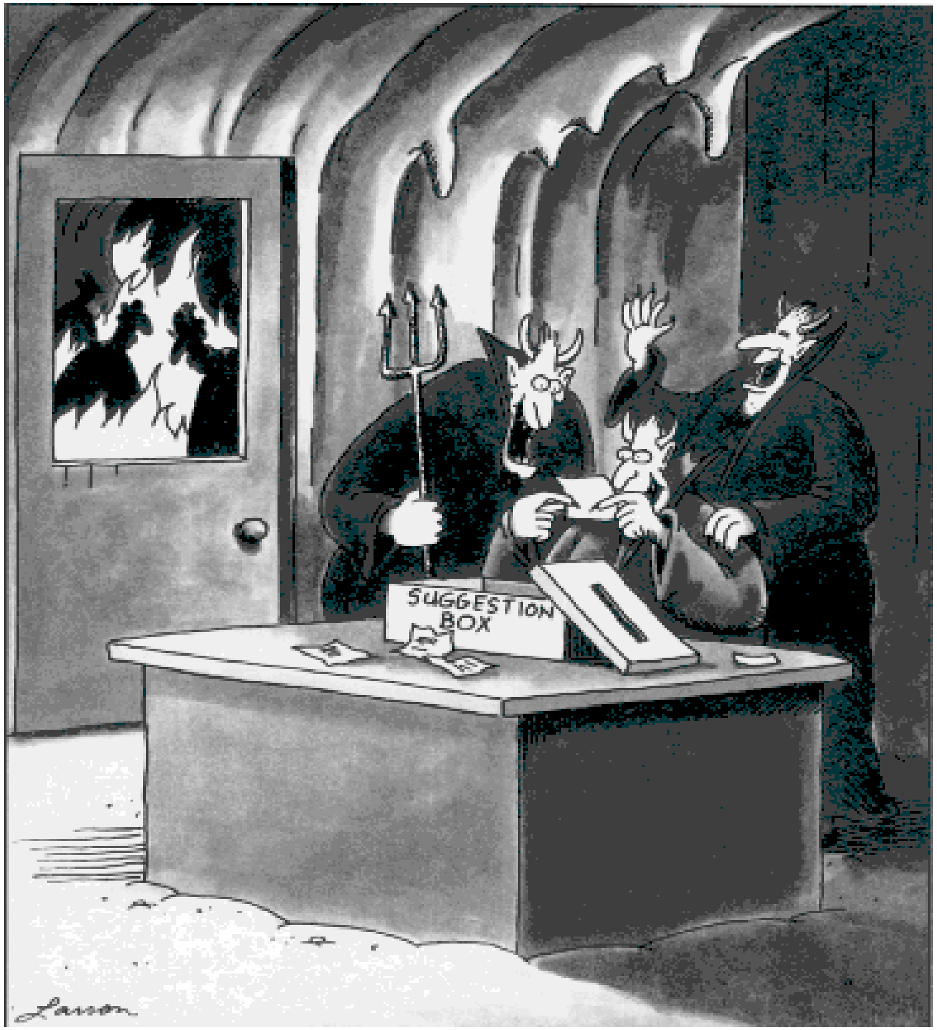
Undefined responsibilities

"Hey... this could be the chief"

Project planning can help identify needed responsibilities

Why software projects fail...

Missed user requirements



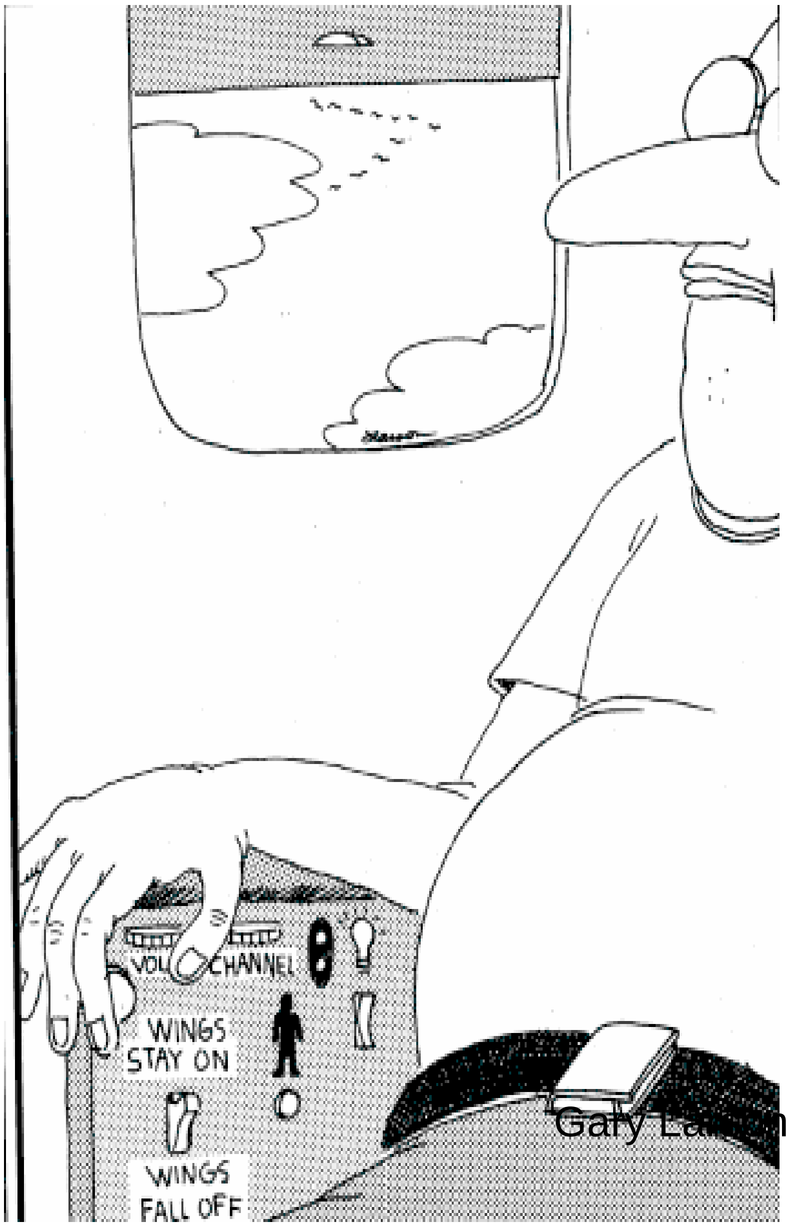
Write down and discuss requirements with the users

Iterate to get them right

Why software projects fail...

Badly defined interfaces

Fumbling for his recline button, Bob unwittingly instigates a disaster



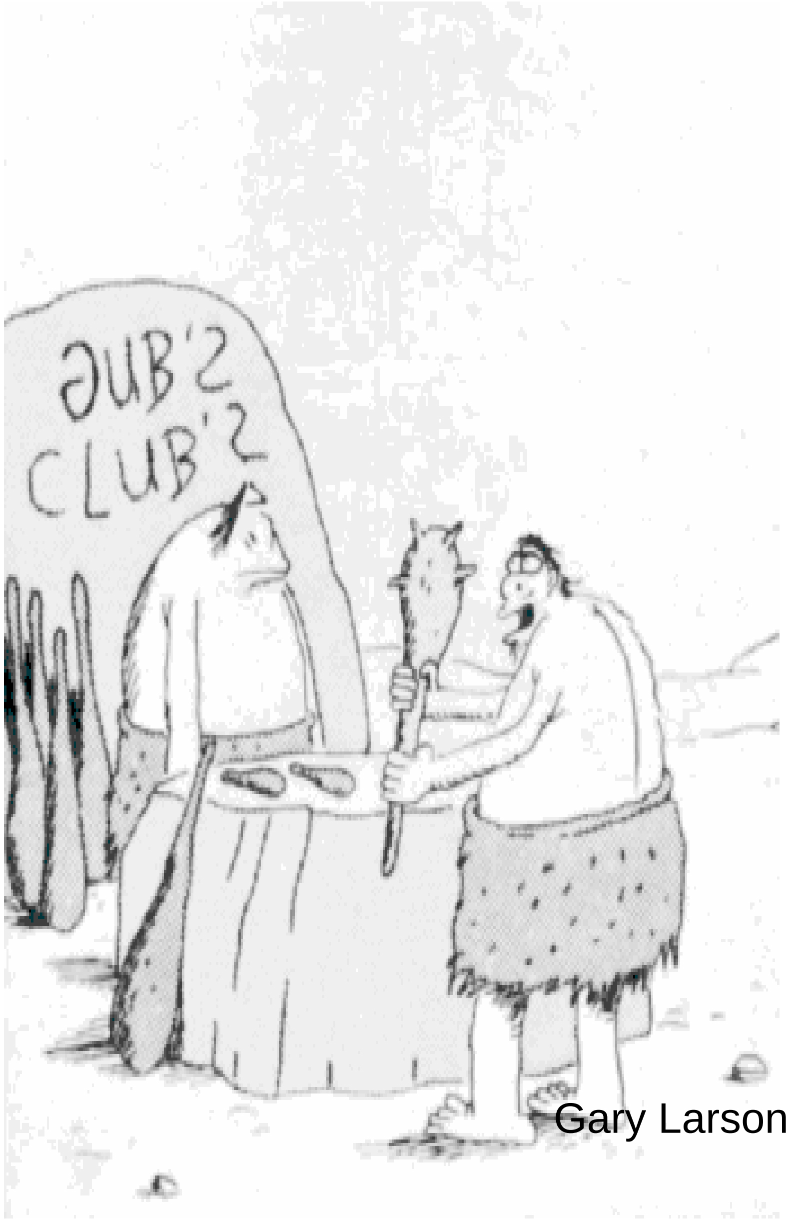
Gary Larson

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"



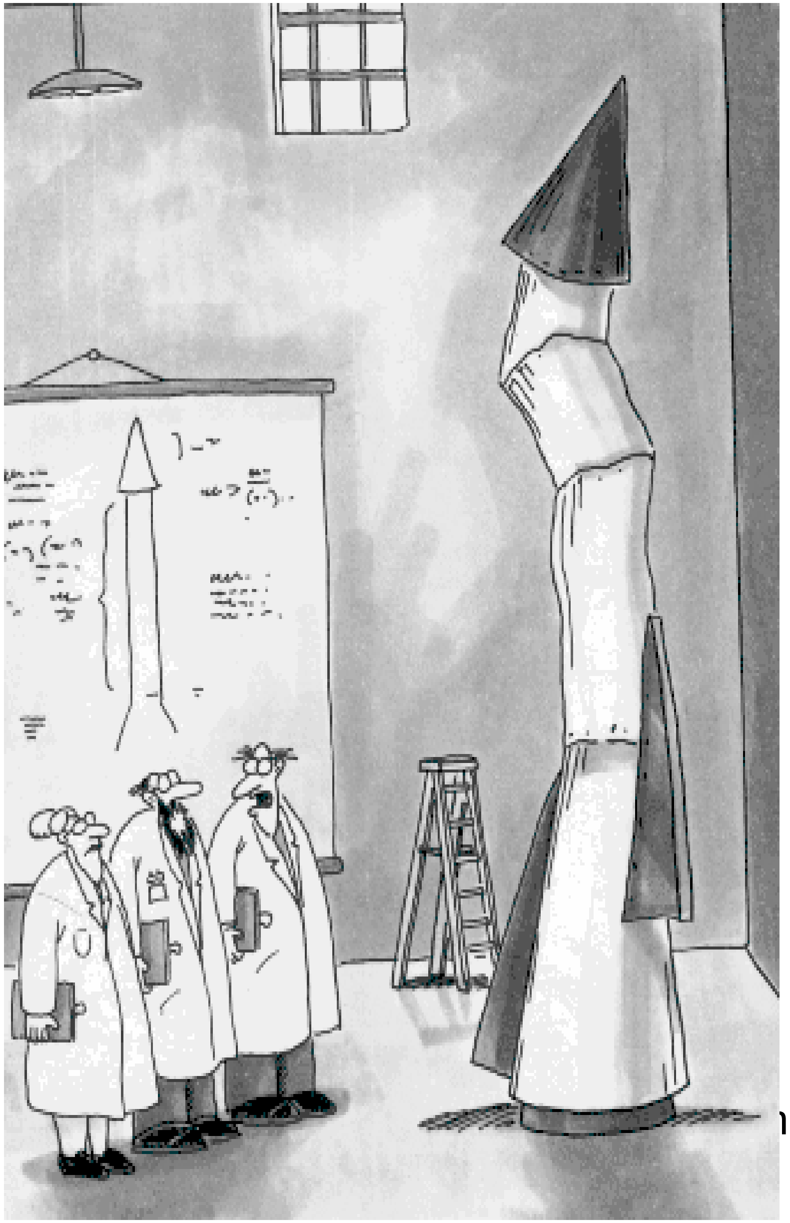
Gary Larson

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

The life time of HEP software

Software is a long-term commitment

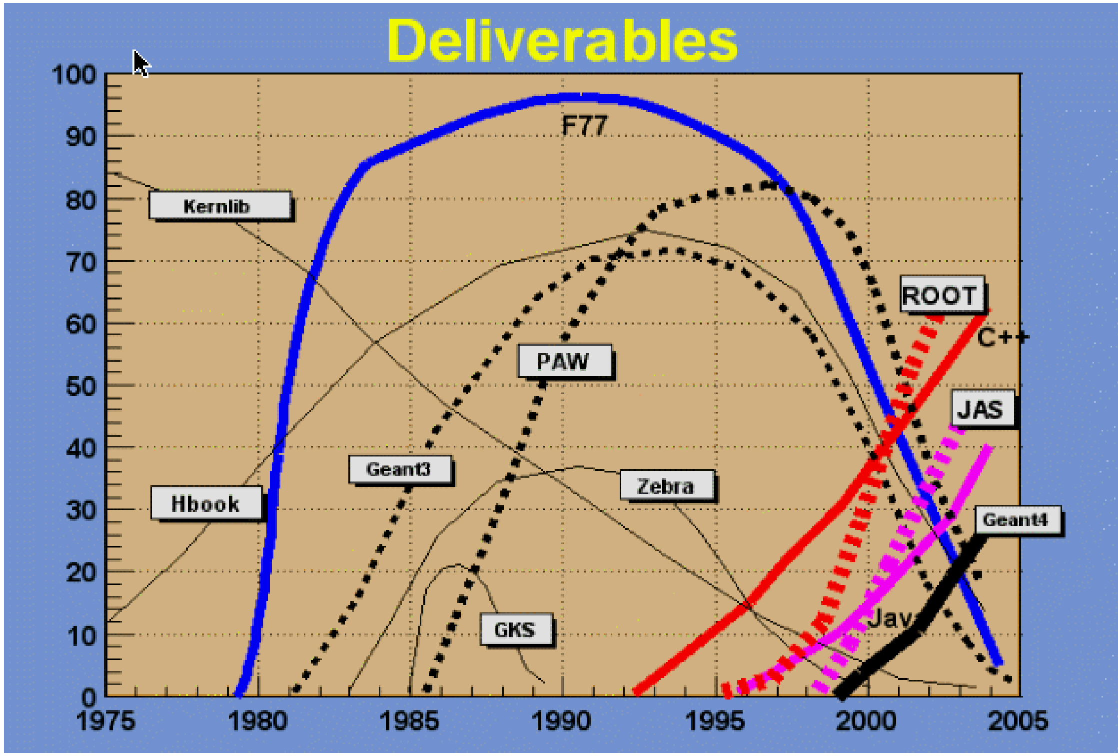
Users like stable and maintained systems

Vote with their feet

It takes time to develop a new system

- Geant3 6+ yrs 3 people 300 KLOCs
- PAW 6+ yrs 5 people 300
- Zebra 4+ yrs 2 people 100
- ROOT 5* yrs 3 people 630
- Working system after 1 year.

Real work is after that !!



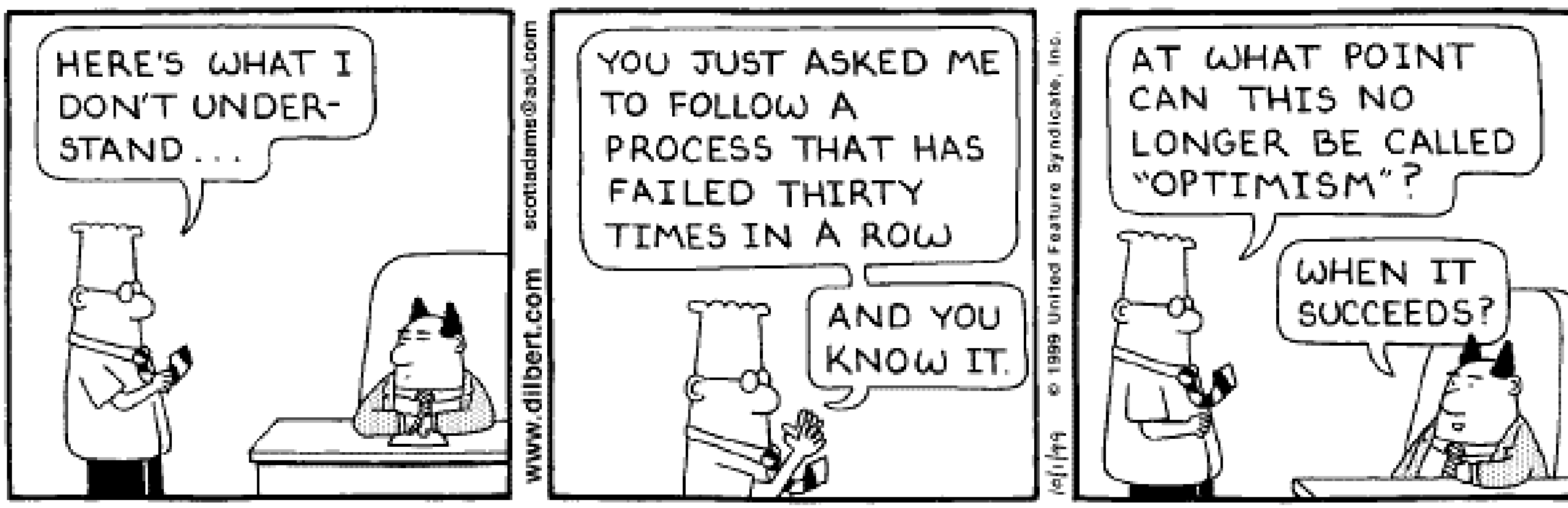
Many releases of the software are needed over its lifetime to fix bugs, add new features, support new platforms etc

How do we cope?

We try to find a way of working that leads to success

- We create a "process" for building systems
- We devise methods of communicating and record keeping: "models"
- We use the best tools & methods we can lay our hands on

And we engage in denial:



So many software processes!

“OMT”, “Booch”, “Objectory”, “Unified”,…

People have been defining and promoting processes for decades

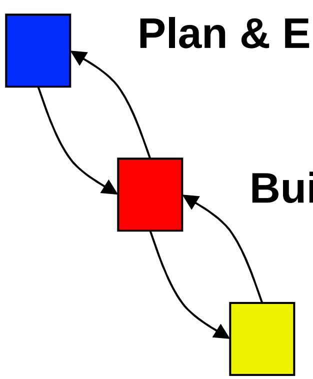
- Millions of books sold & conference talks given

But, much commonality between them:

- Process stages

  Plan and Elaborate, Build, Deploy

- Iterative software development



Plan & Elaborate - define scope of project and justify its need

plan project, specify features & baseline architecture

Build - construct the project

Deploy - move the software to the users site

- Models constructed

  Use cases, class diagrams, interaction diagrams, …

The Unified Software Development Process

Published in 1998 (http://www.rational.com)

Key concepts

- Iterative
- Architecture-centric
- Use-case driven
- Risk confronting

Describes a list of tasks to follow to develop software

- Not all tasks are required for or even applicable for all development projects

How do we represent the development process?

Through models

- The language of the designer
- Representations of the system that allow reasoning about some characteristic of the real system
- Vehicle for communications with various stakeholders
- Visual

Through views

- View = simplified model (slice of model)
- An architectural view is an abstraction of a system from a particular perspective or vantage point, covering particular concerns, and omitting entities that are not relevant to this perspective

How do we document models and views?

Use a standard language and diagramming method

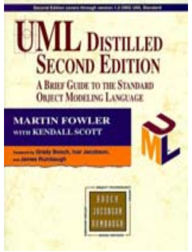
- Unified Modelling Language (UML)
- Standardized by the Object Management Group (OMG) in 1997

  http://www.omg.org
- A(nother) language for representing a sw. dev. process

  Booch, Rumbaugh, etc all defined their own languages before UML
- UML is "process independent"

  it is a language for modelling, it does not define how to use the language to assist in software development

For more information see book
UML Distilled (2nd. Edition)
Martin Fowler et al, Addison-Wesley, 1999



Overview of UML

The UML is a language for

- Visualizing
- Specifying
- Constructing
- Documenting

Communicating

Covers all phases of software development process

What do people communicate with UML?

Requirements of a software system

- Use Cases

Structure/Architecture of a software system

- Class diagrams / (Object diagrams)

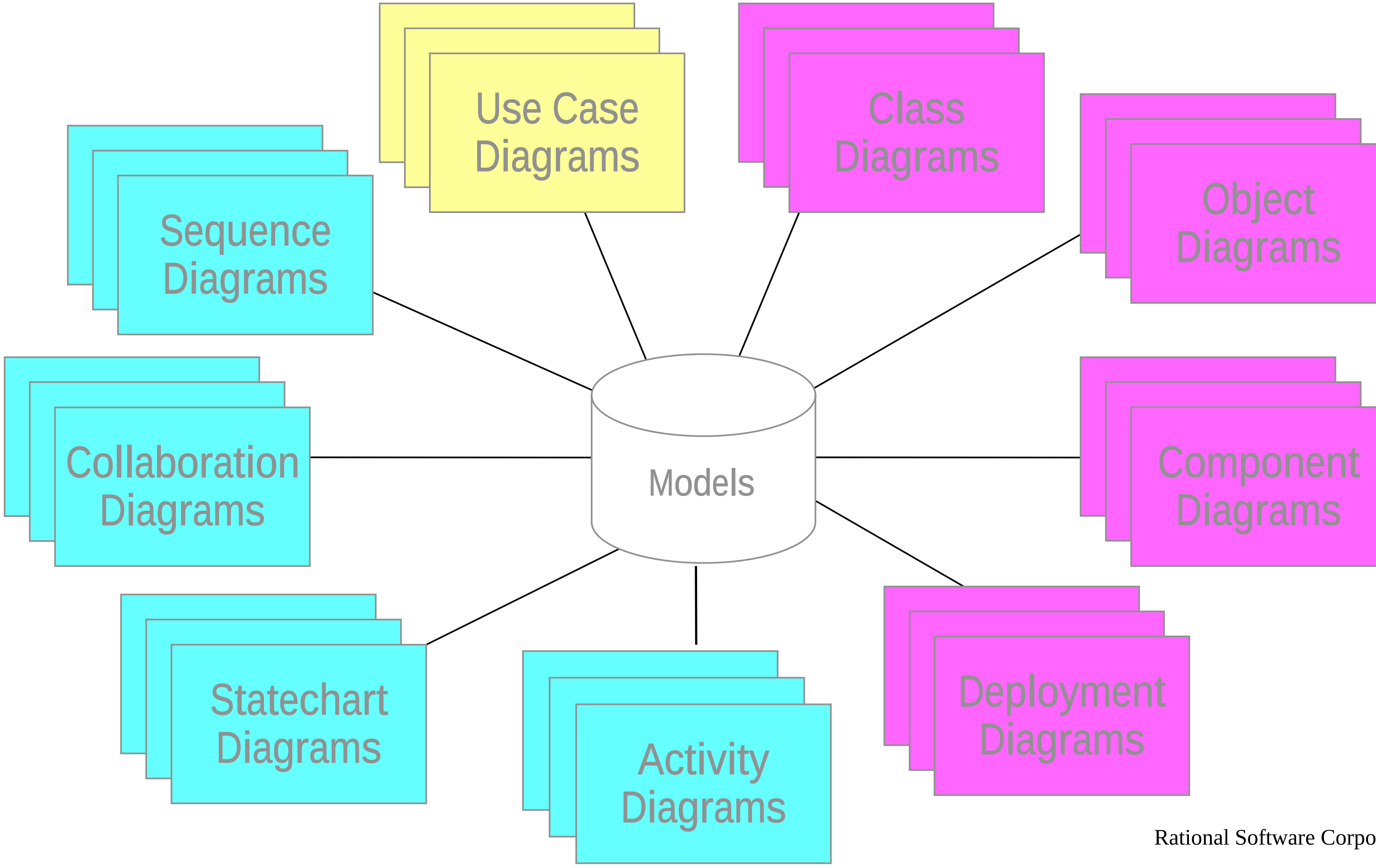
  Views that emphasize concepts, specifications, implementation

- Deployment diagrams
- Component/package diagrams

Dynamic behavior of a software system

- Sequence/Interaction (Collaboration) diagrams
- State charts
- Activity diagrams

UML Diagram Types

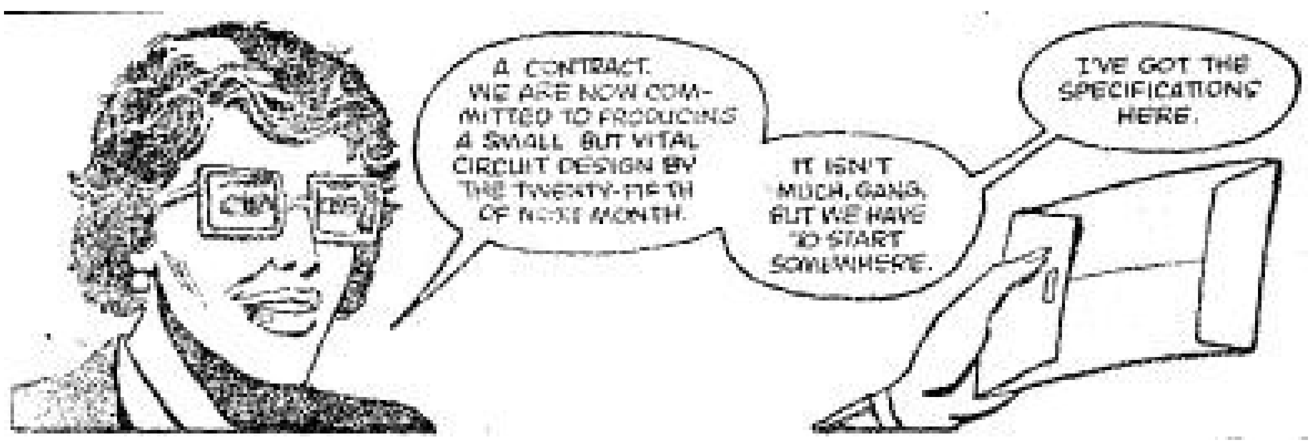


Rational Software Corporation

Requirements: What do we need to build?

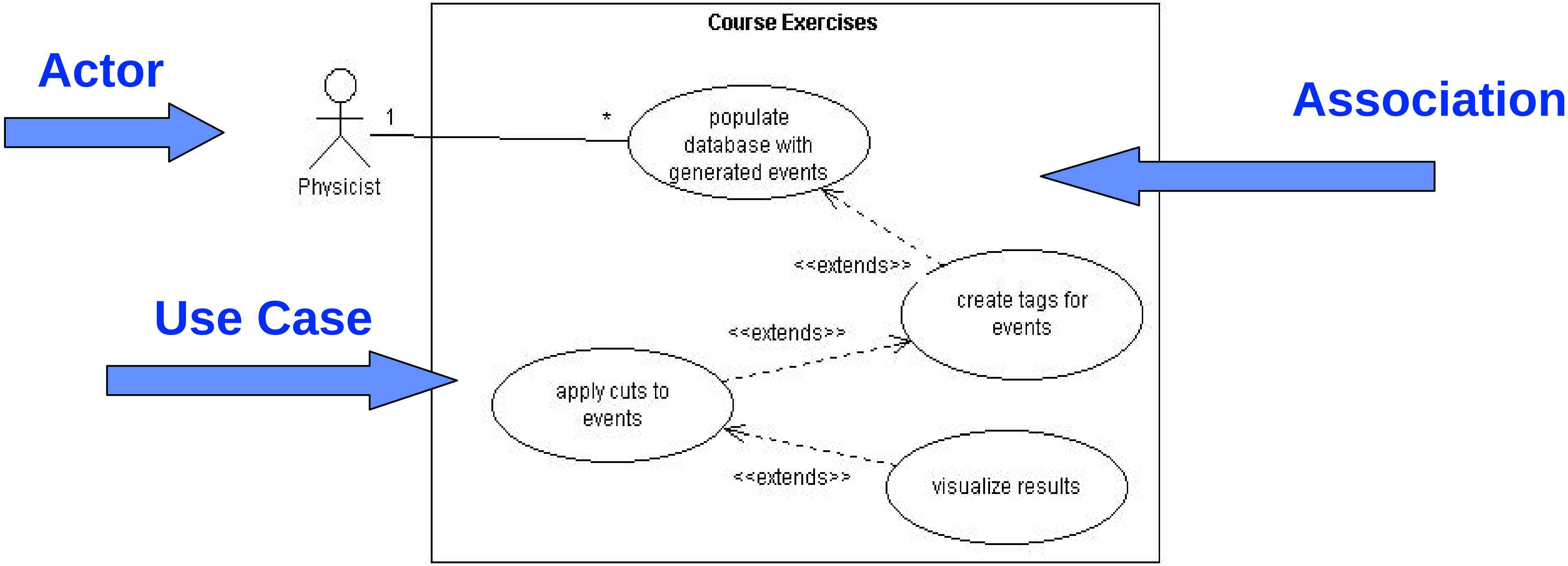
Initial description of needs/desires of a product

- Overview statement
- Customers/users
- Goals
- System functions - what is the system supposed to do?
- System attributes - what are desirable qualities of the system?



Capturing functional requirements with use cases

Captures system functionality as seen by users



A typical interaction between a user and the system under development

Use Cases

"Narrative document describing the sequence of events of an actor (external agent) using a system to complete the process" -*Ivar Jacobson*

- Not requirements or functional specifications, but they imply requirements

High-level use case format

Use Case:	*Create reduced data*
Actors:	*Physicist*
Type:	*primary (secondary/optional)*
Description:	*The physicist provides a reduction routine to be run, and selection criteria for the input data. The processing is done without further interaction, and when completed the reduced output is available to be selected for further processing.*

What does this buy us?

Use cases & the discussion surrounding their creation:

- Provide a high-level description for discussion with stakeholders
- Ensure that the requirements of the system are captured
- Help decompose tasks into small manageable entities
- Drive the conceptual/object model construction
- Ensure that important requirements are tackled early

The physicist provides a reduction routine to be run, and selection criteria for the input data. The processing is done without further interaction, and when completed the reduced output is available to be selected for further processing.

Ranking use cases

- Is this a main purpose of the system?

  Primary - major tasks

  Secondary - minor or rare tasks

  Optional - tasks that *may* be tackled

- Some other factors:

  Does the use case impact the overall architectural design?

  Is insight obtained with little effort?

  Is the use case risky, time critical or complex?

Use case summary

Use cases are part of the analysis phase

- Emphasize *what* rather than *how*

Use cases help lead to real functional requirements

- Good starting point
- Not performance & environmental constraints, etc. (see Contracts)

Use cases help scheduling

- Determine focus of project iterations and development

Use cases remain a focus as you develop

- "Can I do this one yet?"

Capturing structure with deployment diagrams

Shows the "configuration of run-time processing elements with the software components, processes and objects that live on them"

Includes

- Communication associations (networks)
- Nodes (processors)
- Components (software packages)
  - components can depend on other components
  - components can show objects

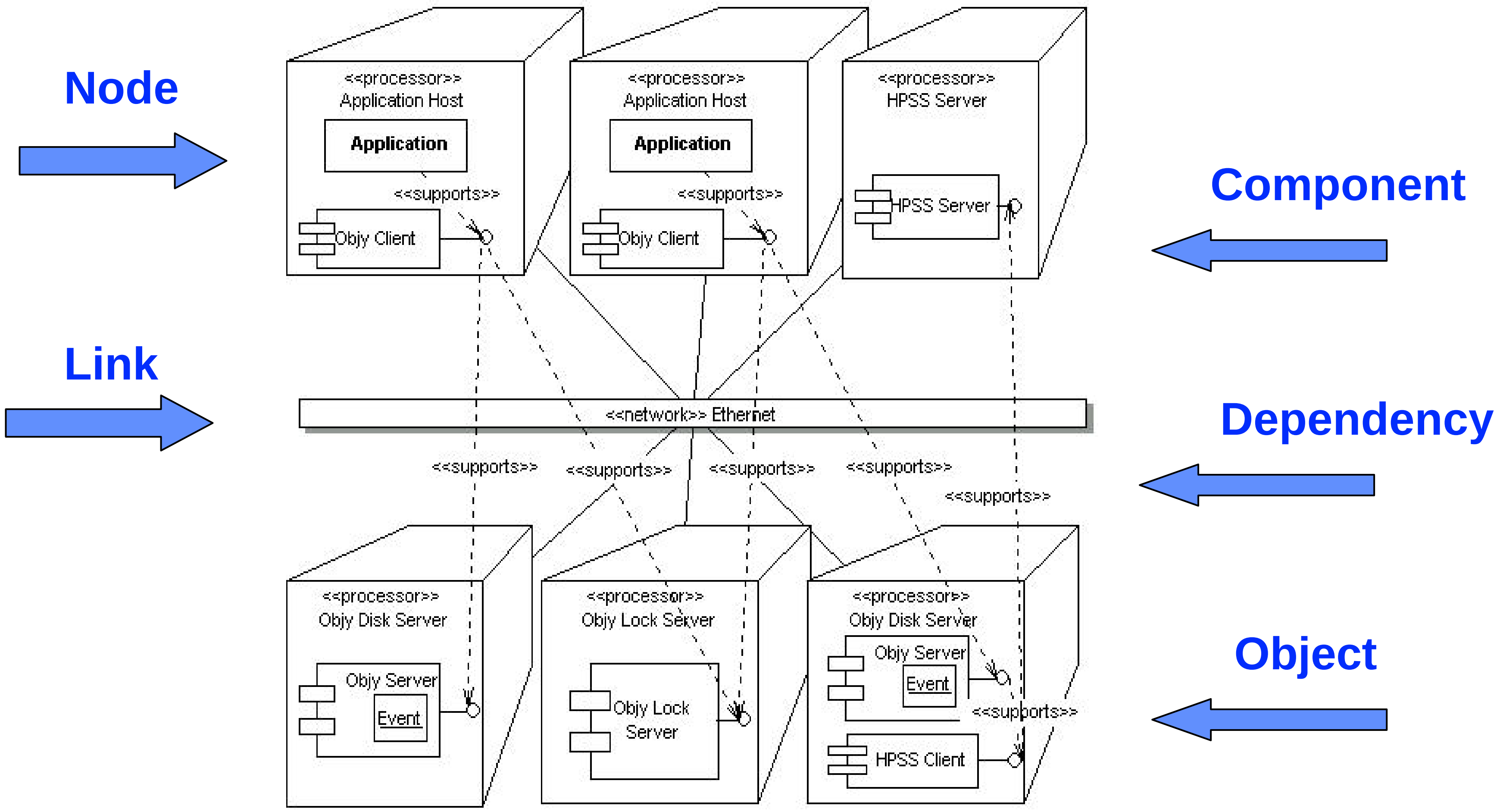
Often called system "architecture"

Architectural Design Qualities

A well designed architecture has certain qualities:

- Clear interfaces
- Layered subsystems
- Low inter-subsystem coupling
- Robust, resilient and scalable
- High degree of reusable components
- Driven by the most important and risky use cases
- EASY TO UNDERSTAND

Example Deployment Diagram



Process Summary

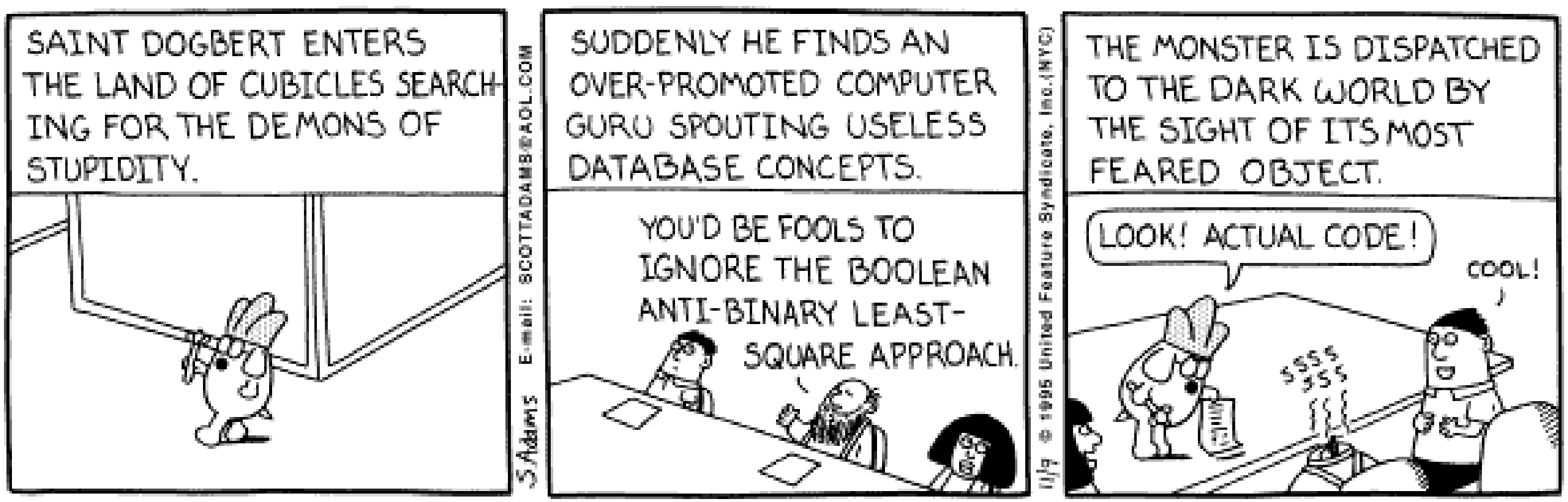
Choice of process depends on scale of problem

A process is a (partially ordered) set of tasks to develop and deploy a software system

The process should be

- iterative & architecture centric
- use-case driven & risk confronting

The Unified Modeling Language is a common/standard way to document the models and views of the process



Don’t become a process evangelist!

In closing,

When Boeing wanted to design the 747, they had two choices:

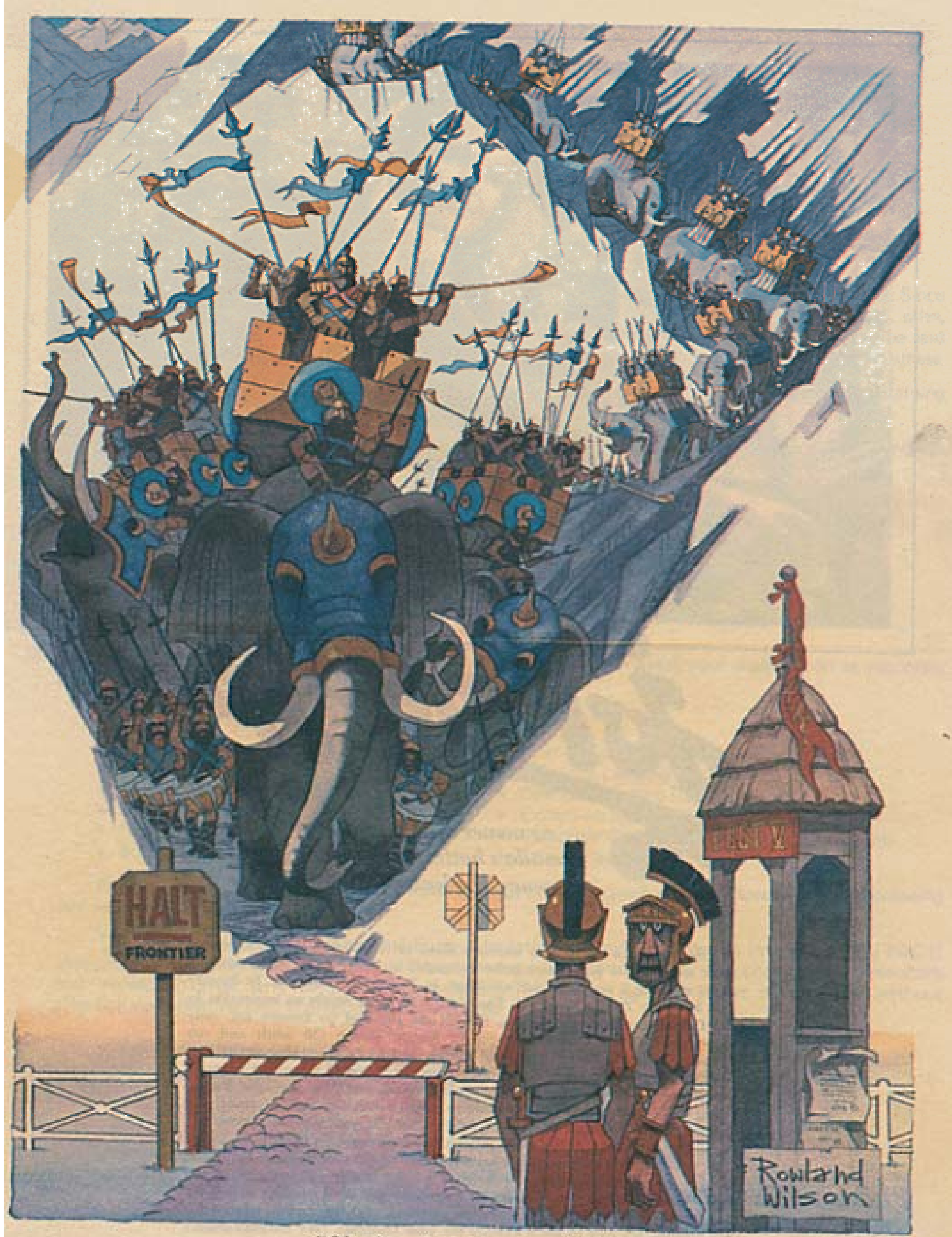
1. Hire “SuperEngineer”, who could do it by alone
2. Hire 7,200 engineers and organize them to cooperate

Which did they choose?

Why?

What can we learn from this?

But the problems just keep on coming….



"Oh, oh . . . here comes trouble!"

101

Design

Specify the details of inter-object collaboration *mechanisms*

- Determine the *structure* of classes and their associations

  Relationships of access, ownership, authority

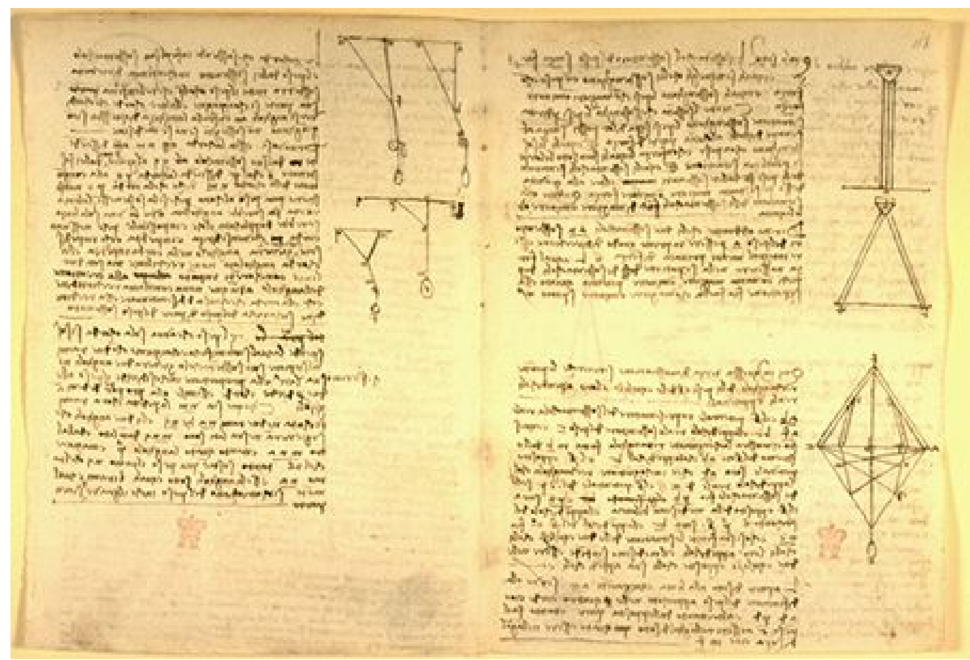
- Determine the *behavior* of classes

  E.g. Interactions with other objects

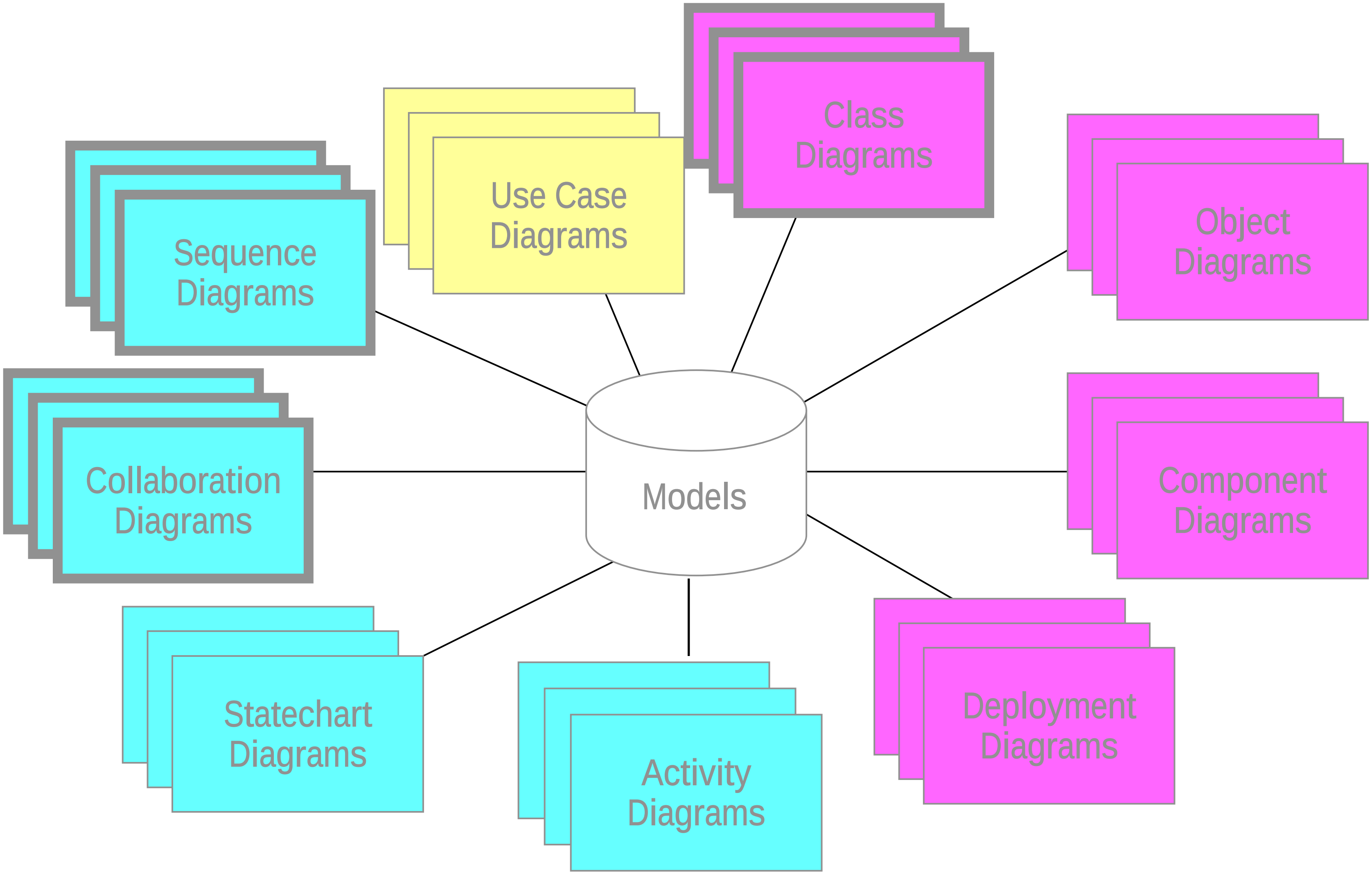
    Collaboration

    Sequence

How do we record and communicate this?

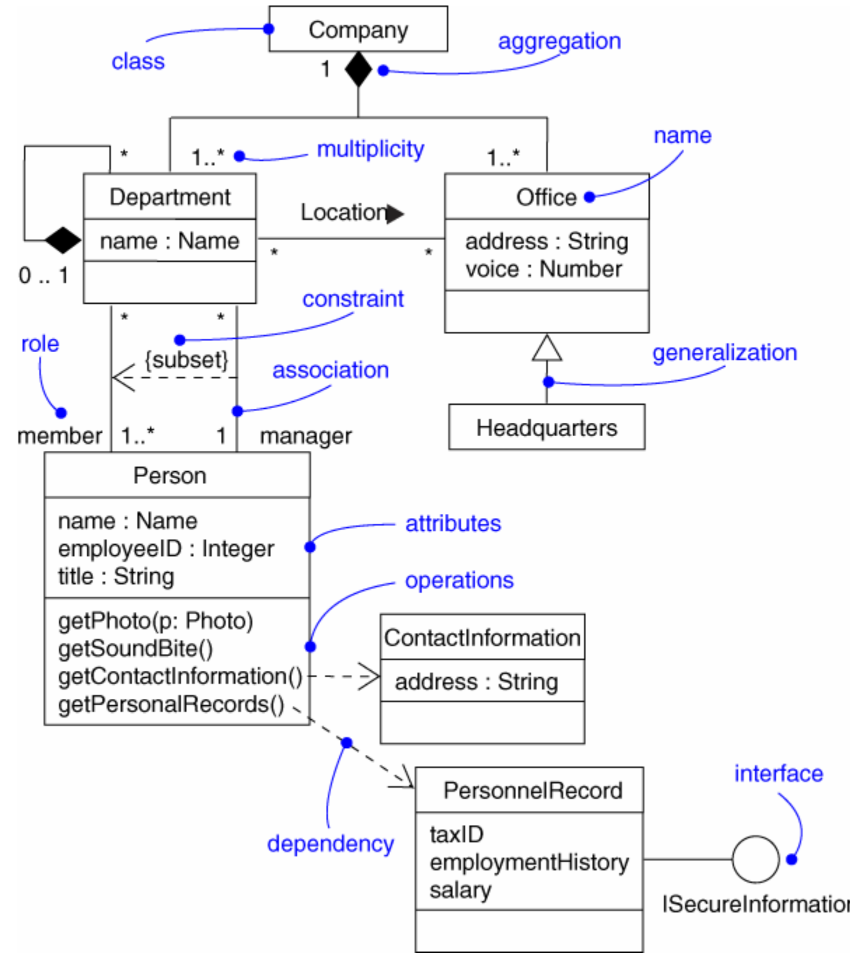


UML Diagrams



Class Diagram

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

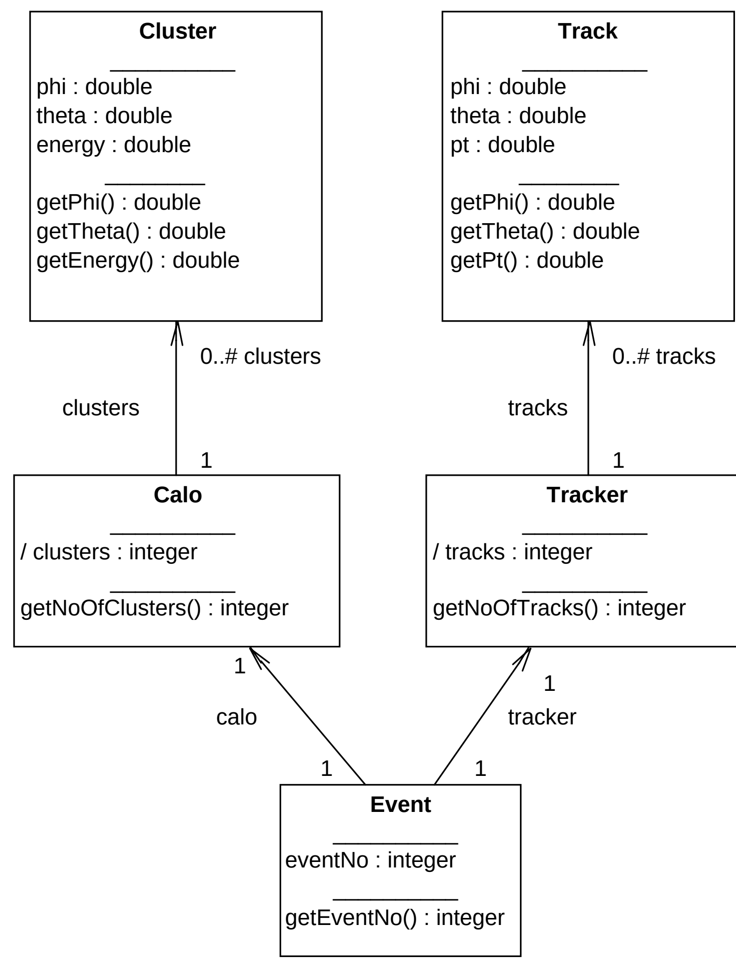


Rational Software Corporation

Example Class Diagrams

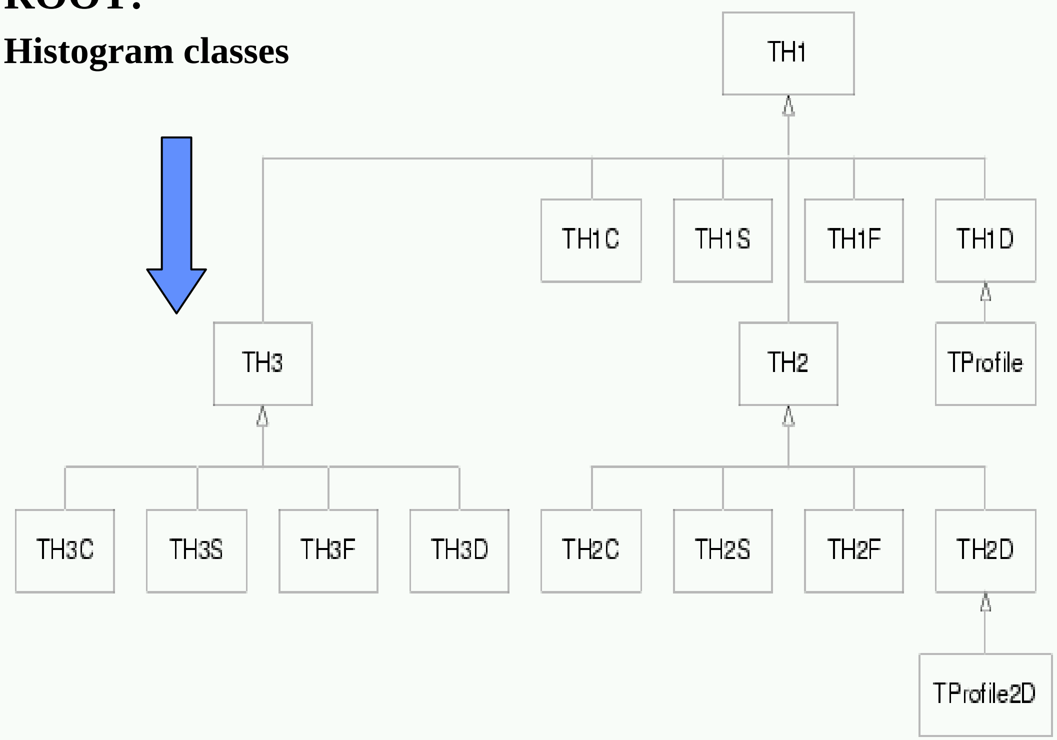
LHC++/Anaphe:

Event structure as defined in DDL file for populateDb exercise



ROOT:

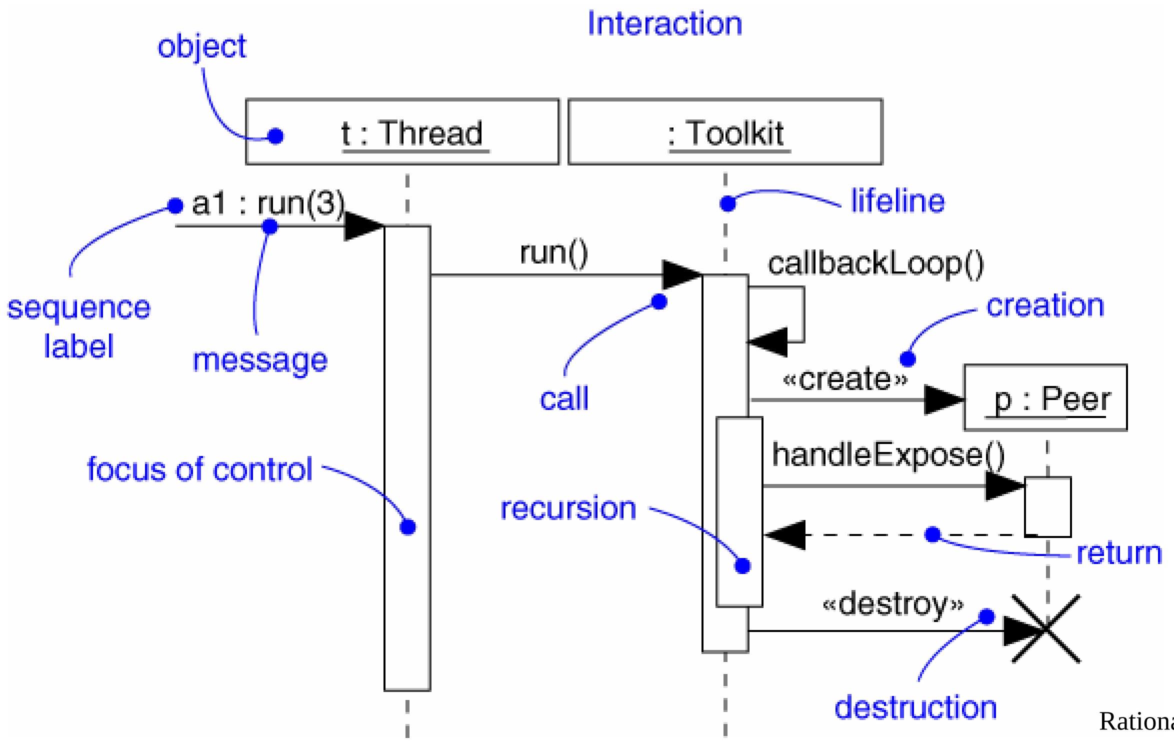
Histogram classes



Sequence Diagram

Captures dynamic behavior (time-oriented)

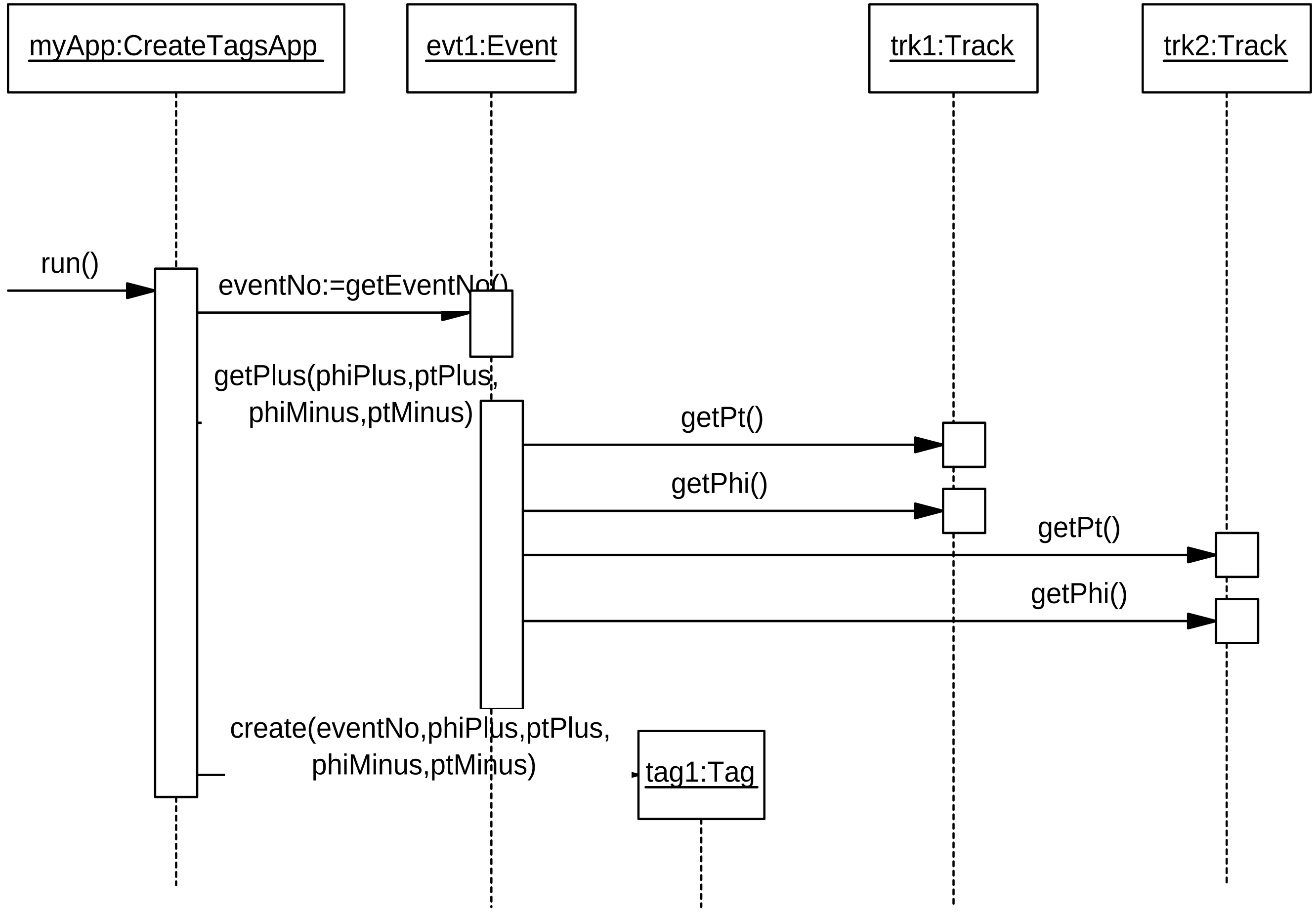
- Model flow of control
- Illustrate typical scenarios



Rational Software Corporation

Example Sequence diagram

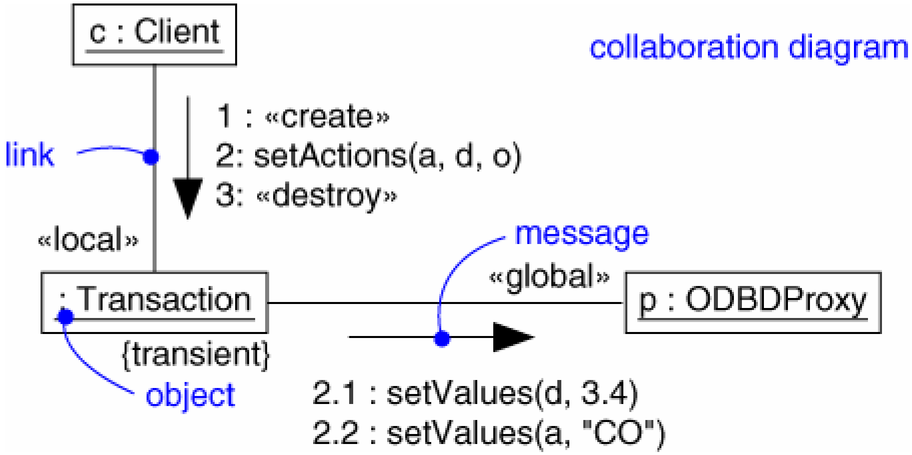
LHC++/Anaphe: scenario for createTag exercise with 1 event and 2 tracks



Collaboration Diagram

Captures dynamic behavior (message-oriented)

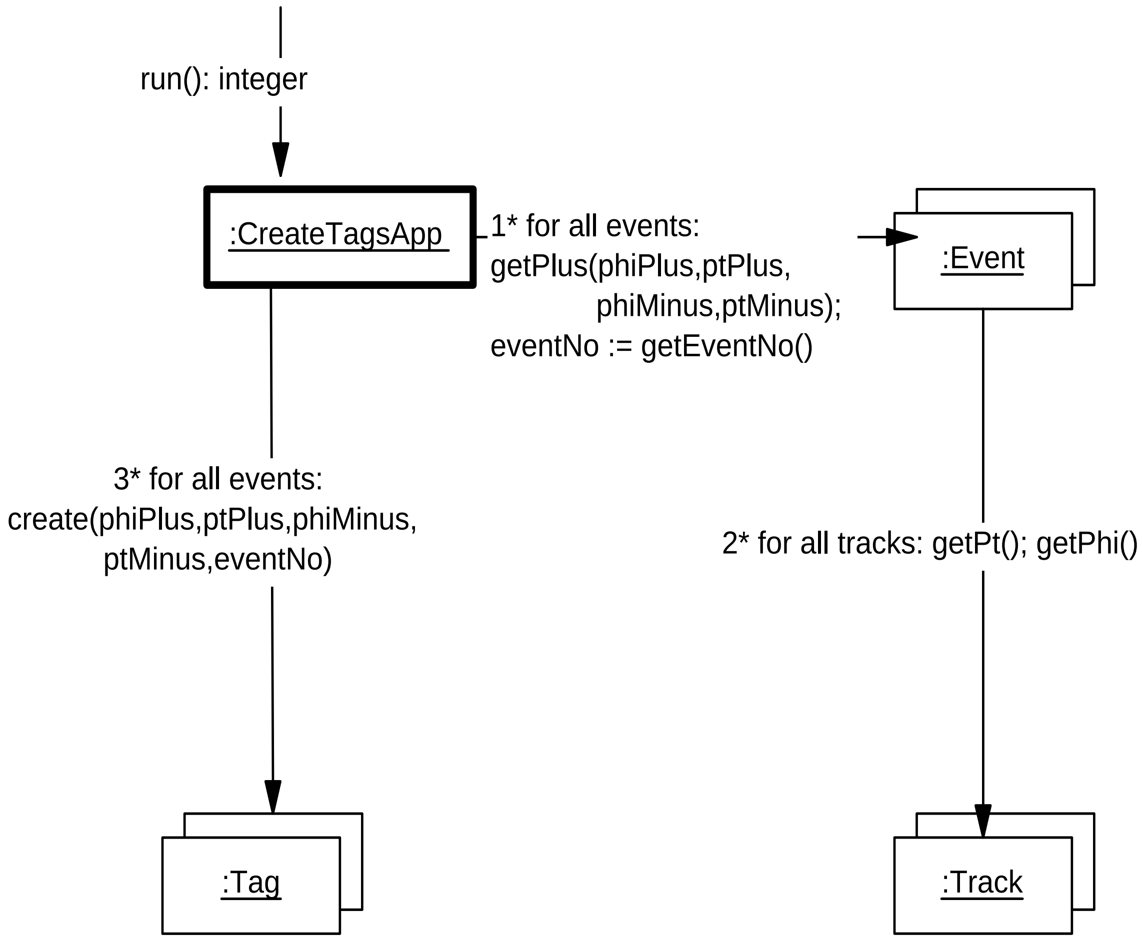
- Model flow of control
- Illustrate coordination of object structure and control



Rational Software Corporation

Example Collaboration Diagram

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



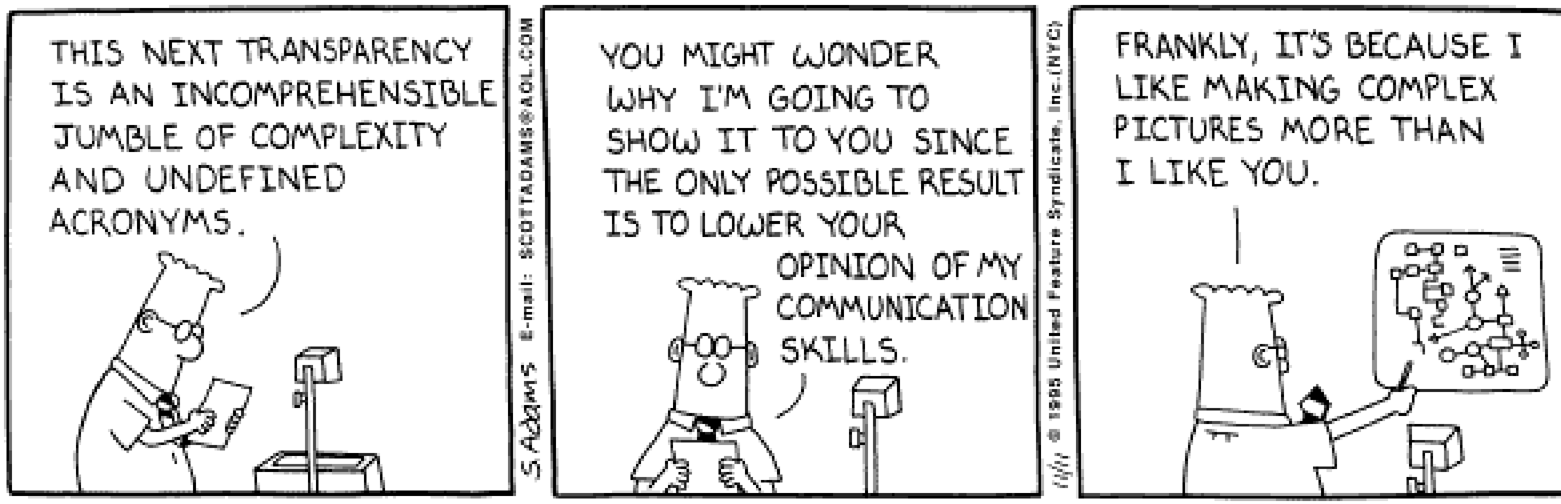
“These are complicated”

“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



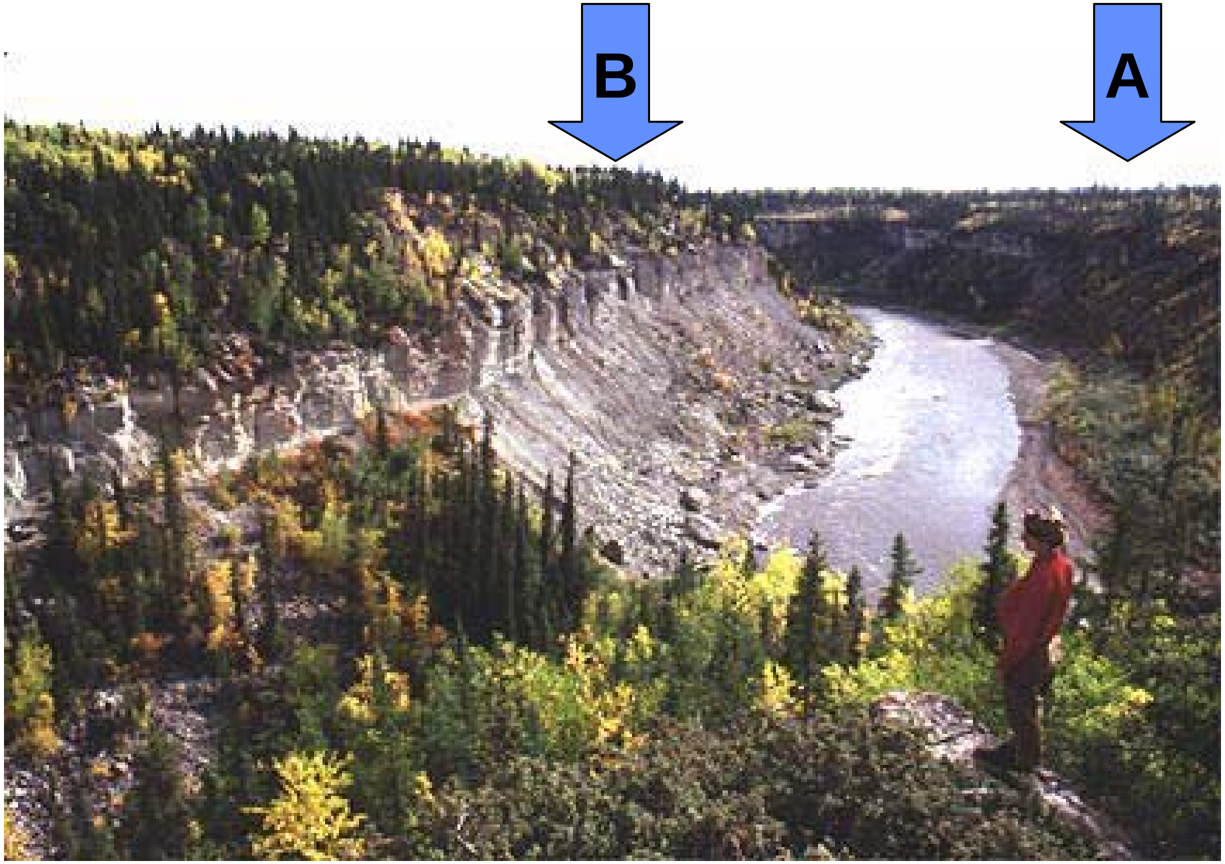
“Yes, and how do we know they’re right?”

- That’s the key question.

This is where iterative development comes in…

Imagine the project is not to build software but a bridge…

Initial Requirements: A to B



Iteration I

Meets primary requirement: A to B

Basic architecture is in place

Single user version

Can only be used in winter

Not very safe

Iteration II

New requirements:

- Works in the summer
- Multi-user

Same basic architecture but different technology

Multi-user version!

Can be used all year round

Iteration III

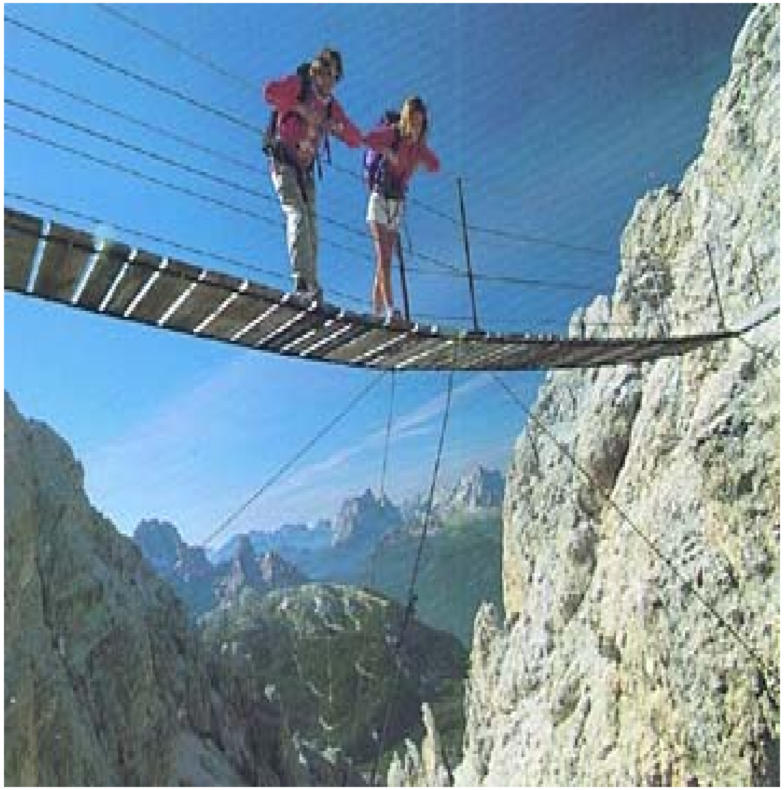
New requirements

- more stable and safe

Same architecture and technology

More solid construction

Extra security



Iteration V

New requirements

- Protected from the rain
- Two-way

Same architecture with improved technology

Protected from environment (at least from above)

Bi-directional

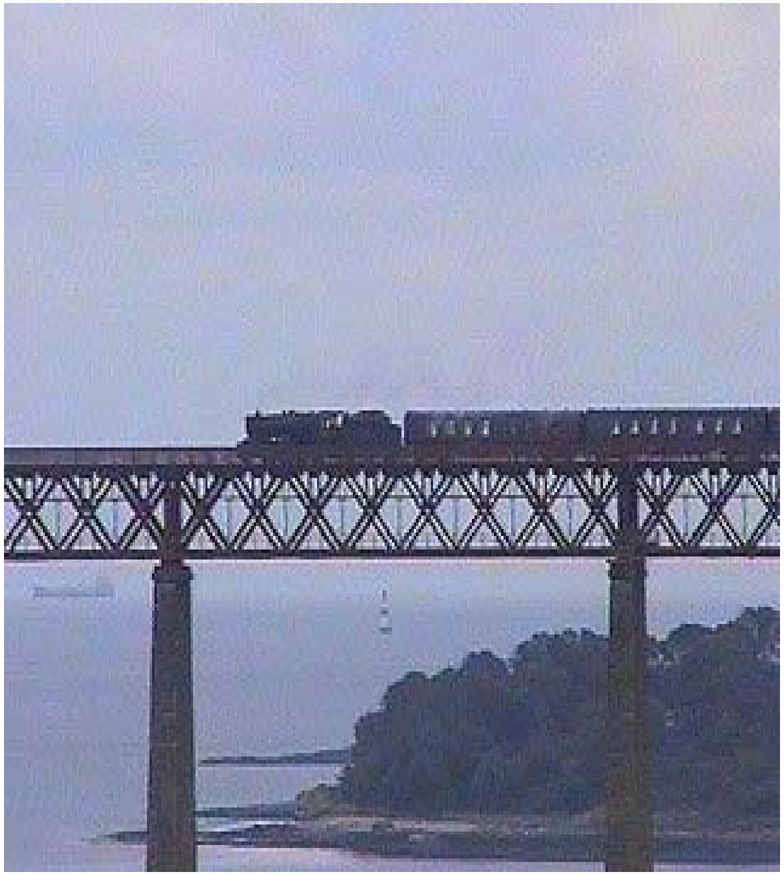
Iteration VI

New requirements:

- *"I want to move house to B"*

Same basic architecture but advanced technology

Can carry other goods



Iteration VII

New requirements:

- *"I want to be able to use my car and let ships go by"*

Multi-purpose

Successful Development Program!

Analogy shows successful iterations:

- The basic *product* existed from the first iteration and met the primary requirement: A to B
- Early emphasis on defining the architecture
- Basic architecture remained the same over iterations
- Extra functionality/reliability/robustness was added at each iteration
- Each iteration required more analysis, design, implementation and testing
- Use case (requirements) driven

  does what the users want - not what the developers think is cool

Some limits to analogy:

It took people centuries to figure out how to build big bridges

And we developed engineering processes to do the big ones!

Little of the early cycles survived in final one

How to pick what goes in the next iteration?

Choice of additions for an iteration is *risk driven*

- Early development focuses on components with the highest risk and uncertainty

    Avoids investing resources in a project that is not feasible

- But it has to do something basically useful

    So all involved will take it seriously

Bad outcomes I

Does not go from A to B

Went for "full functionality" from the start

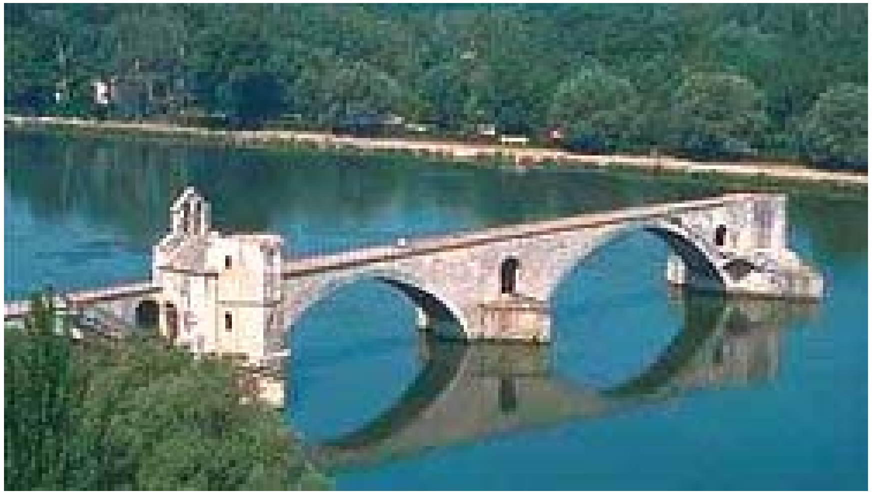
- Big bang approach
- Face too much complexity at the start

Users/sponsors got cold feet?

- Ran out of resources, patience
  or enthusiasm

Requirements have long since changed

- no feedback from users since never used



Sounds like the traditional "one-pass" approach?

Bad outcome II

Does not go from A to B any more
Insufficient testing?
Unstable environment?
Lack of routine maintenance?
Too many concurrent users?

Went straight to the code?

Legacy systems

Still goes from A to B

Been in use for a long time

Difficult to determine the original architecture

The original development team are no longer around

No documentation

Lots of inconsistencies resulting from later additions made with insufficient analysis and design

Advantages of Iterative and Incremental Development

Complexity is never overwhelming

Only tackle small bits at a time

Avoid *analysis paralysis* and *design decline*

Early feedback from users

Provides input to the analysis of subsequent iterations

Developers skills can *grow* with the project

Don't need to apply latest techniques/technology at the start

Get used to delivering finished software

Requirements can be modified

Each iteration is a mini-project (analysis, design….)

Note that these benefits come from completing, deploying and using the iterations!

Detailed design

Important step just before coding

- maps to code in the chosen programming language

Determine the structure of an object's information and it's manipulation

- data structures (attributes)
- implementation of associations
- sets of operations defined for the data (methods)
- visibility of data and operations

  (C++: private, protected, public)
- Error handling techniques (e.g. exceptions thrown)

Associations

Implementation depends on nature and locality

objects in the same thread, different processes or machines

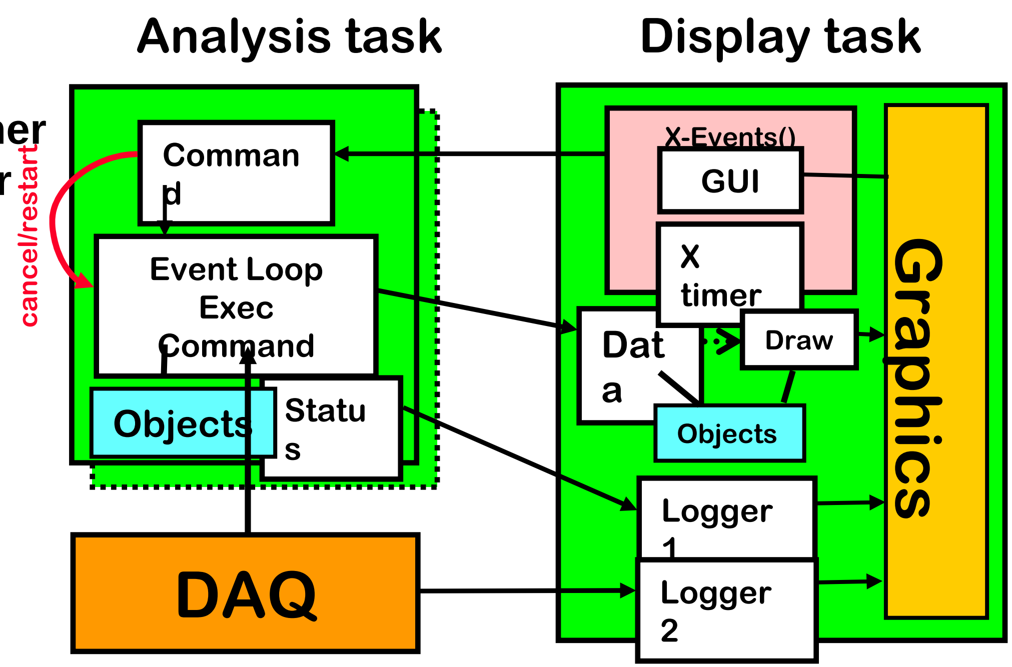
persistent or transient

cardinality of association

Examples

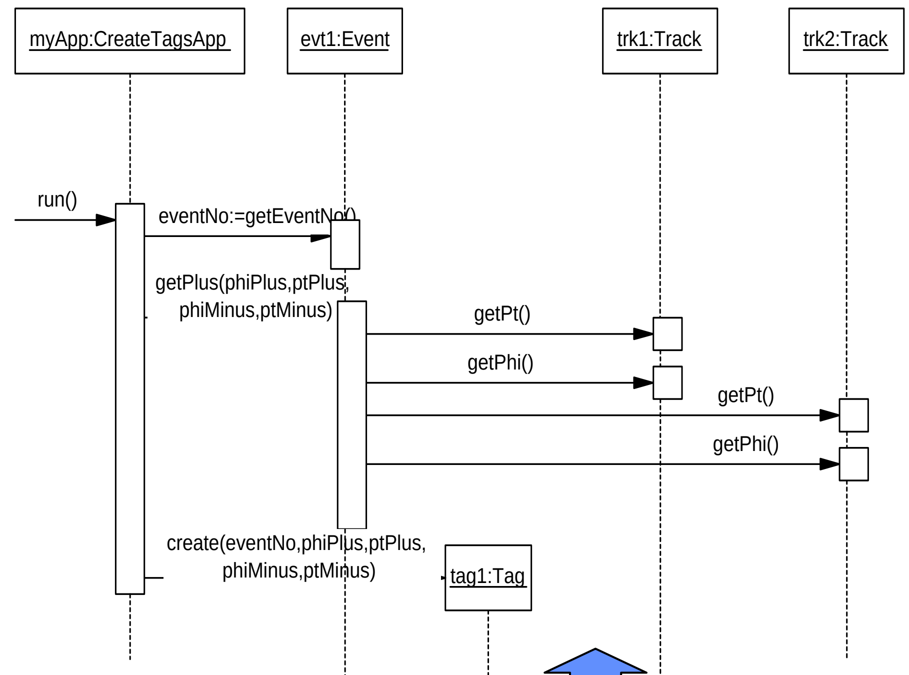
```
class testAssoc {
 T                t1; // 1-to-1 only
 T*               t2; // 1-to-1 & 1-to-(0,1)
 list<T>          t3; // 0-to-n STL container
 TList *tracks;   t4; // 0-to-n ROOT container
 d_vector<T>      t5; // 0-to-n Objy container
};
```

Mix of tasks, threads and comms.

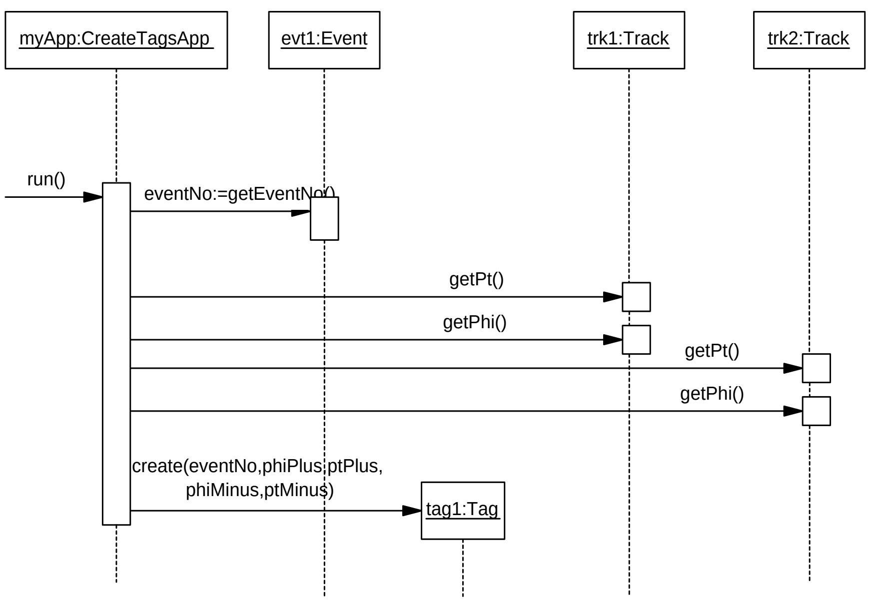


Operations

Mapping of methods from design to code may change according to code ownership, dynamics and practicality



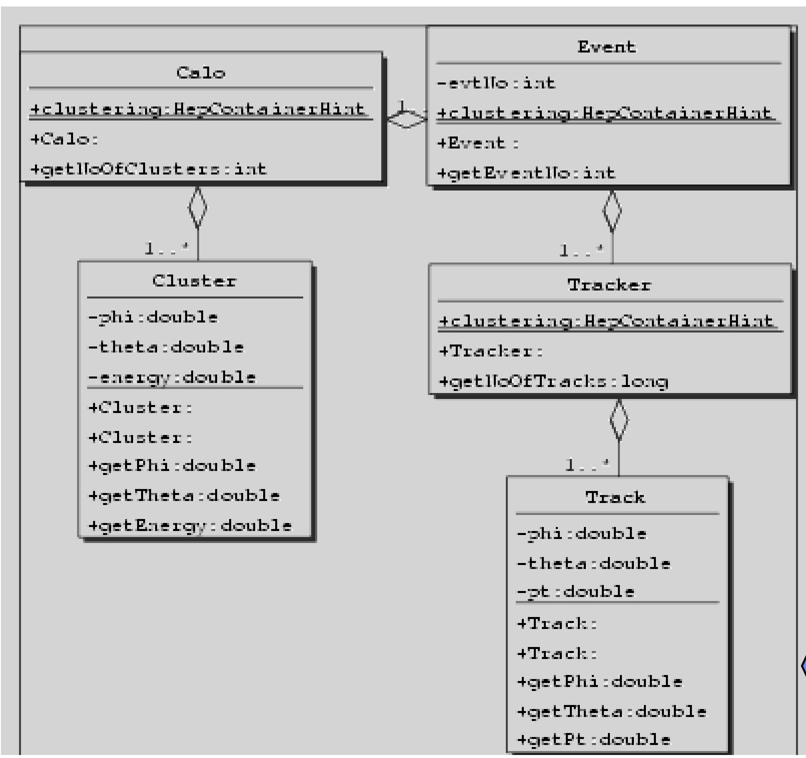
design

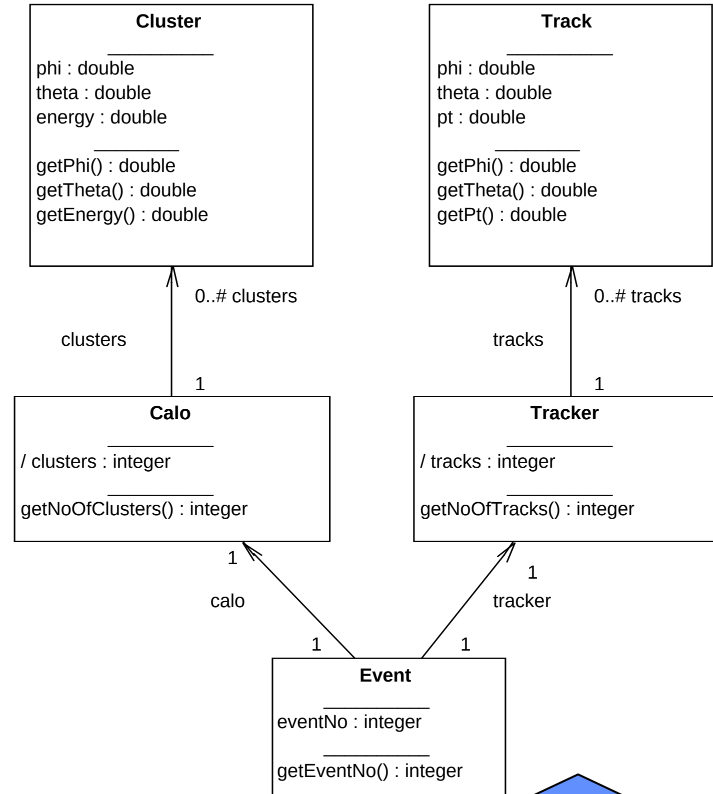


implementation

Class definitions

Difference between design class diagram and implementation





implementation

design

Reverse engineered from LHC++/Anaphe Event.ddl using Together/C++ CASE tool

Lecture summary

Software engineering is the art of building complex computer systems

It's ideas and techniques spring from our need to handle size & complexity

As you do your own work & develop your own skills, consider:

- How your effort effects or contributes to things 10X, 100X, 1000X larger
- How you'll do things different/better when it's your problem

Exercise 8 is way to consider some of these ideas in context

- Adding some minor functionality to an <u>existing</u> system

Today's Exercises

6) Demonstration of profiling tools
7) Practice tuning a small application

8) Project: Add a new feature to an existing program

Instruction sheets are available via web browser at

file:/home/jake/index.html