The Engineering Process

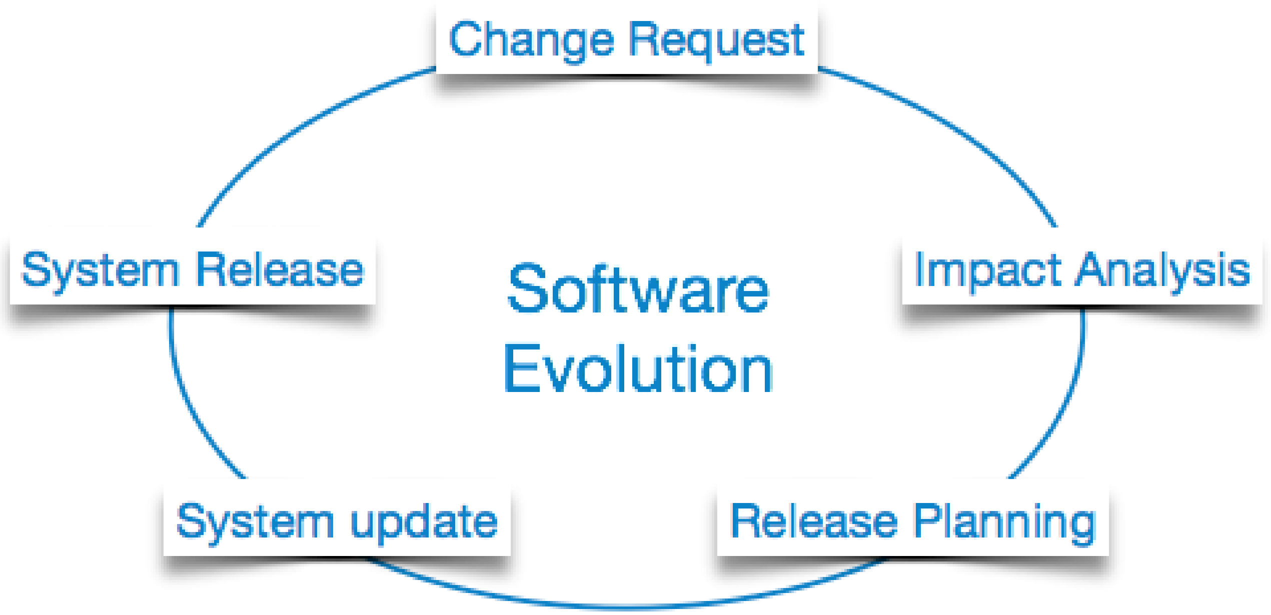
Software Development Process
Unified Process
Round-Trip Engineering
Reverse Engineering
Examples

Software Engineering

- The seminal definition (by Fritz Bauer):
  - Software Engineering (SE) is “the establishment and use of *sound engineering principles* in order to obtain economically software that is *reliable and works efficiently on real machines”*.
- The IEEE definition:
  - Software Engineering (SE) is: (1) The application of a *systematic, disciplined, quantifiable approach* to the *development, operation, and maintenance* of software; that is, the application of engineering to software; (2) The study of approaches as in (1).

Software evolution

- The process of developing a software product using software engineering principles and methods is referred to as **software evolution**. This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.



Critics to SE

Most software development is a chaotic activity, often characterized by the phrase ‘code and fix.’

The software is written without much of an underlying plan, and the design of the system is cobbled together from many short term decisions.

This works pretty well if the system is small, but as the system grows it becomes increasingly difficult to add new features to the system.

Furthermore, bugs have become increasingly prevalent and increasingly difficult to fix.

A typical sign of such a system is a long test phase after the system is ‘feature complete.’

Martin Fowler, Chief Scientist of ThoughtWorks (2005)

Solution: the software process

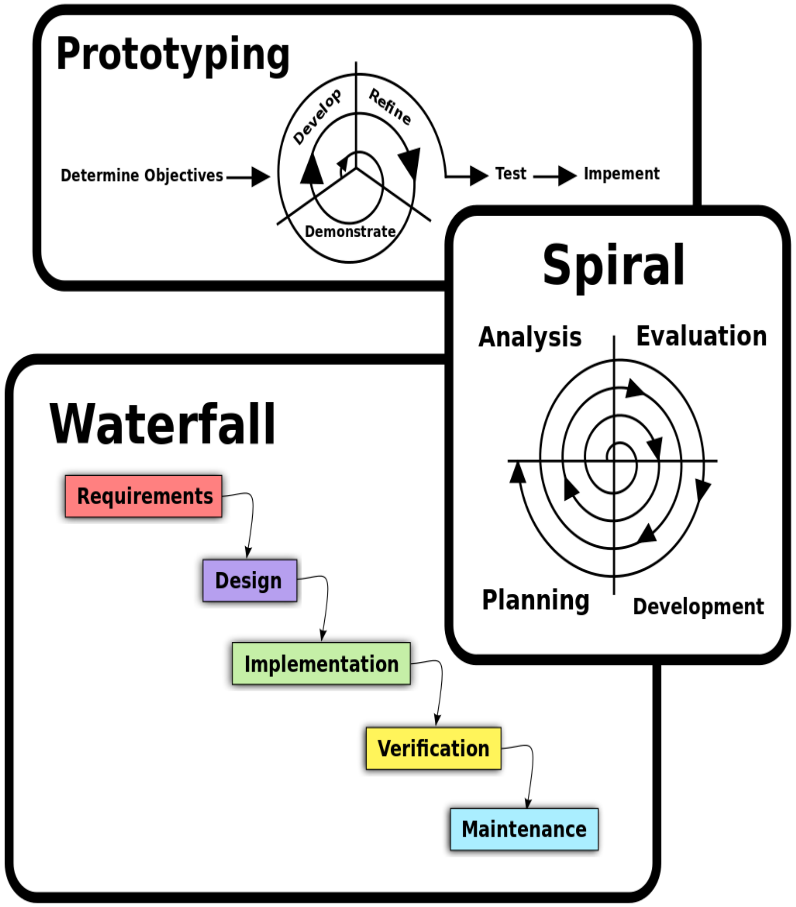
- A process is a collection of activities, actions and tasks that are performed when some work product is to be created.
- There is **not a rigid prescription** for how to build computer software. Rather, it is an adaptable approach that enables the people doing the work to pick and choose the **appropriate** **set of work actions** and tasks.
- Purpose of process is to deliver software in a timely manner and with sufficient quality to satisfy those who have sponsored its creation and those who will use it.

What?/Who?/Why? in process models?

- **What**: a *road map* with series of predictable steps, for a timely, high-quality results.
- **Who**: Software engineers, their managers, and clients – adapt the process to their needs and follow it.
- **Why**: for stability, control, and organization to activities, but modern software engineering approaches must be agile – with such activities, controls and work products that are appropriate.
- **What**:
  - work products such as programs, documents, and data
  - the steps of the process you adopt depends on the software that you we are building.
- **How**: software process assessment mechanisms enable us to determine the maturity of the software process – here, quality, timeliness and long-term viability of the software are the best indicators of the process efficacy.

Definition of software process

- A **framework** for the activities, actions, and tasks that are required to build high-quality software.
- SP defines the approach containing related activities that leads to the production of the software.
- Is not equal to software engineering, which also encompasses **technologies** that populate the process – technical methods and automated tools.



Software development process

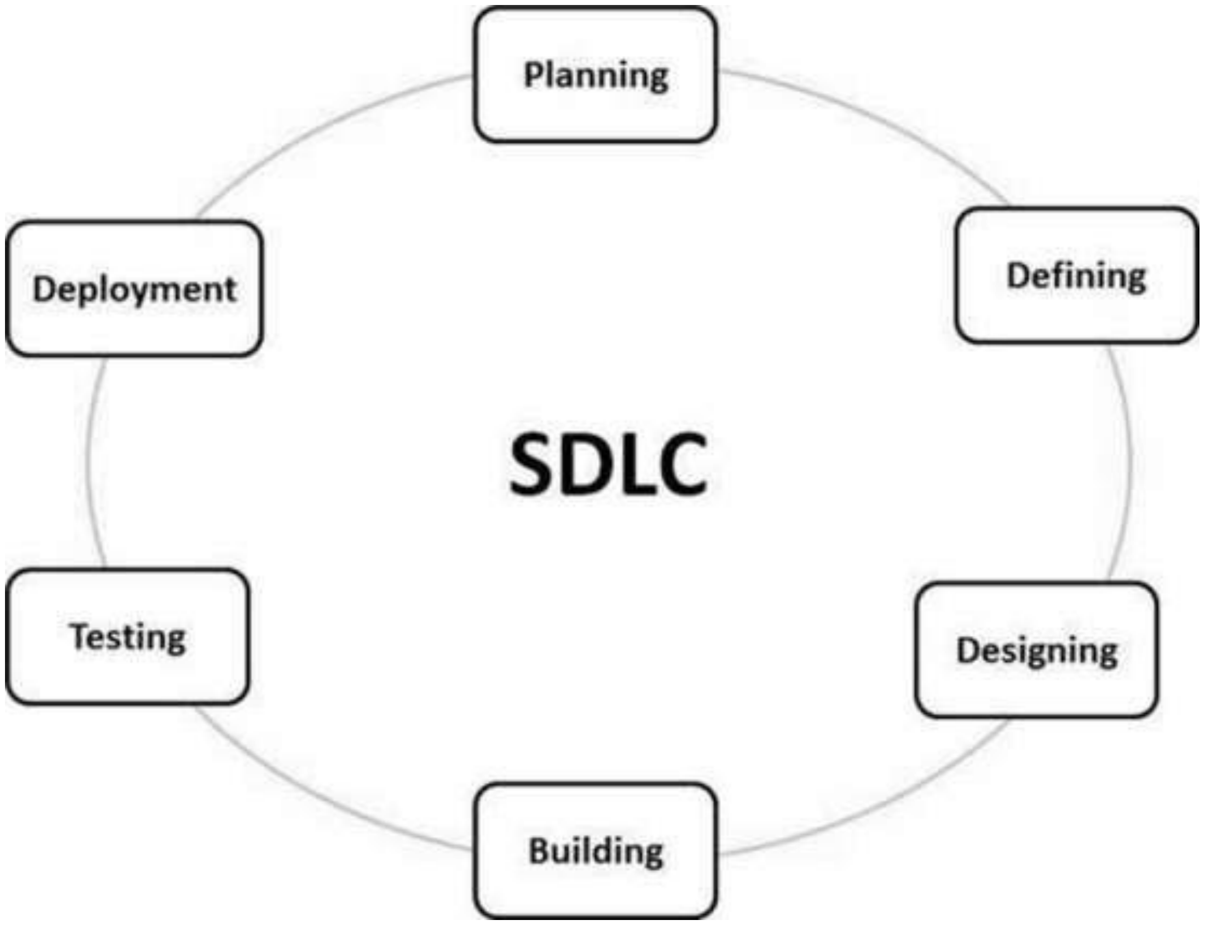
- The software development process is a general term describing the over-arching process of developing a software product
- Companies often select a development process that fits their personnel and resources. Several development process methods are in existence, and a growing body of software development standardizing organizations are currently implementing and rating different methods.

Source: https://www.techopedia.com/definition/13295/software-development-process

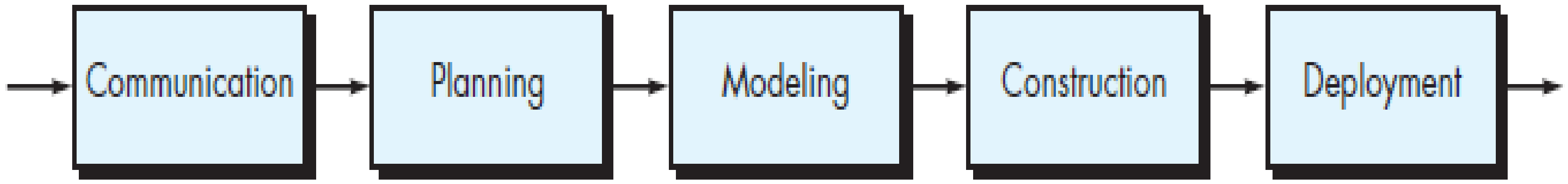
Software Development Life Cycle

- Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software.
- The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.
- SDLC is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software. *https://www.tutorialspoint.com/sdlc/sdlc_overview.htm*

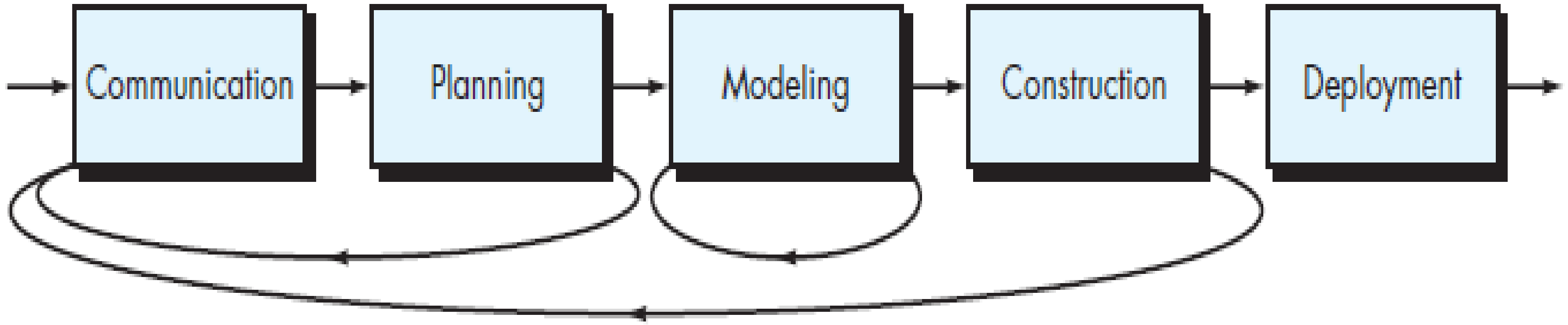
Stages of a typical SDLC



Types of process flows 1/2

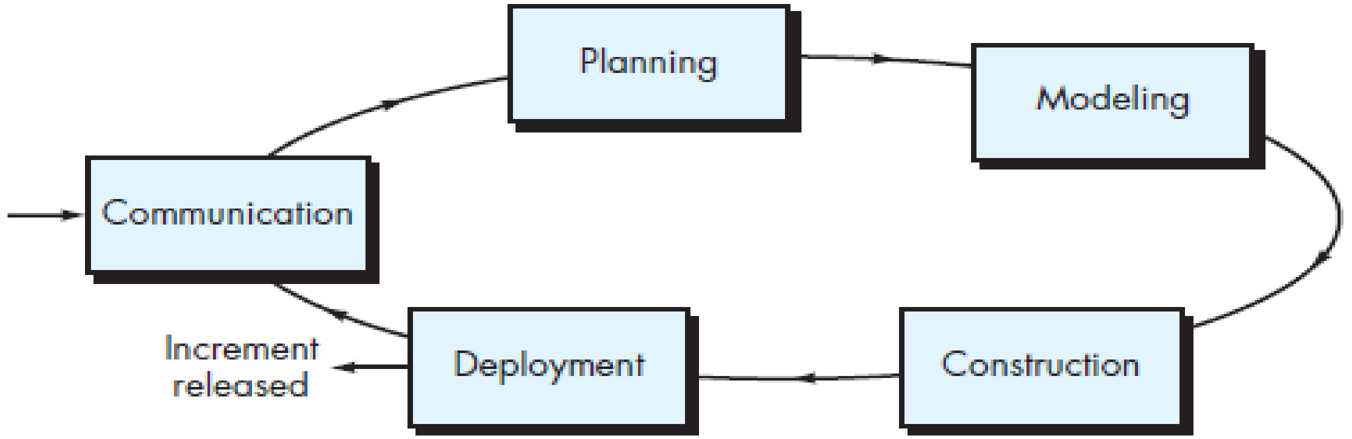


(a) Linear process flow

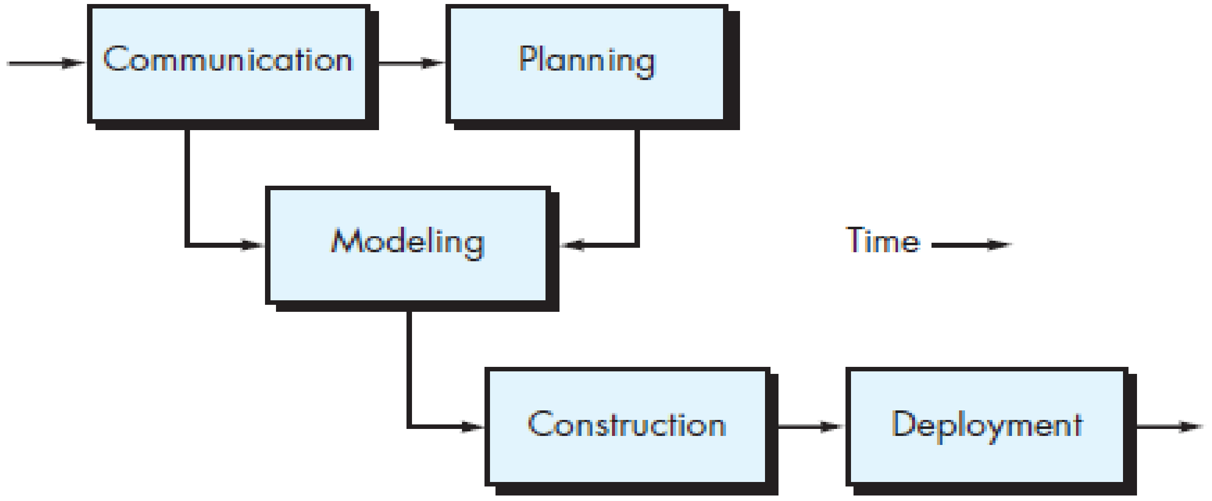


(b) Iterative process flow

Types of process flows 2/2



(c) Evolutionary process flow



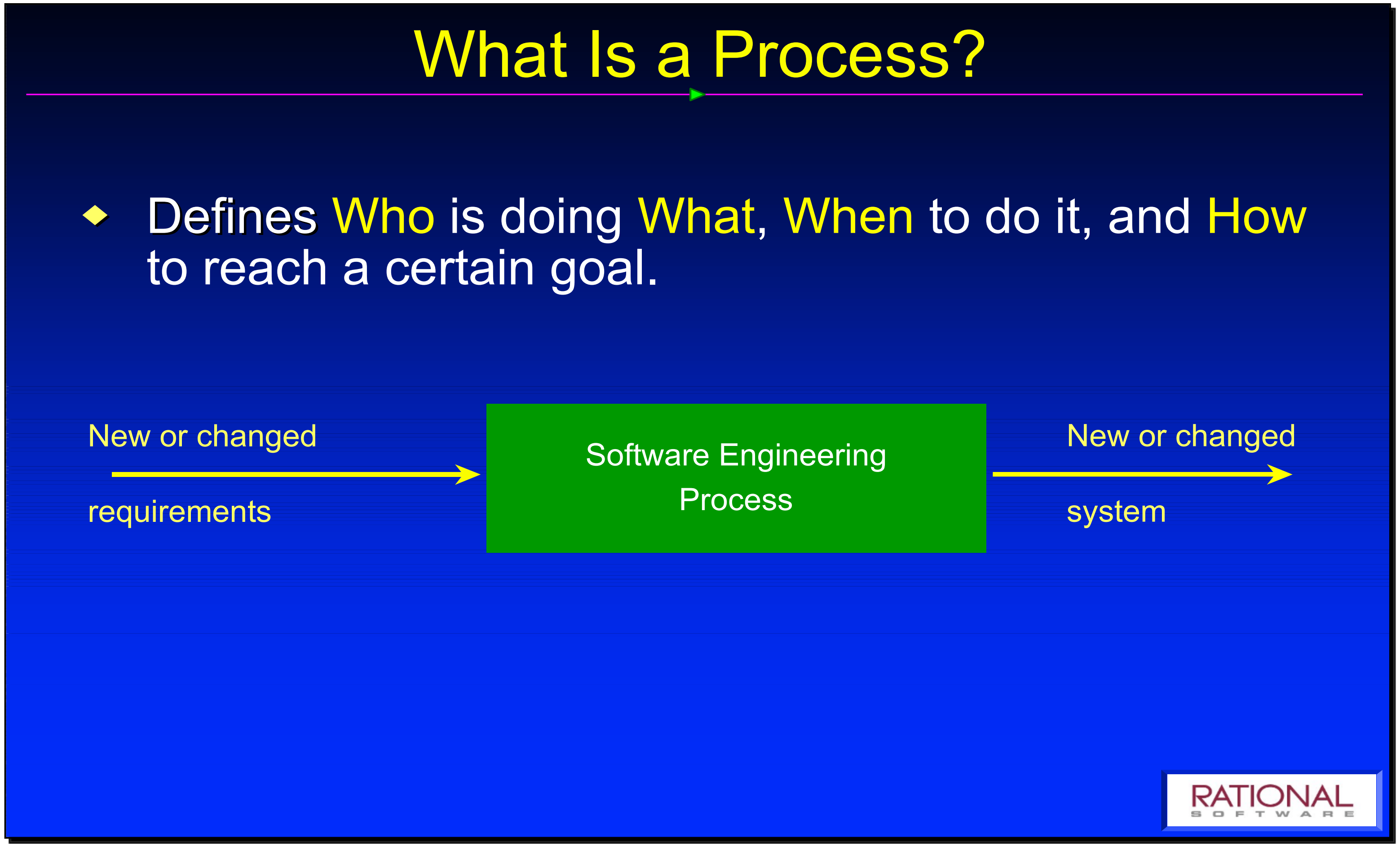
(d) Parallel process flow

The software development process

- The **software development process** is the process of dividing software development work into distinct phases to improve design, product management, and project management.
- Known as well as a **software development life cycle**.
- The methodology may include the pre-definition of **specific deliverables and artifacts** that are created and completed by a project team to develop or maintain an application.

Source: Suryanarayana, Girish (2015). "Software Process versus Design Quality: Tug of War?". *IEEE Software*. **32** (4): 7–11.

The SW Dev. Process Itself



What Is a Process?

- Defines Who is doing What, When to do it, and How to reach a certain goal.

New or changed
requirements
Software Engineering Process
New or changed
system
RATIONAL SOFTWARE

Phases of Software Development – Analysis

- **Requirements analysis** - specifying the functional capabilities needed in the software. ***Use-cases*** are an important tool for communication about requirements between software developers and their clients.
  Products: software requirements documents for the software
  Objectives: capture the client's needs and wants
- **Domain analysis** - developing concepts, terminology, and relationships essential to the client's model of the software and its behavior. ***Conceptual-level class diagrams and interaction diagrams*** are important tools of domain analysis.
  Products: client-oriented model for the software and its components
  Objectives: capture the client's knowledge framework

Source: G. Shute, UMN

Phases of Software Development – Design

- **Client-oriented design** - specifying components of the software that are visible to the client and its' behavior in terms of their attributes, methods, and relationships to other components. ***Specification-level class diagrams and interaction diagrams*** are important tools here.
  Products: client-oriented specifications for components
  Objectives: define the structure of interactions with the client, providing methods that satisfy the client's needs and wants, operating within the client's knowledge framework
- **Implementation-oriented design** - determining internal features and method algorithms for the software. Usage of ***component diagrams.***
  Products: implementation-oriented specifications for components
  Objectives: define internal structure and algorithms for components that meet client-oriented specifications

Source: G. Shute, UMN

Phases of Software Development – Implementation and Integration

- **Implementation** - writing and compiling code for the individual software components.
  Products: source/binary code for software components and their test software
  Objectives: to produce coded components that accurately implement the implementation-oriented design
- **Integration** - putting the software components into a context with each other and with client software. Usage of software integration tools (https://code-maze.com/top-8-continuous-integration-tools/)
  Products: integrated software components
  Objectives: test the integrated software components in the context in which they will be used

Source: G. Shute, UMN

Phases of Software Development – Packaging

- **Packaging** - bundling the software and its documentation into a deliverable form. Usage of package diagrams.
  Products: software and documentation in an easily installed form
  Objectives: to manage the software in an efficient way

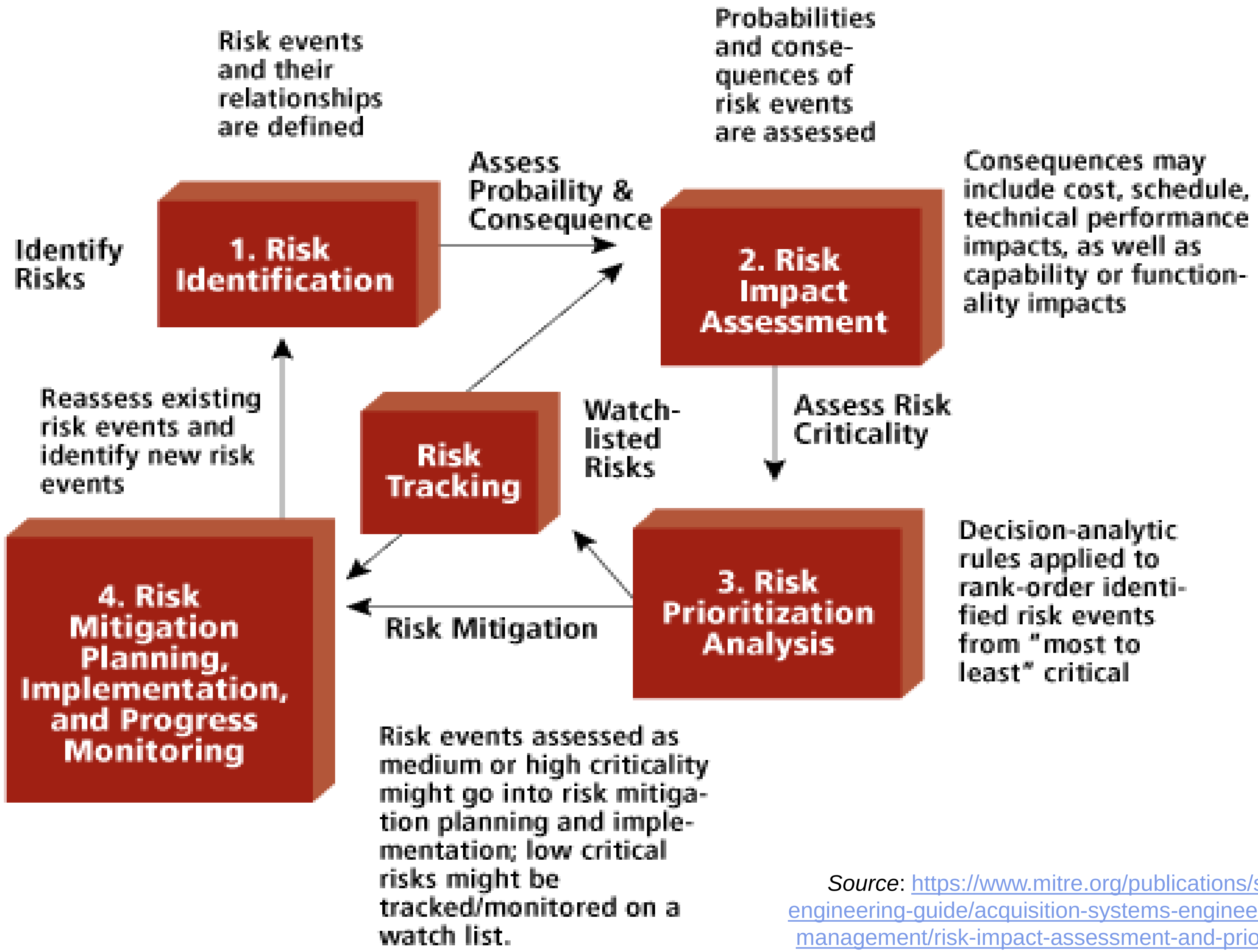
Source: G. Shute, UMN

Ongoing Activities in Software Development 1/2

- **Risk analysis** - management activities that attempt to identify aspects of the development process that have a significant chance of failing.

- **Planning** - management activities that determine the specific goals and allocate adequate resources for the various phases of development. Resources include time, work and meeting space, people, and developmental hardware and software. Risk analysis can be viewed as preparation for planning.

Source: G. Shute, UMN


Risk events and their relationships are defined
Probabilities and consequences of risk events are assessed
Assess Probaility & Consequence
Consequences may include cost, schedule, technical performance impacts, as well as capability or functionality impacts
Identify Risks
1. Risk Identification
2. Risk Impact Assessment
Reassess existing risk events and identify new risk events
Risk Tracking
Watch-listed Risks
Assess Risk Criticality
Decision-analytic rules applied to rank-order identified risk events from "most to least" critical
4. Risk Mitigation Planning, Implementation, and Progress Monitoring
Risk Mitigation
3. Risk Prioritization Analysis
Risk events assessed as medium or high criticality might go into risk mitigation planning and implementation; low critical risks might be tracked/monitored on a watch list.

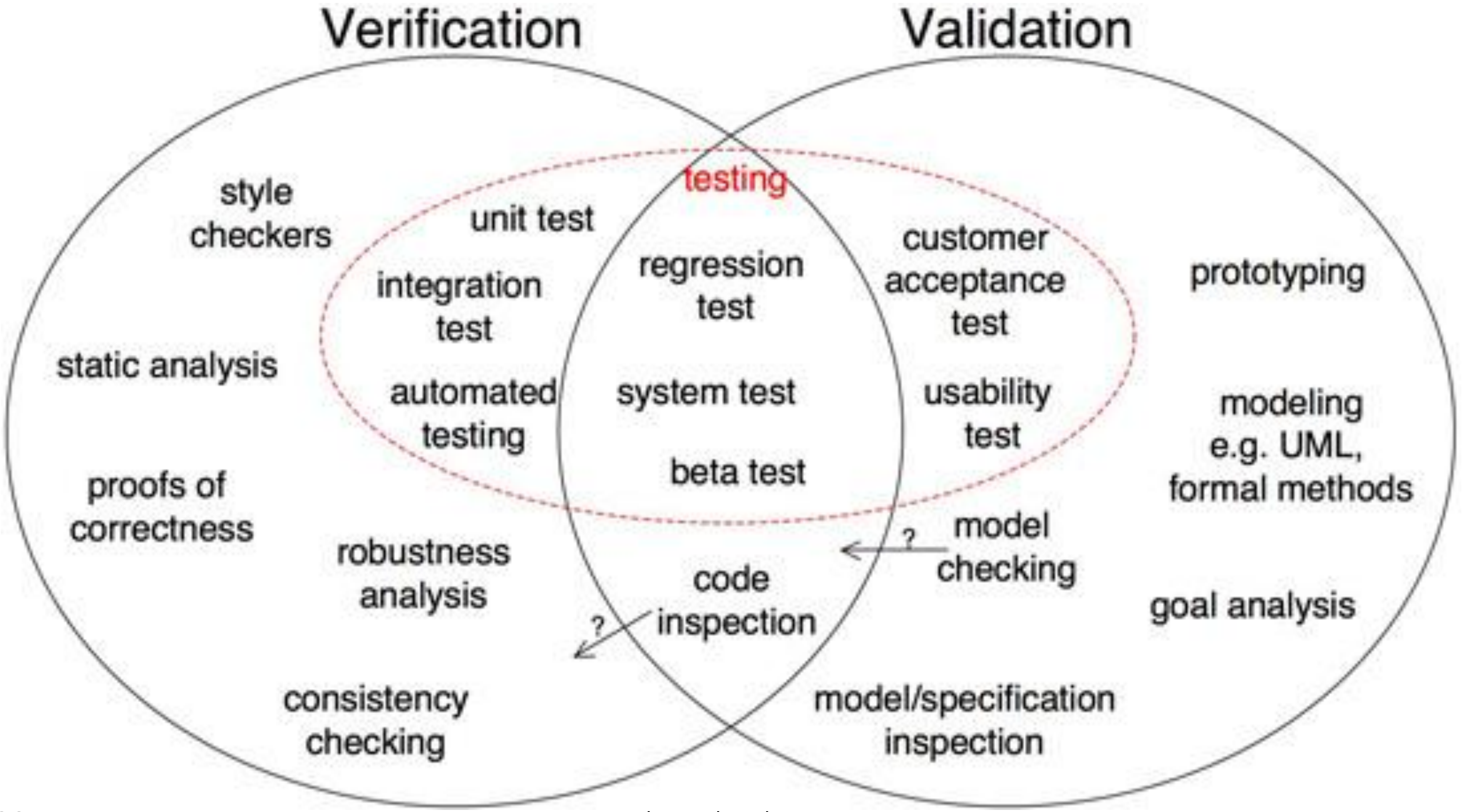
Source: https://www.mitre.org/publications/systems-engineering-guide/acquisition-systems-engineering/risk-management/risk-impact-assessment-and-prioritization

Ongoing Activities in Software Development 2/2

- **Software verification** – to assure that *software fully satisfies all the expected requirements*, with activities directed at ensuring that the products of the various phases of development meet their objectives. Testing is an important part of verification that takes place during implementation and integration. Two kinds of testing:
  - Black-box testing is testing how software meets its client-oriented specifications, without regard to implementation.
  - White-box testing uses knowledge of implementation to determine a testing plan that all paths of control have been exercised.
- **Software validation** – the process of *checking whether the specification captures the customer's needs*
- **Documentation** - providing instructions and information needed for *the installation, use, and maintenance of software*.

Source: G. Shute, UMN

Software Verification vs Validation



Source: https://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/

Some software development methodologies

1990s

- Rapid application development (RAD), since 1991
- Dynamic systems development method (DSDM), since 1994
- Scrum, since 1995
- Team software process, since 1998
- Rational Unified Process (RUP), maintained by IBM since 1998
- eXtreme Programming (XP), since 1999

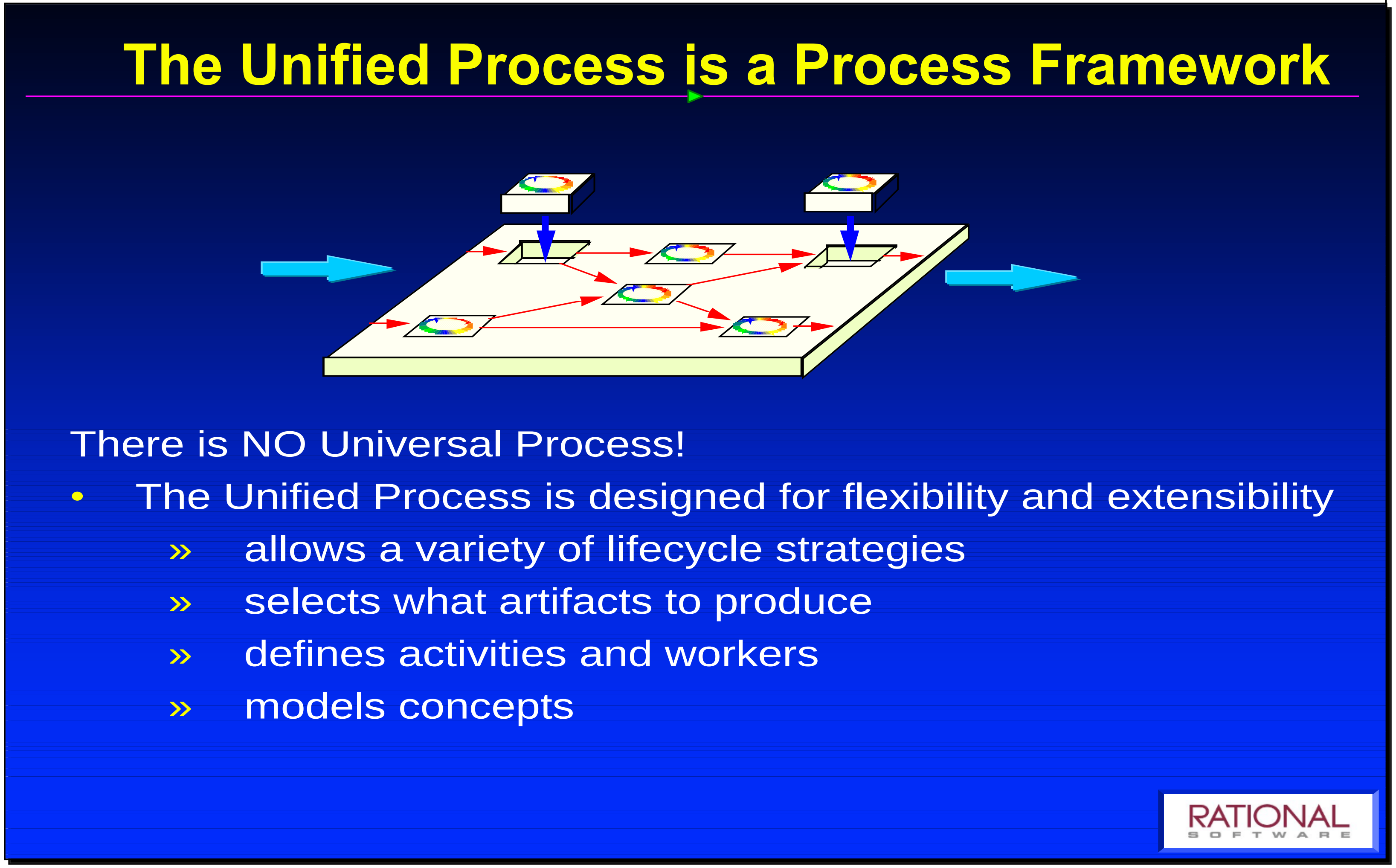
2000s

- Agile Unified Process (AUP) maintained since 2005 by Scott Ambler
- Disciplined Agile Delivery (DAD) supersedes AUP

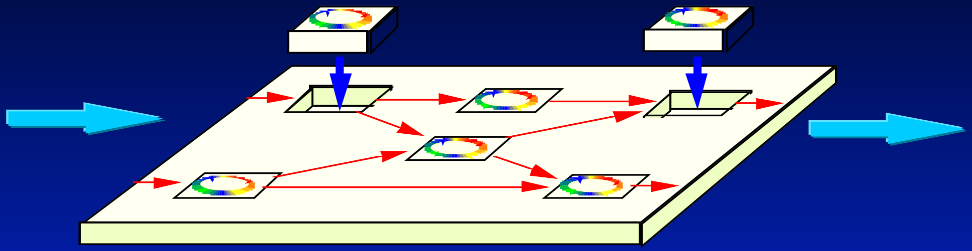
2010s

- Scaled Agile Framework (SAFe)
- Large-Scale Scrum (LeSS)

The Rational/IBM Unified Process



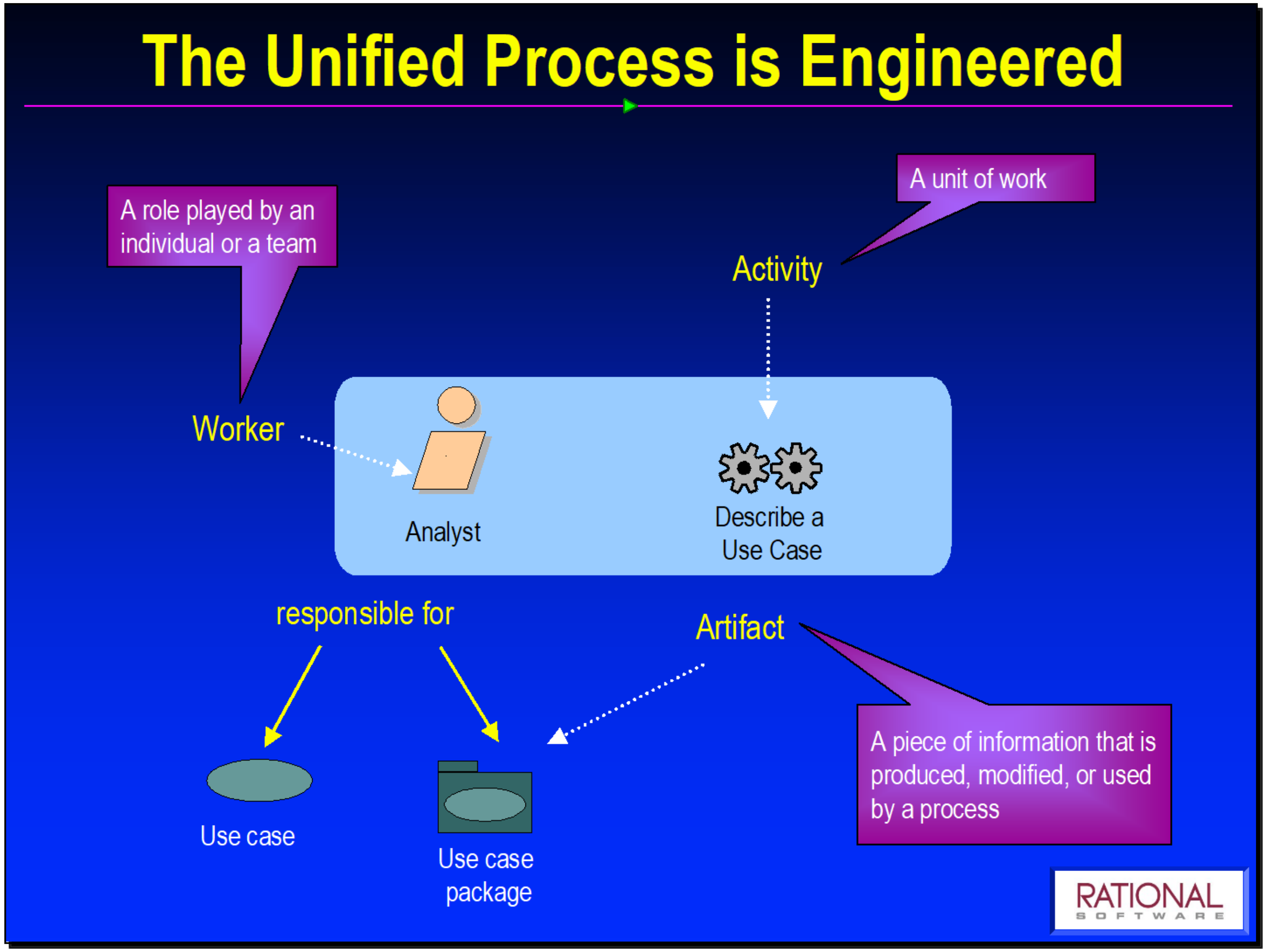
The Unified Process is a Process Framework



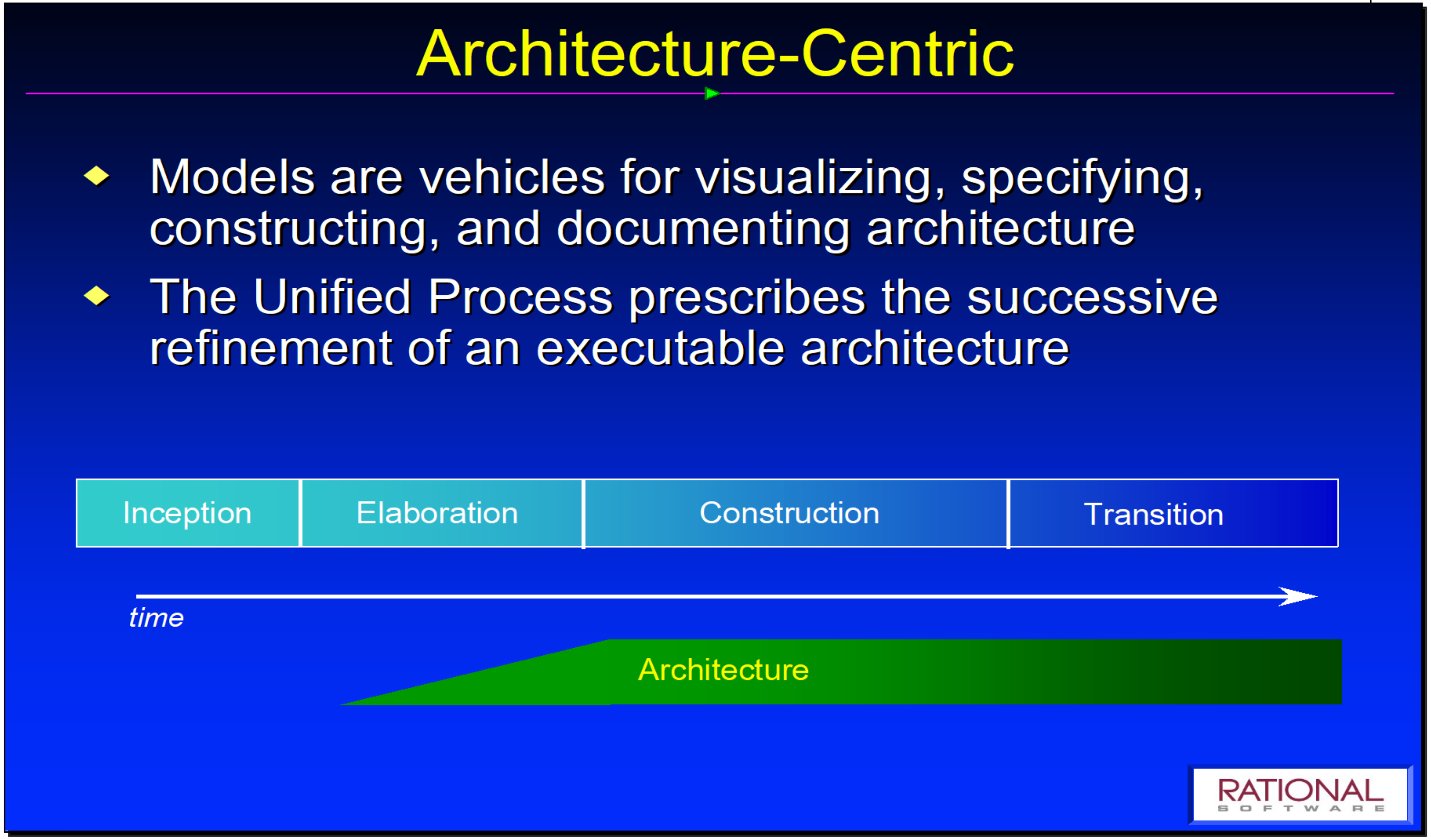
There is NO Universal Process!

- The Unified Process is designed for flexibility and extensibility
  - » allows a variety of lifecycle strategies
  - » selects what artifacts to produce
  - » defines activities and workers
  - » models concepts

The Unified Process for SW Engineering


The Unified Process is Engineered
A role played by an individual or a team
A unit of work
Activity
Worker
Analyst
Describe a Use Case
responsible for
Artifact
Use case
Use case package
A piece of information that is produced, modified, or used by a process
RATIONAL SOFTWARE

The Unified Process is Architecture-Centric

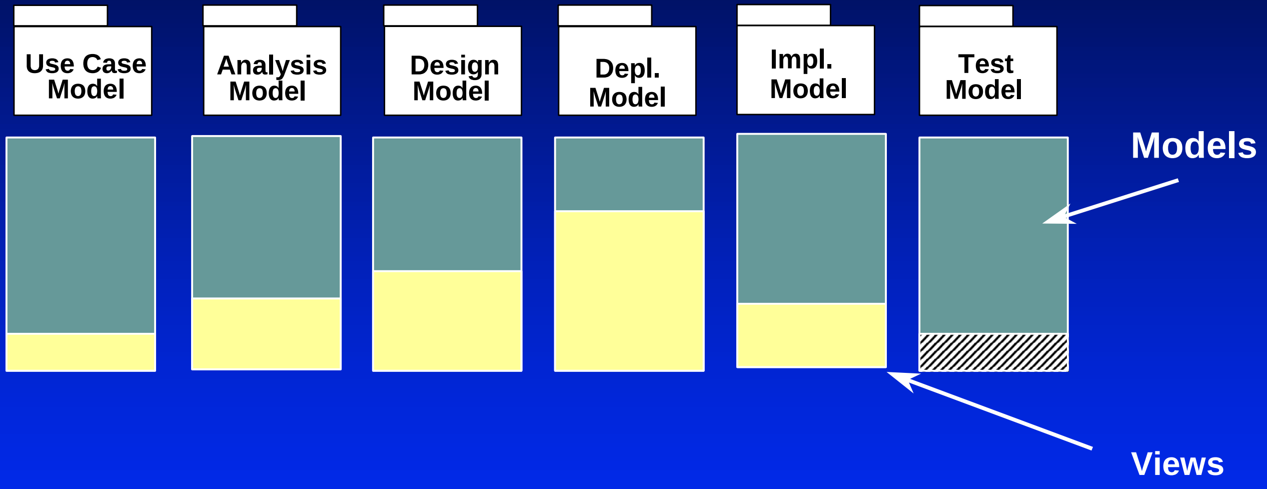


Architecture-Centric

- Models are vehicles for visualizing, specifying, constructing, and documenting architecture
- The Unified Process prescribes the successive refinement of an executable architecture

Inception
Elaboration
Construction
Transition
time
Architecture
RATIONAL SOFTWARE

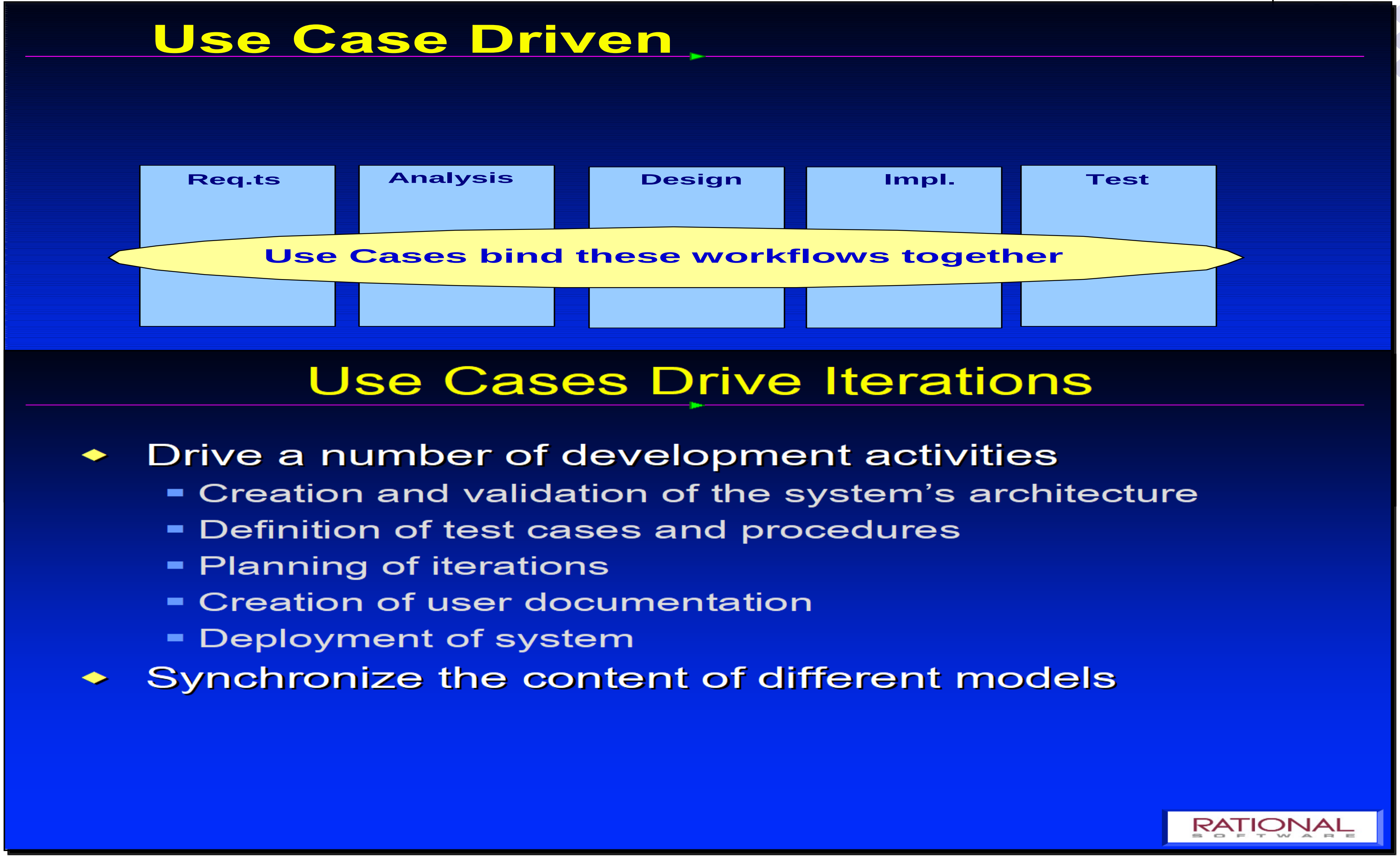
Architecture and Models



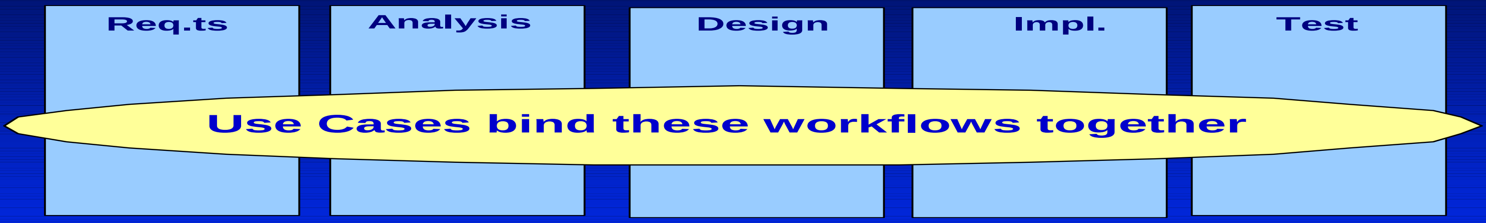
Architecture embodies a collection of views of the models

RATIONAL SOFTWARE

The Unified Process is Use-Case Driven



Use Case Driven

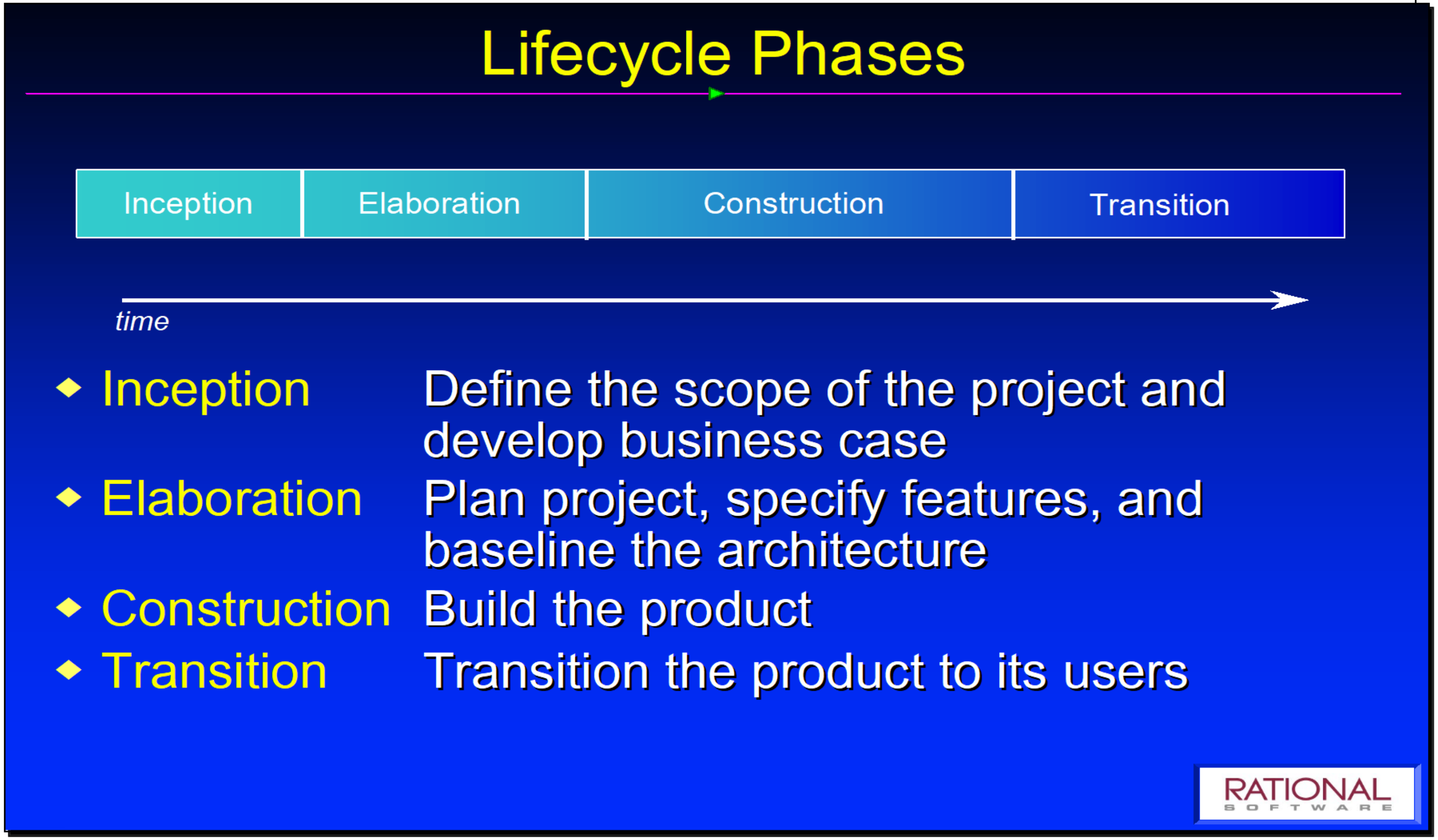


Use Cases Drive Iterations

- Drive a number of development activities
  - Creation and validation of the system's architecture
  - Definition of test cases and procedures
  - Planning of iterations
  - Creation of user documentation
  - Deployment of system
- Synchronize the content of different models

RATIONAL SOFTWARE

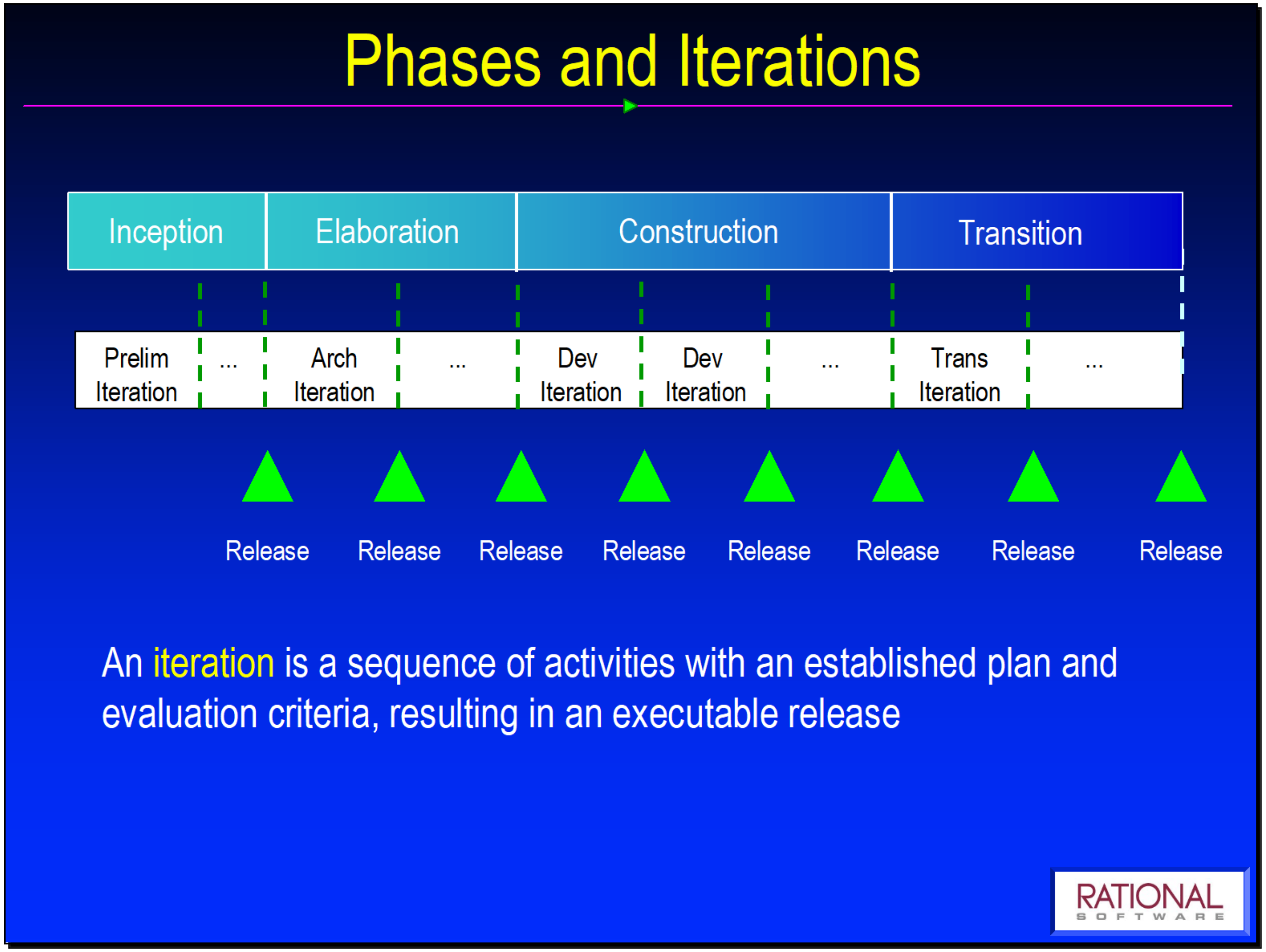
The Unified Process is Iterative and Incremental



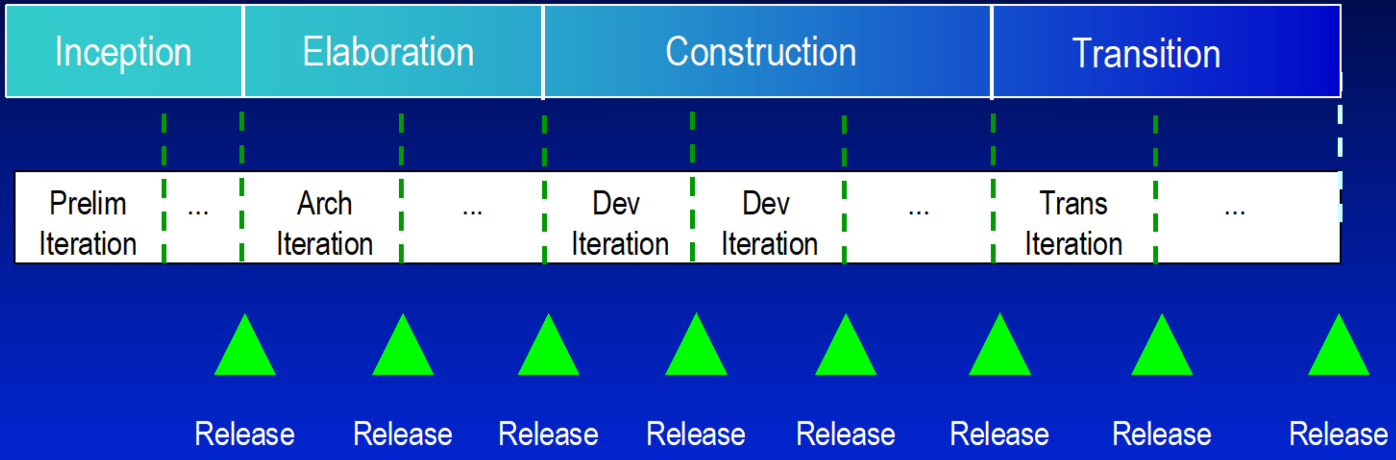
Lifecycle Phases

Inception
Elaboration
Construction
Transition
time
RATIONAL SOFTWARE

Inception	Define the scope of the project and develop business case
Elaboration	Plan project, specify features, and baseline the architecture
Construction	Build the product
Transition	Transition the product to its users



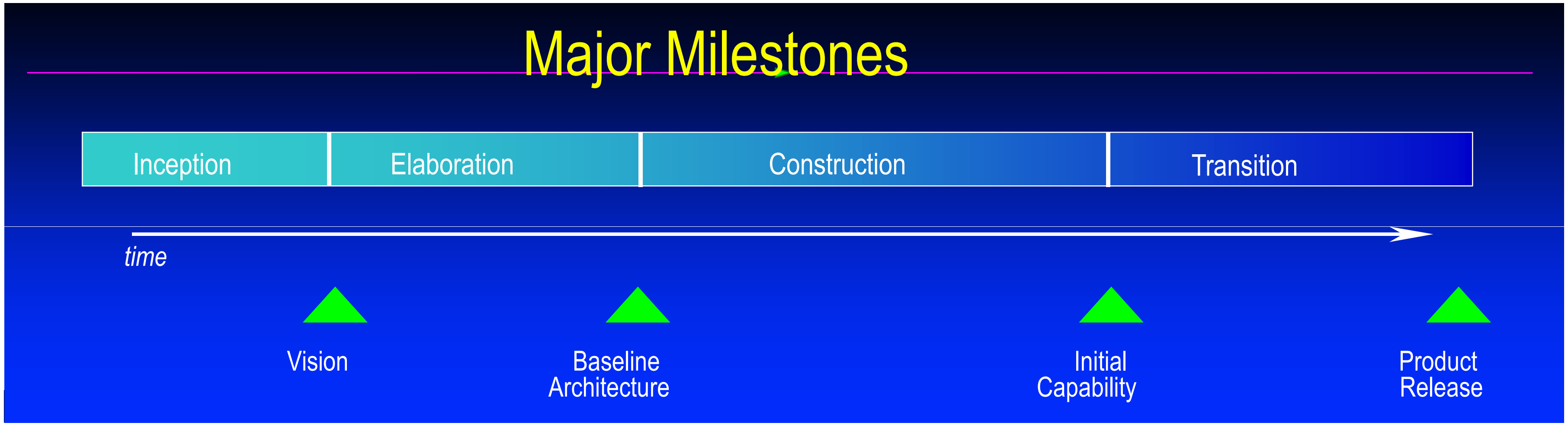
Phases and Iterations



An iteration is a sequence of activities with an established plan and evaluation criteria, resulting in an executable release

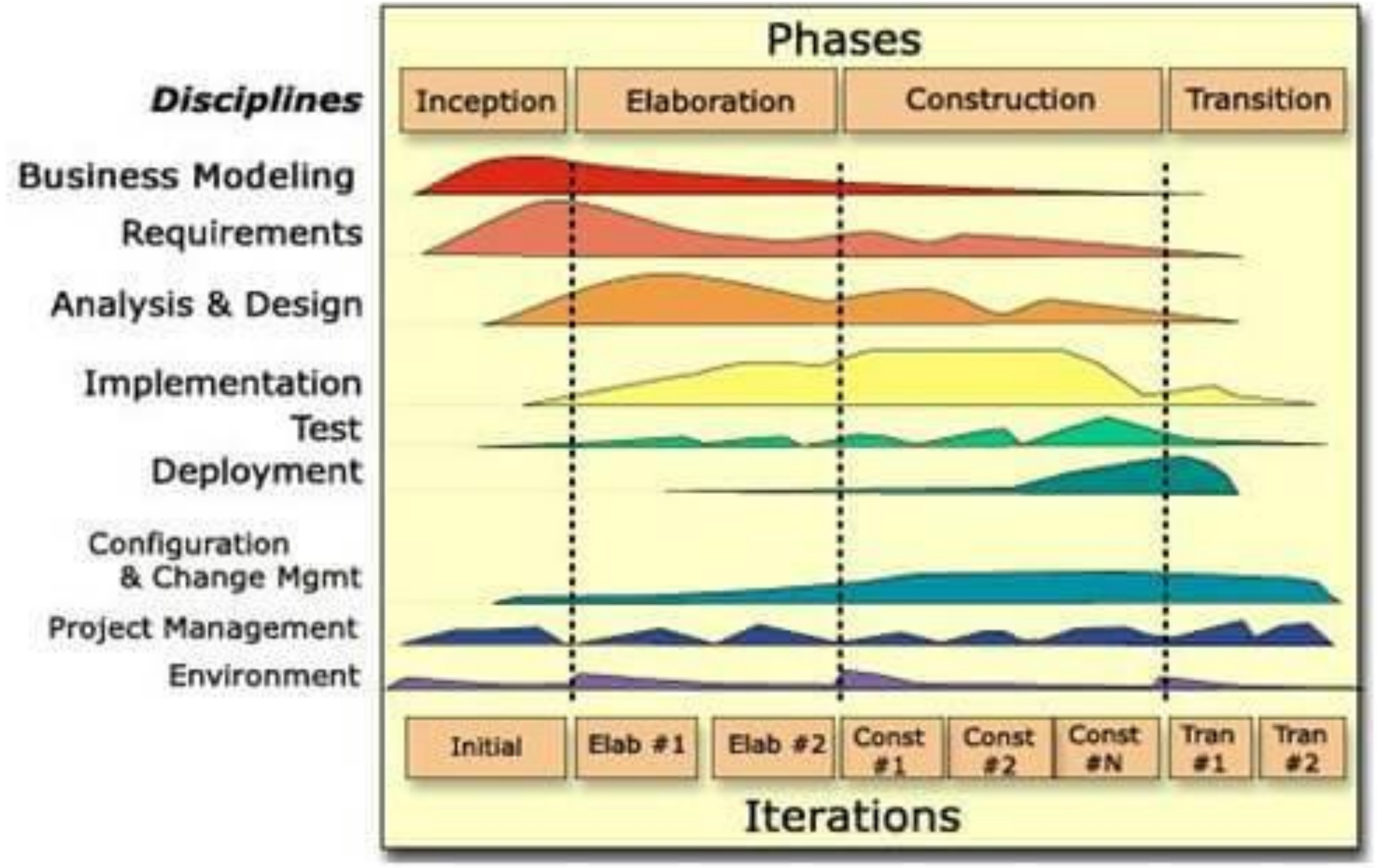
Milestones, Phases and Releases

- **Milestone** - the point at which an iteration formally ends; corresponds to a release point. Major and minor milestones.



- **Phase** - the time between two major project milestones, during which a well-defined set of objectives is met, artifacts are completed, and decisions are made to move or not move into the next phase.
- **Release** - a subset of the end-product that is the object of evaluation at a major milestone.

Workload during the Phases and Workflows (Disciplines)

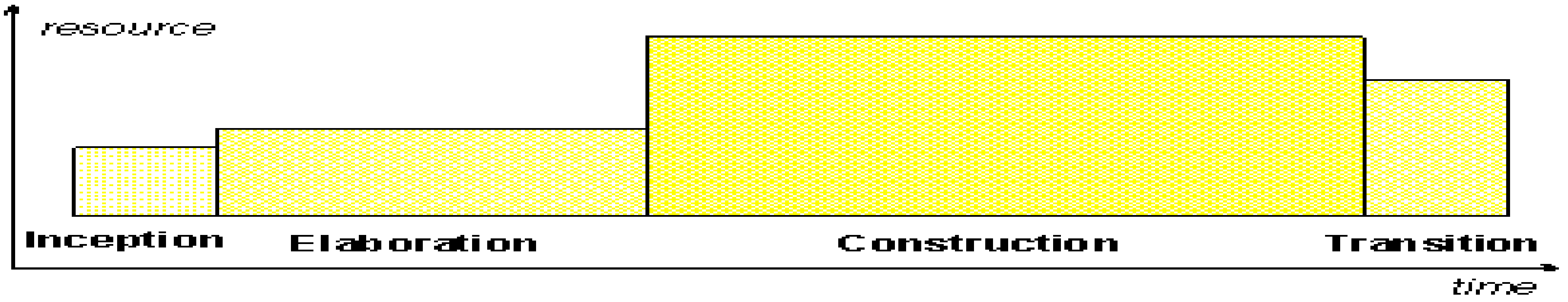


Source: https://larion.com/

Phases are not identical in terms of schedule and effort

- A typical initial development cycle for a medium-sized project should anticipate the following distribution between effort and schedule:

	Inception	Elaboration	Construction	Transition
Effort	~5 %	20 %	65 %	10%
Schedule	10 %	30 %	50 %	10%

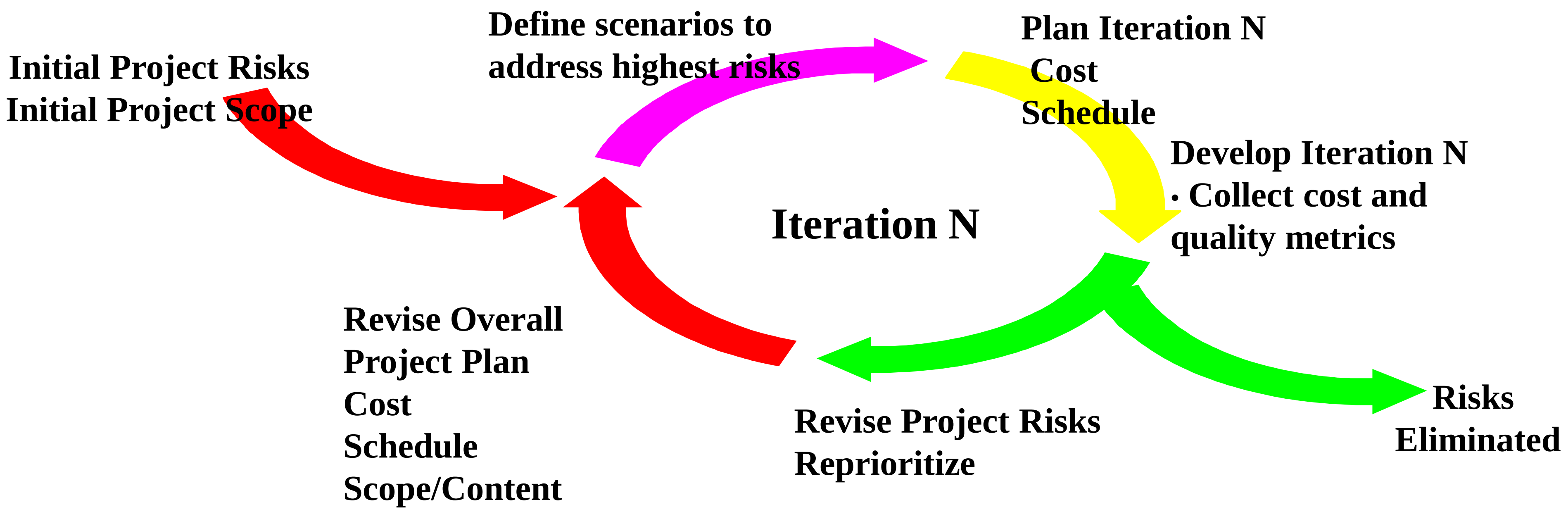


- Project plan: a time-sequence set of activities and task, assigned to resources, containing task dependencies, for the project. Iteration Plan.
- Determining the number of iterations and the length of each iteration

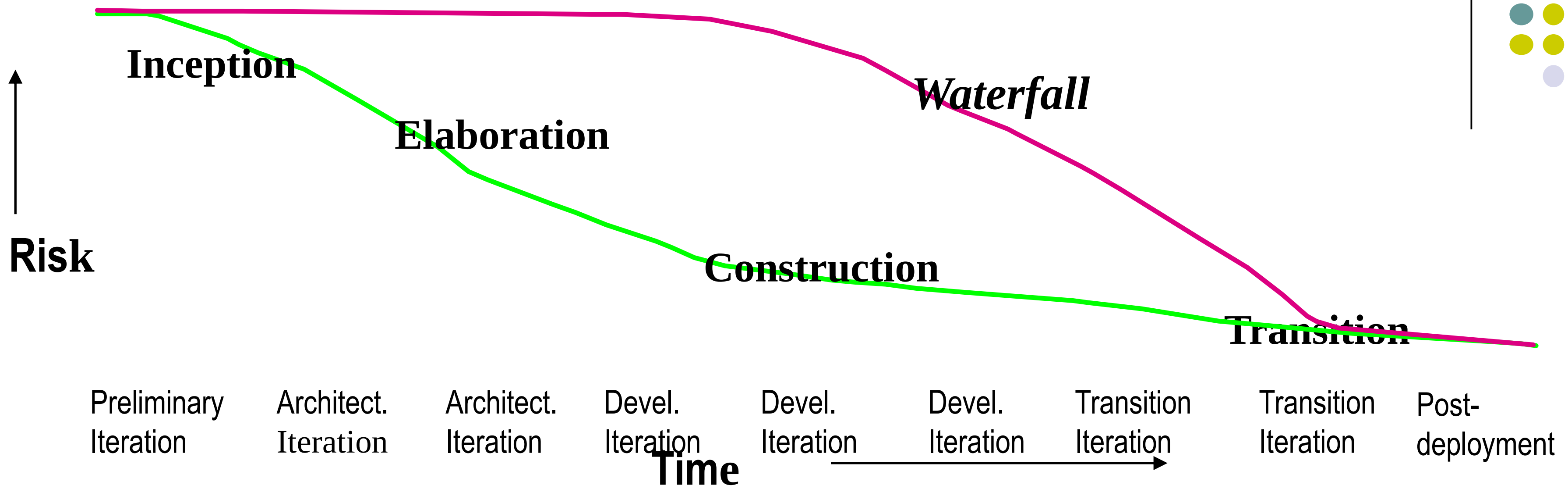
A Risk-Driven Approach

- A **risk** is a variable that, within its *normal distribution*, can take a value that endangers or eliminates success for a project
- Attributes of a risks:

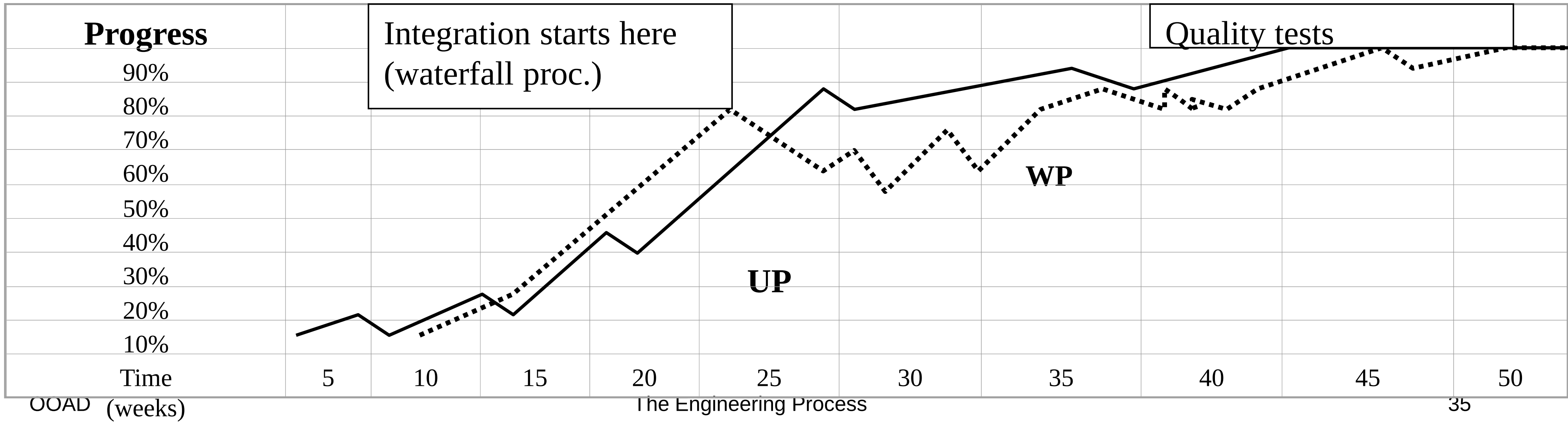
1. **Probability** of occurrence
2. **Impact** on the project (severity)
3. **Magnitude** indicator: *High, Significant, Moderate, Minor, Low*.



Risks in Iterative and Waterfall Development Processes

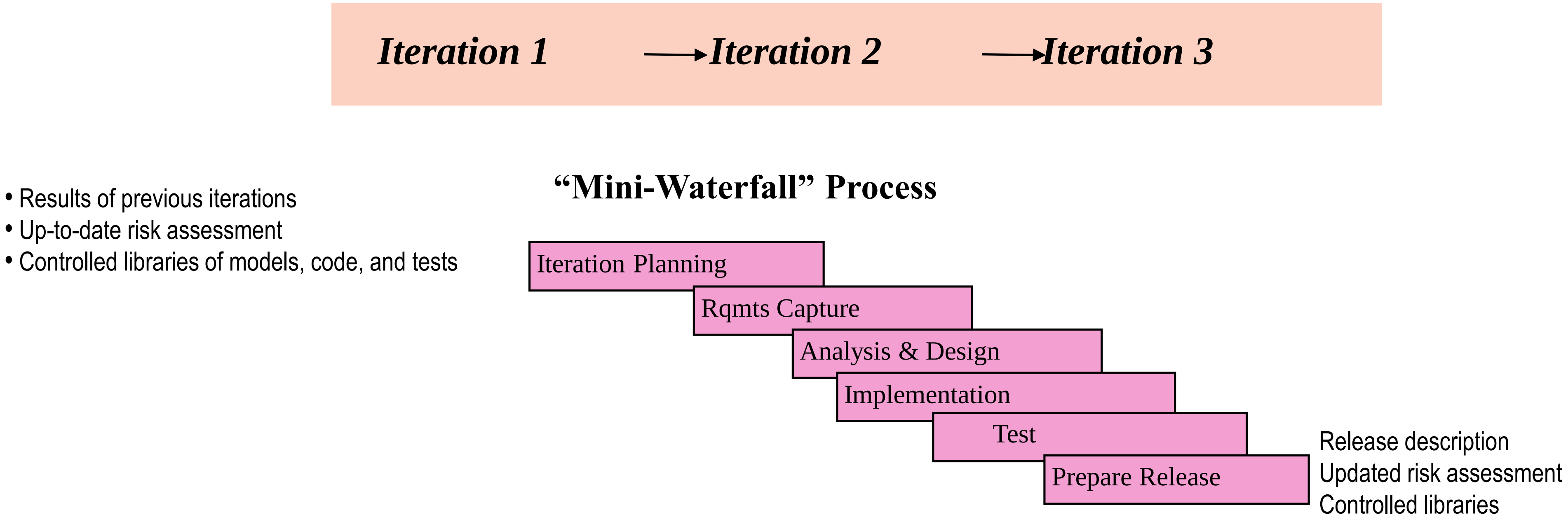


↑Risk Profile Comparison ↓Coding Comparison



Types of Risks

- Resource risks (organization, funding, people, time)
- Business risks (contract type, client, competitors)
- Technical risks (scope, technology, external dependency)
- Schedule risks



Resulting Benefits

- Planning and monitoring
- No “90% done with 90% remaining” effect
- Can incorporate problems/issues/changes into future iterations rather than disrupting ongoing production
- The project’s elements (testers, writers, tool-smiths, QA, etc.) can better schedule their work

Software Engineering Taxonomy

Taxonomy Project of the IEEE-CS Technical Council on Software Engineering (TCSE) has developed a unified taxonomy. Here, they present definitions of:

- **Forward engineering**
- **Reverse engineering**
- **Reengineering**
- **Round Trip Engineering**

Forward, Reverse and Reengineering

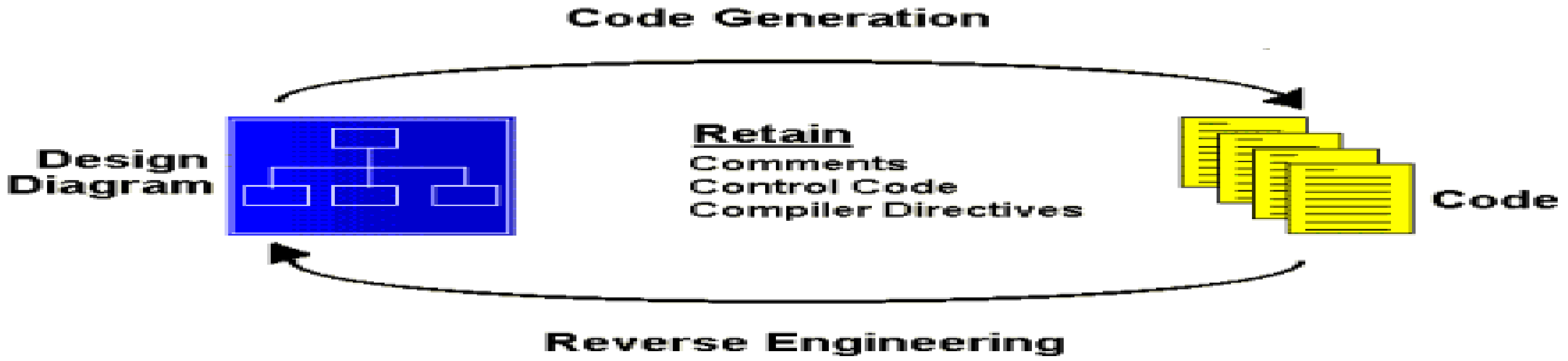
Forward engineering - "the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system."

Reverse engineering - "the process of analyzing a subject system with two goals in mind:
(1) to identify the system's components and their interrelationships; and,
(2) to create representations of the system in another form or at a higher level of abstraction."

Reengineering - "the examination of a subject system to reconstitute it in a new form and the subsequent implementation of the new form."

Round Trip Engineering

With **Round Trip Engineering** you can incrementally develop software, starting either from a new design or from an existing body of code. You can change the source code and keep design diagrams up to date, using any editor you like. Or you can change the design diagrams and keep the source code up to date.



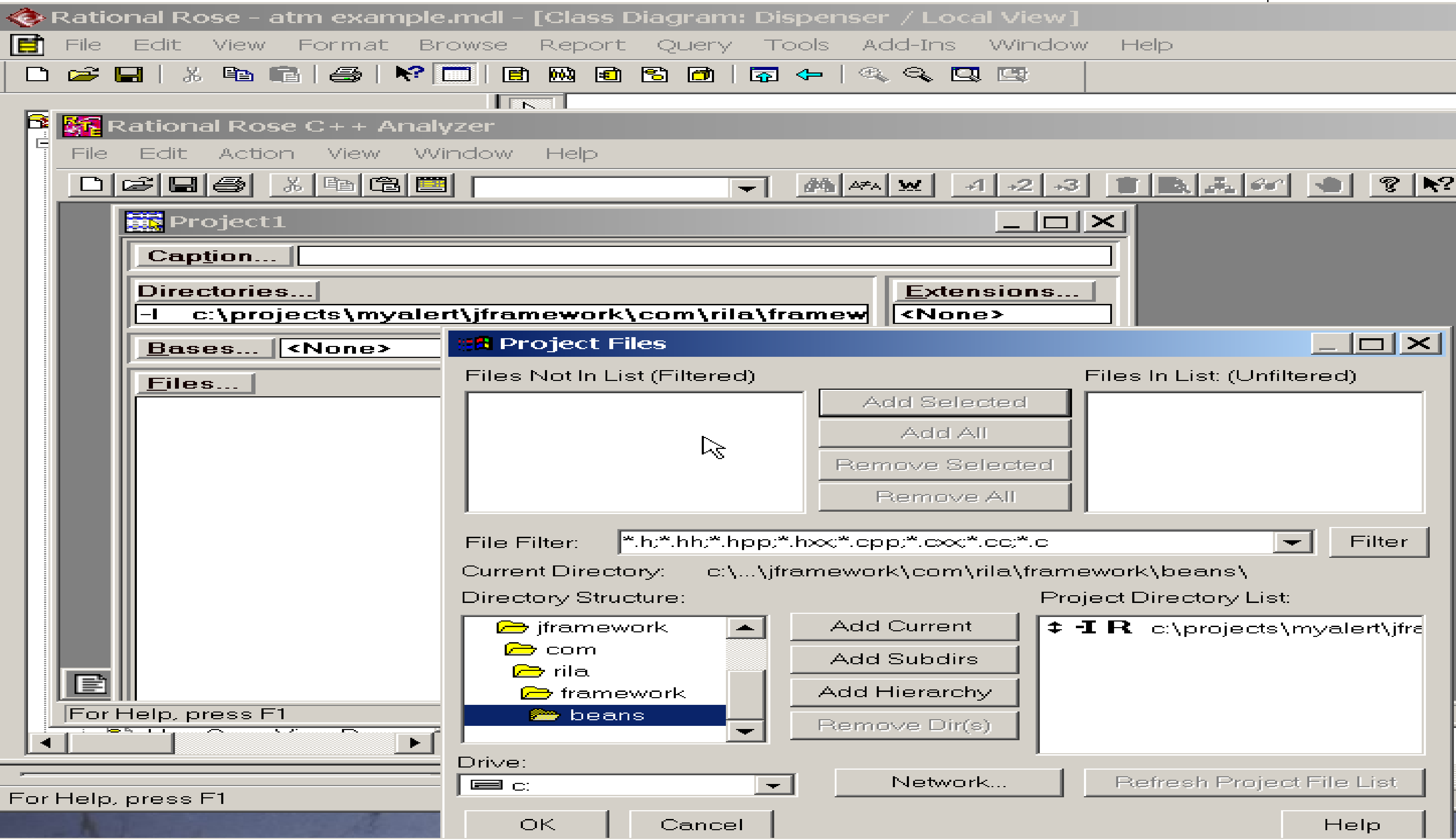
Reverse engineering is the process of evaluating an existing body of code to capture important information describing a system, and representing that information in a format useful to software engineers and designers.

Reverse Engineering with IBM Rose™

Rose Reverse engineering is the process of examining a program's source code to recover information about its design.

IBM Rose includes a C++ and Java Analyzer. The Rational Rose C++ Analyzer extracts design information from a C++ application's source code and uses it to construct a model representing the application's logical and physical structure.

The IBM Rose C++ Analyzer



-I c:\projects\myalert\jframework\com\rila\framew

<None>

<None>

File Filter: *.h;*.hh;*.hpp;*.hxx;*.cpp;*.cxx;*.cc;*.c

Current Directory: c:\...\jframework\com\rila\framework\beans\

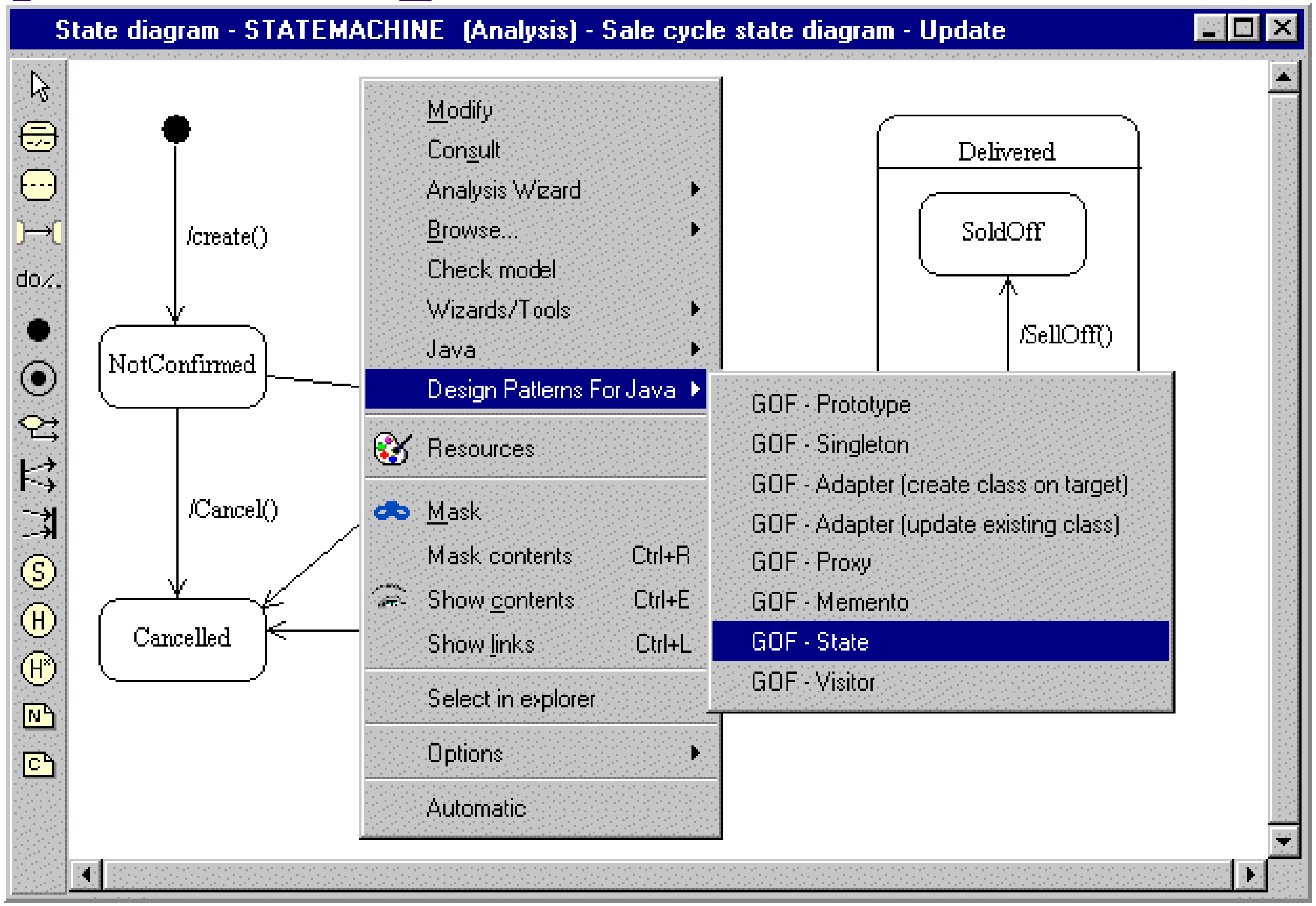
Project Directory List: ‡ -I R c:\projects\myalert\jfra

Drive: c:

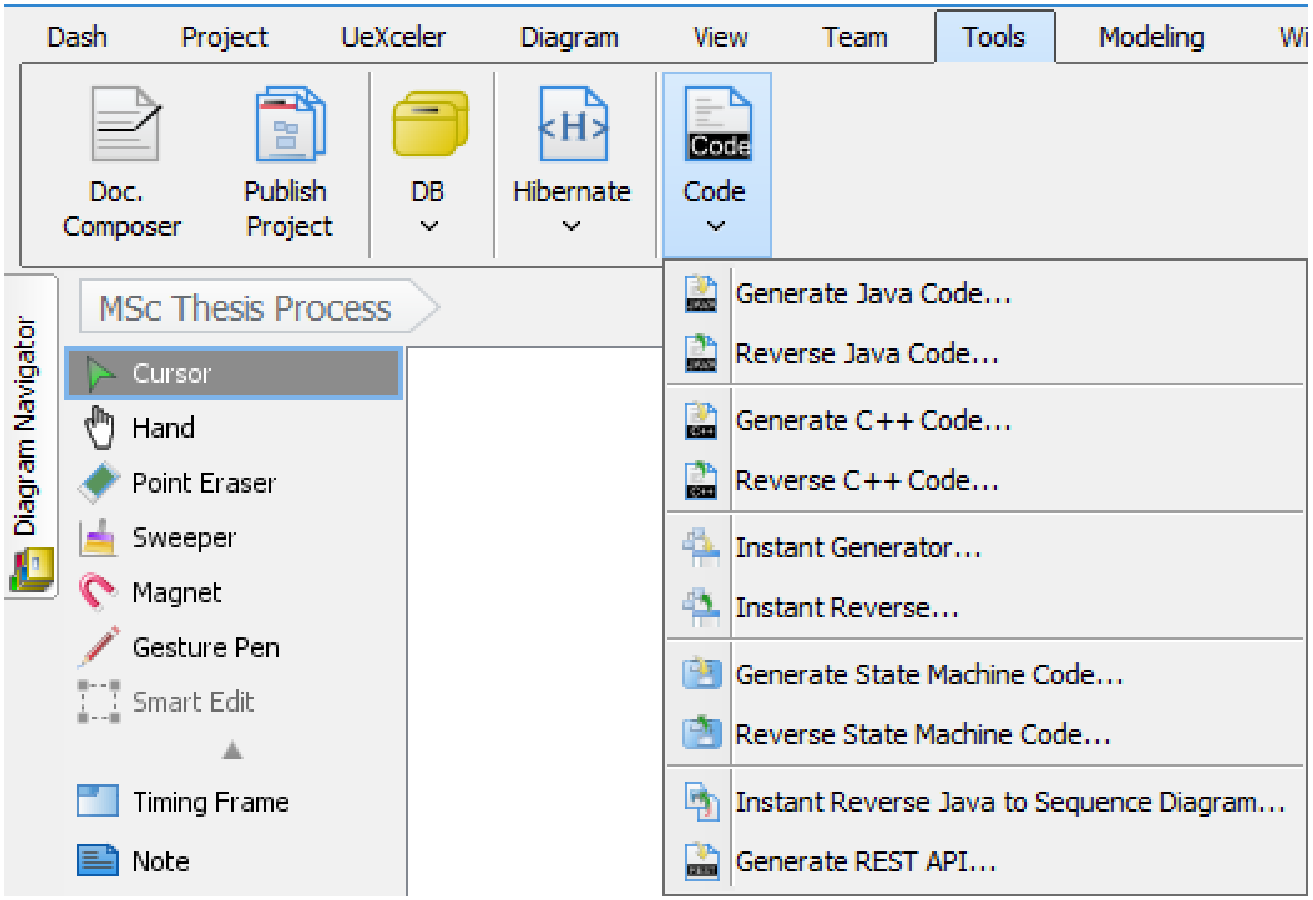
Generation Limits - Objecteering™

Generate code for the dynamic model thanks to PatternsObjecteering/UML associates Design Patterns and code generators to allow you to generate the code from the model's dynamic application. The ***State design pattern***, developed by Gamma, is automated, so as to automatically transform the UML state diagram model into a class model. The code generator then transforms this class model into Java code. By applying the State design pattern, you can be sure of generating Java code which corresponds to the state diagrams, thereby guaranteeing a highly efficient result.

Generation Limits - Objecteering™

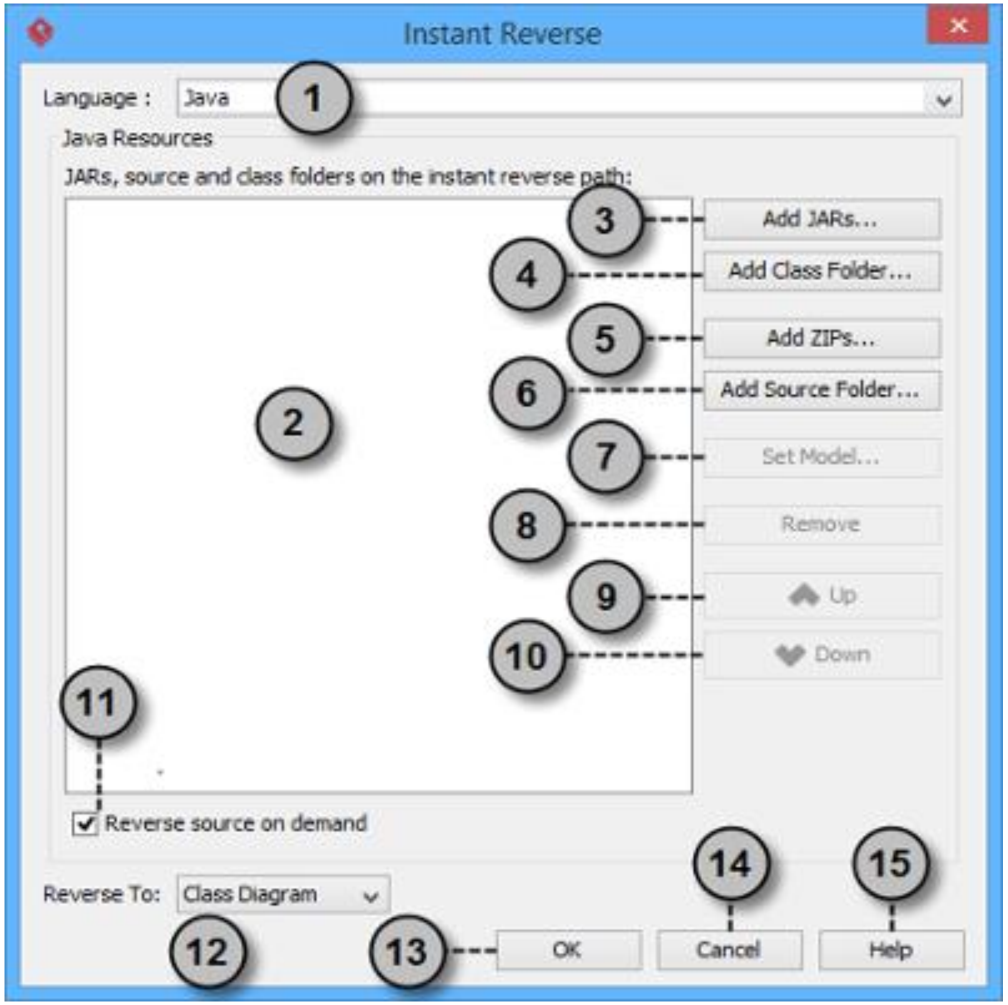


Generation processes in VP



Instant reverse Java sources and classes

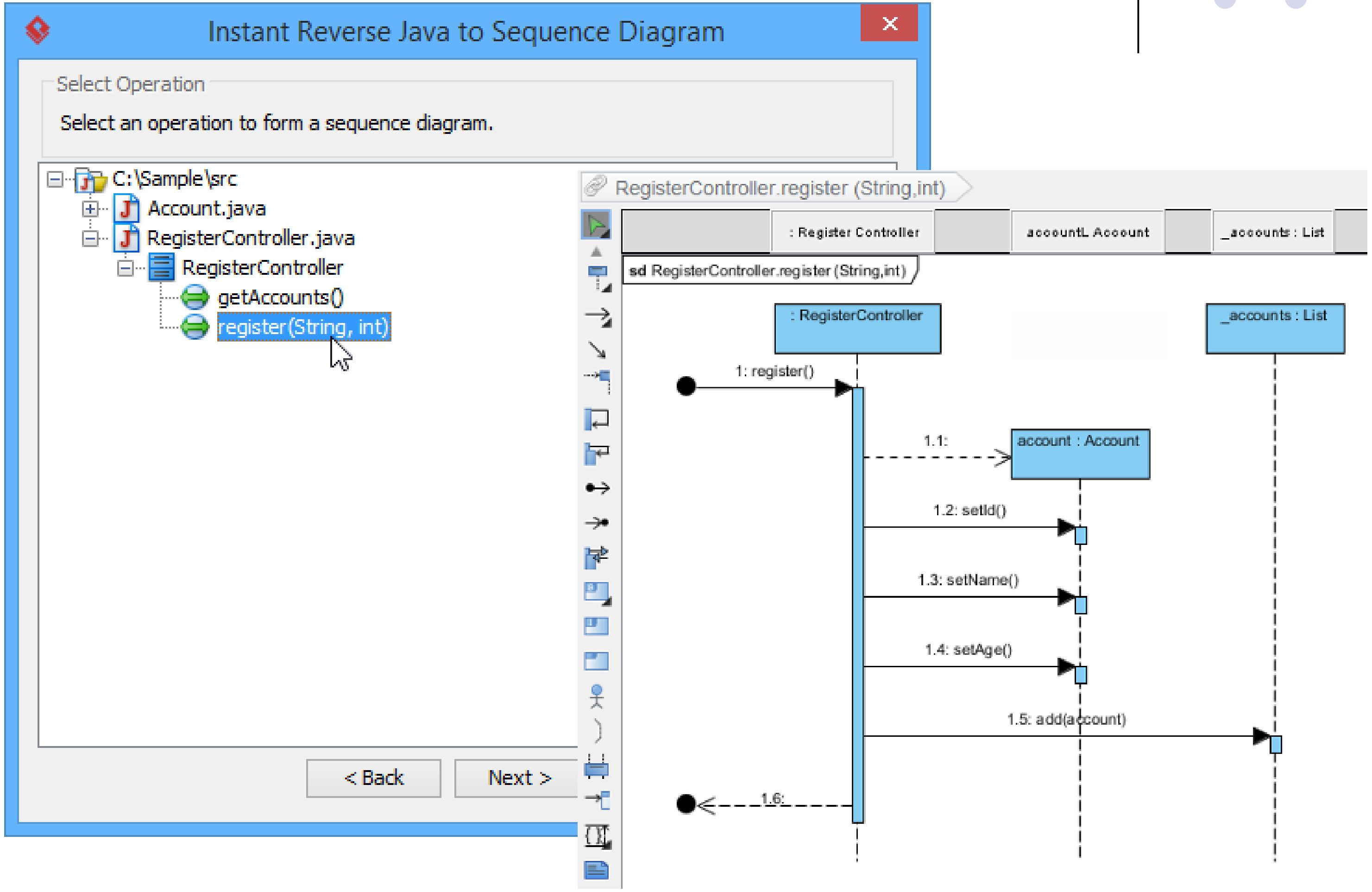
Tools ->
Code ->
Instant Reverse...



For more, see: https://www.visual-paradigm.com/support/documents/vpuserguide/276/277/28011_reverseengin.html

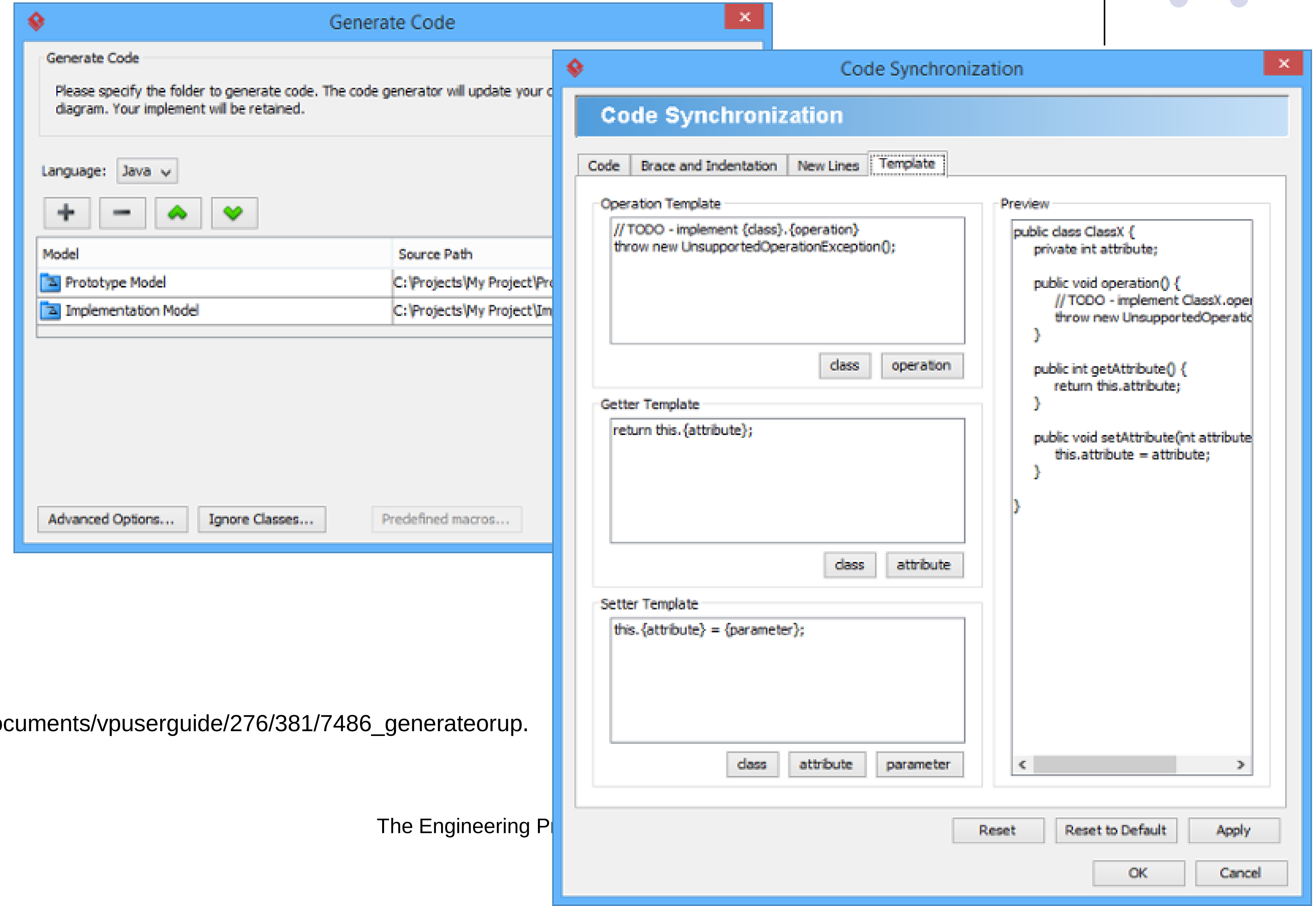
Instant reverse Java sources to sequence diagram

Tools -> Code -> Instant Reverse Java to Sequence Diagram...



Java Round-Trip: Generate/Update Java code

Tools -> Code -> Generate Java Code...



Please specify the folder to generate code. The code generator will update your c diagram. Your implement will be retained.

Language: Java

Model	Source Path
Prototype Model	C:\Projects\My Project\Pr
Implementation Model	C:\Projects\My Project\Im

```
// TODO - implement {class}.{operation}
throw new UnsupportedOperationException();
```

```
return this.{attribute};
```

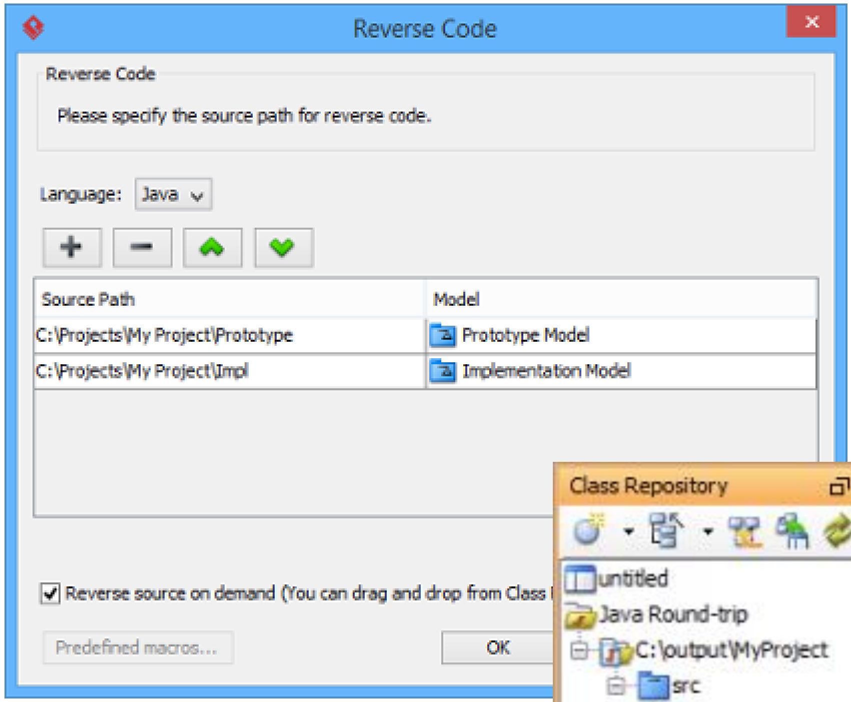
```
this.{attribute} = {parameter};
```

```
public class ClassX {
    private int attribute;

    public void operation() {
        // TODO - implement ClassX.oper
        throw new UnsupportedOperatic
    }

    public int getAttribute() {
        return this.attribute;
    }

    public void setAttribute(int attribute
        this.attribute = attribute;
    }

}
```

https://www.visual-paradigm.com/support/documents/vpuserguide/276/381/7486_generateorup.html

Java Round-Trip: Generate/Update UML classes from Java code

Tools -> Code -> Reverse Java Code...



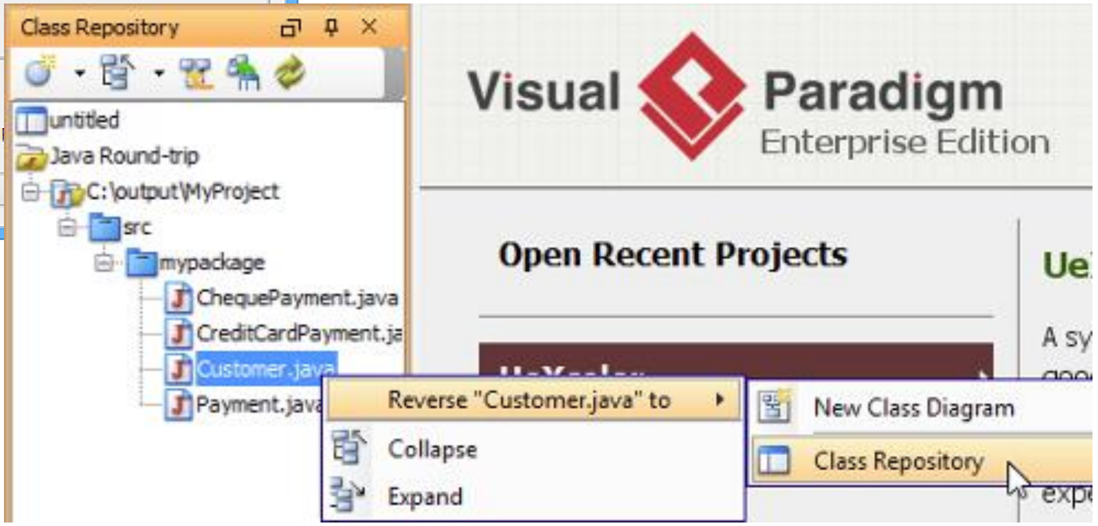
Reverse Code

Please specify the source path for reverse code.

Language: Java

Source Path	Model
C:\Projects\My Project\Prototype	Prototype Model
C:\Projects\My Project\Impl	Implementation Model

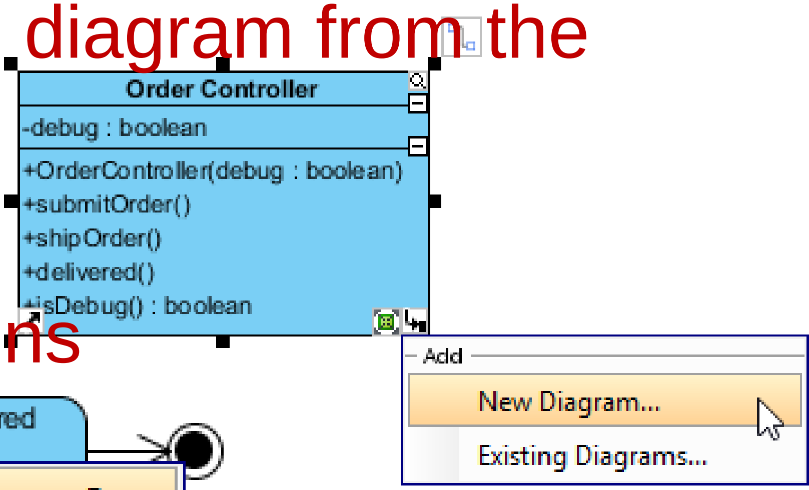
☑ Reverse source on demand (You can drag and drop from Class



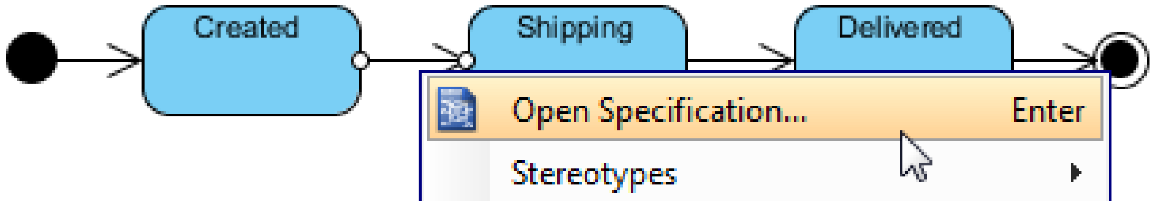
https://www.visual-paradigm.com/support/documents/vpuserguide/276/381/7530_generateorup.html

State Machine Diagram Code Generation

1. Creation of controller class
2. Creation of a sub-state machine diagram from the controller class

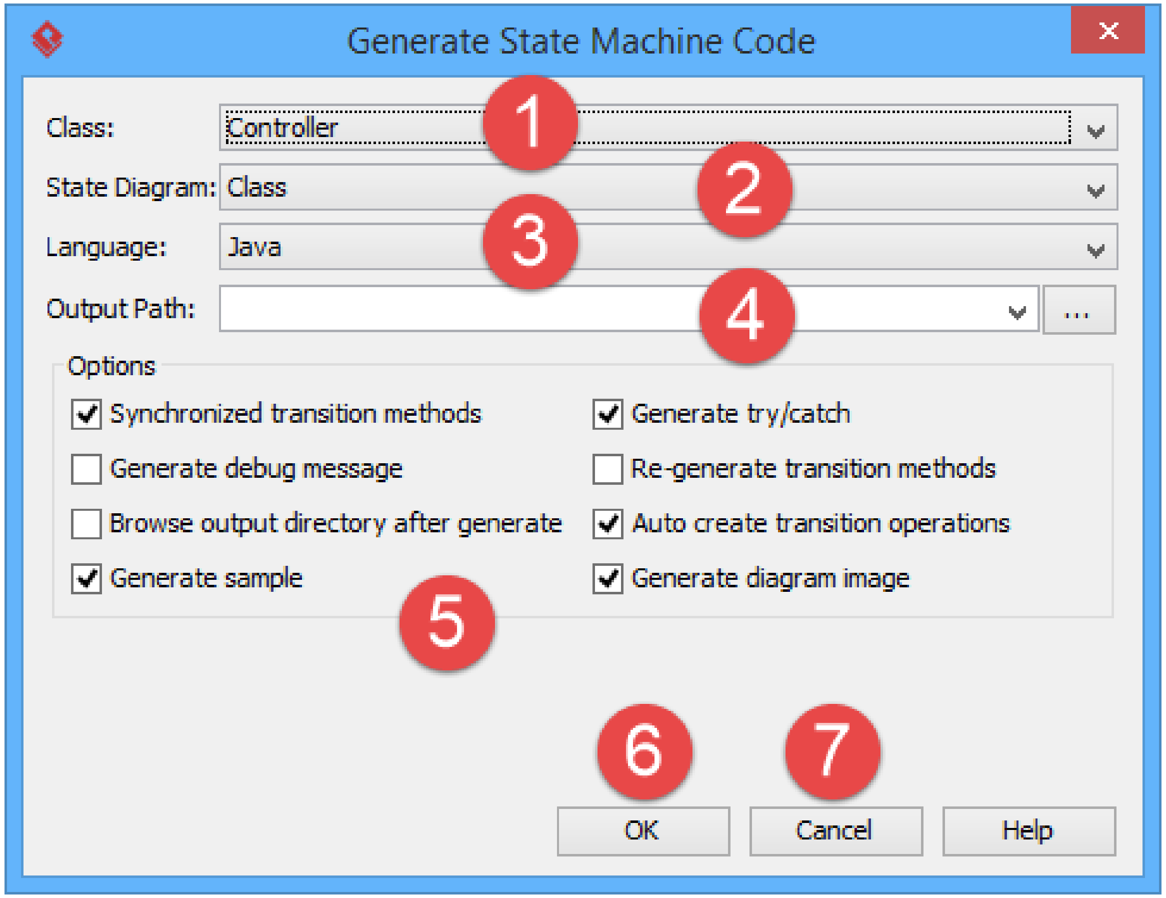


3. Assigning operations to transitions



4. Specifying method body for the entry/exit of state
5. Specifying method body for operation

Tools -> Code -> Generate State Machine Code...


Generate State Machine Code

Class: Controller

State Diagram: Class

Language: Java

Output Path:

Options

- [✓] Synchronized transition methods
- [ ] Generate debug message
- [ ] Browse output directory after generate
- [✓] Generate sample
- [✓] Generate try/catch
- [ ] Re-generate transition methods
- [✓] Auto create transition operations
- [✓] Generate diagram image

https://www.visual-paradigm.com/support/documents/vpuserguide/276/386/28107_generatingst.html

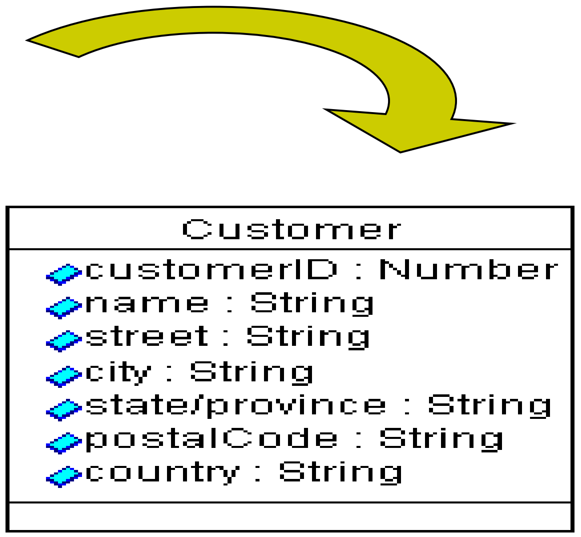
Reverse-engineering Relational DB

Replicating the structure of the database in a class model is relatively straight-forward.

Create a Class for each Table

For each column, create an attribute on the class with the appropriate data type. Try to match the data type of the attribute and the data type of the associated column as closely as possible.

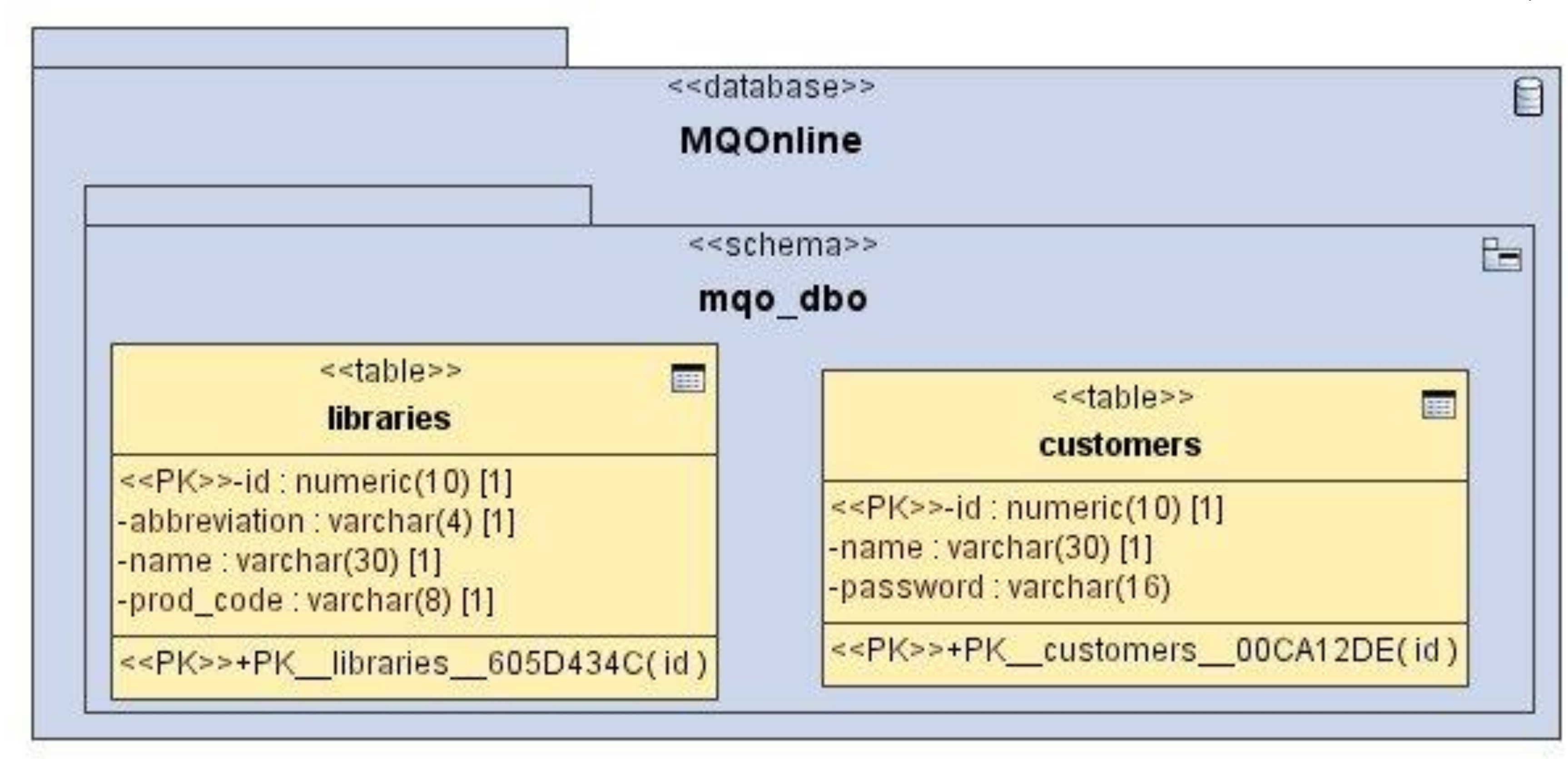
Column Name	**Data Type**
Customer_ID	Number
Name	Varchar
Street	Varchar
City	Varchar
State/Province	Char(2)
Zip/Postal Code	Varchar
Country	Varchar



Example by MagicDraw™ - from DDL…

```
--@(#) C:\md\MagicDraw UML 6.0\script.ddl

DROP TABLE MQOnline.mqo_dbo.customers;
DROP TABLE MQOnline.mqo_dbo.libraries;
CREATE TABLE MQOnline.mqo_dbo.libraries
(
    id numeric (10) NOT NULL,
    abbreviation varchar (4) NOT NULL,
    name varchar (30) NOT NULL,
    prod_code varchar (8) NOT NULL,
    CONSTRAINT MQOnline.mqo_dbo.PK__libraries__605D434C PRIMARY KEY(id)
);

CREATE TABLE MQOnline.mqo_dbo.customers
(
    id numeric() (10) NOT NULL,
    name varchar (30) NOT NULL,
    password varchar (16),
    CONSTRAINT MQOnline.mqo_dbo.PK__customers__00CA12DE PRIMARY KEY(id)
);
```

Example by MagicDraw™ - …to E-R diagram



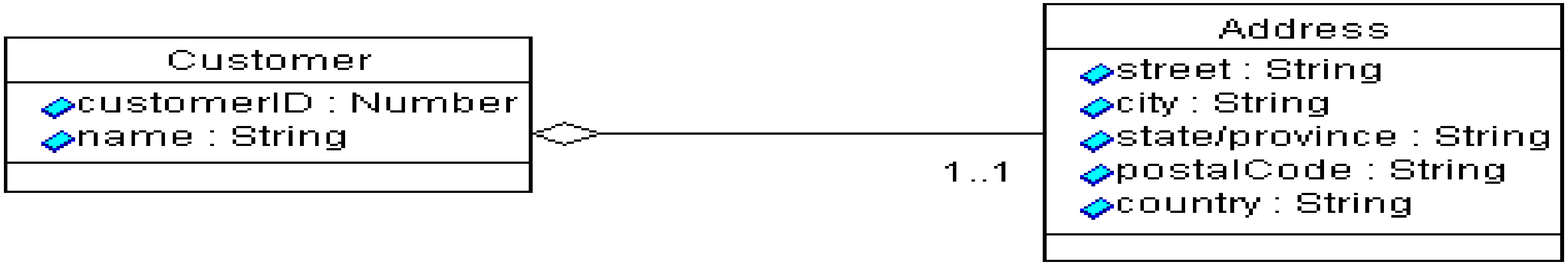
<<database>>
MQOnline
<<schema>>
mqo_dbo
<<table>>
libraries
<<PK>>-id : numeric(10) [1]
-abbreviation : varchar(4) [1]
-name : varchar(30) [1]
-prod_code : varchar(8) [1]
<<PK>>+PK__libraries__605D434C(id)
<<table>>
customers
<<PK>>-id : numeric(10) [1]
-name : varchar(30) [1]
-password : varchar(16)
<<PK>>+PK__customers__00CA12DE(id)

Identify Embedded/Implicit Classes

The class that results from the direct table-class mapping will often contain attributes that *can be separated into a separate class*, especially *in cases where the attributes appear in a number of translated classes*. These *'repeated attributes'* may have resulted from *denormalization of tables* for performance reasons, or may have been the result of an oversimplified data model.

Example: revised **Customer** class, with extracted **Address** class. The association drawn between these two is an **aggregation**, since the customer's address can be thought of as being **part-of**

.

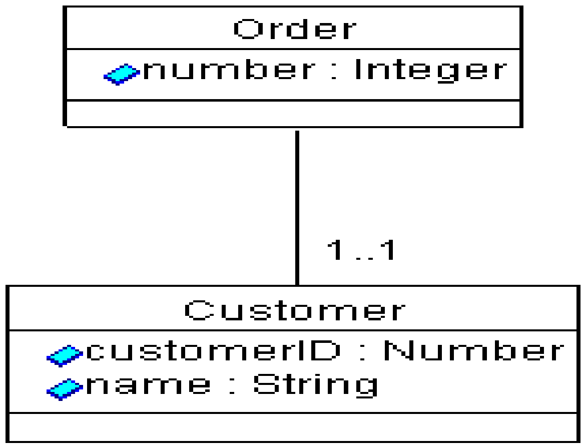


Handle Foreign-Key Relationships

For each foreign-key relationship in the table, create an association between the associated classes, removing the attribute from the class which mapped to the foreign-key column. If the foreign-key column was represented initially as an attribute, remove it from the class.

Column Name	**Data Type**
Number	Number
<<FK>> Customer_ID	Varchar

Example: In the **Order** table above, the **Customer_ID** column is a **foreign-key reference**; this column contains the primary key value of the Customer associated with the Order.



Handle *Many-to-Many* Relationships

RDBMS data models represent many-to-many relationships with a mean which has been called a
join table
, or an
association table
- a foreign key reference can only contain a reference to a single foreign key value; when a single row may relate to many other rows in another table, a join table is needed to associate them.

Handle *Many-to-Many* Relationships – DB model

Product Table

Column Name	**Data Type**
Product_ID	Number
Product_Name	Varchar

Supplier Table

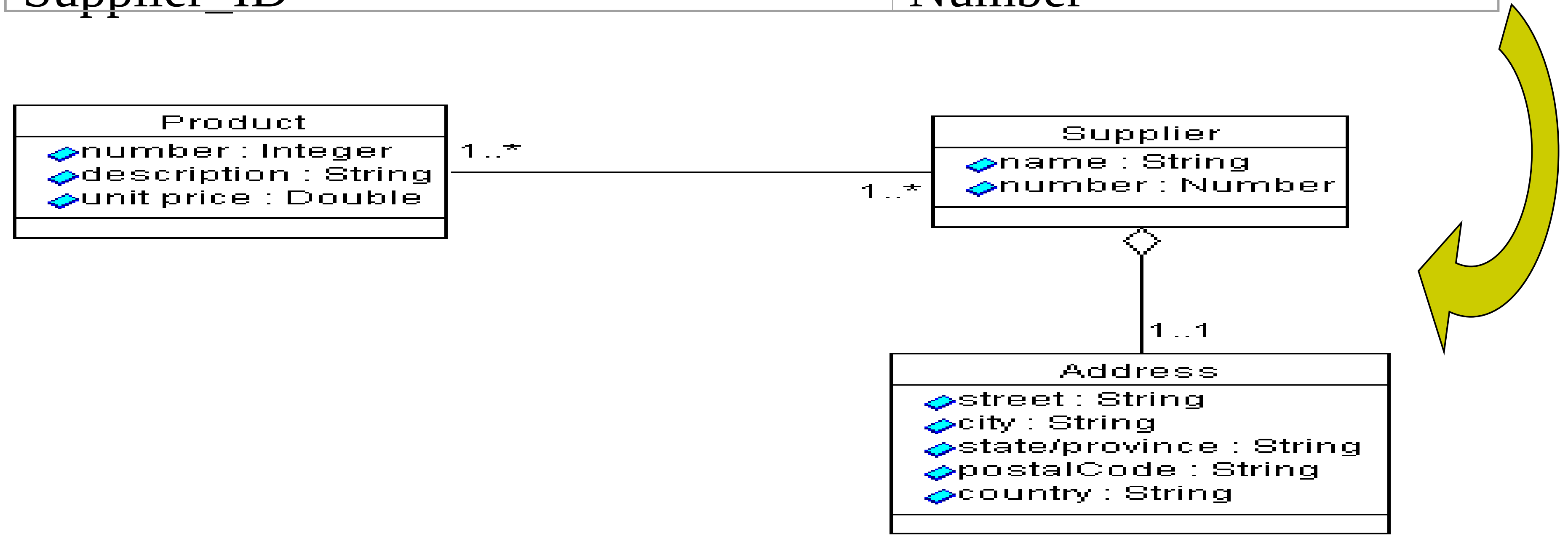
Column Name	**Data Type**
Supplier_ID	Number
Supplier_Name	Varchar

Product-Supplier Table

Column Name	**Data Type**
Product_ID	Number
Supplier_ID	Number

Handle *Many-to-Many* Relationships – Object Model

Product-Supplier Table	
Column Name	**Data Type**
Product_ID	Number
Supplier_ID	Number



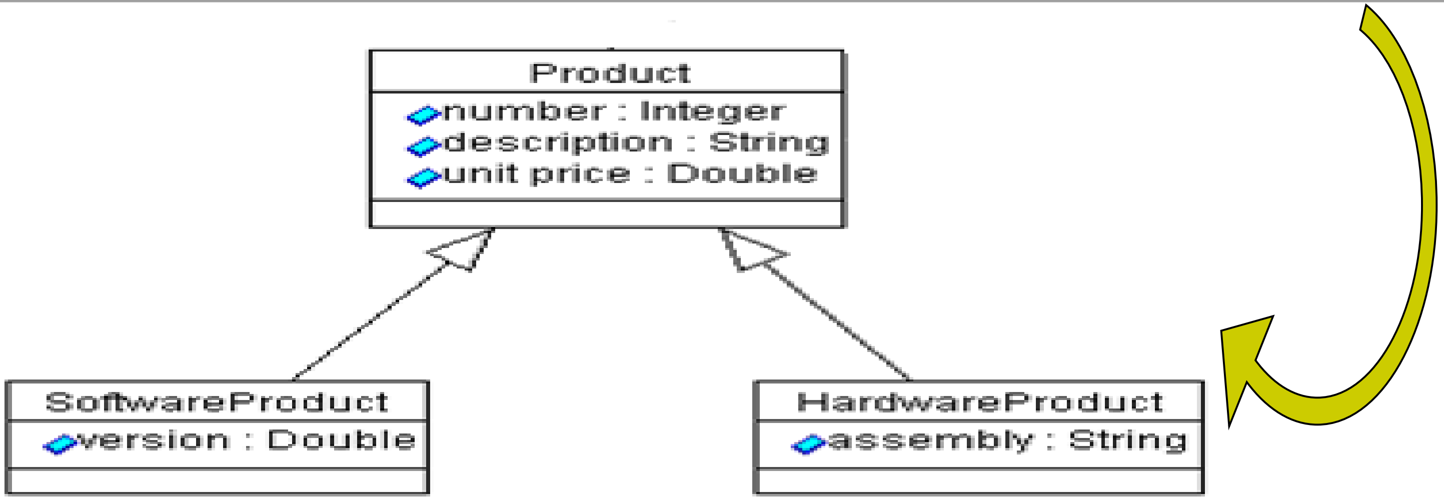
Introducing Generalization

Sometimes common structure results from denormalization for performance, such as is the case with the 'implicit' **Address** table which we extracted into a separate class. In other cases, tables share more fundamental characteristics which we can extract into a generalized parent class with two or more sub-classes. Look for repeated columns in two tables:

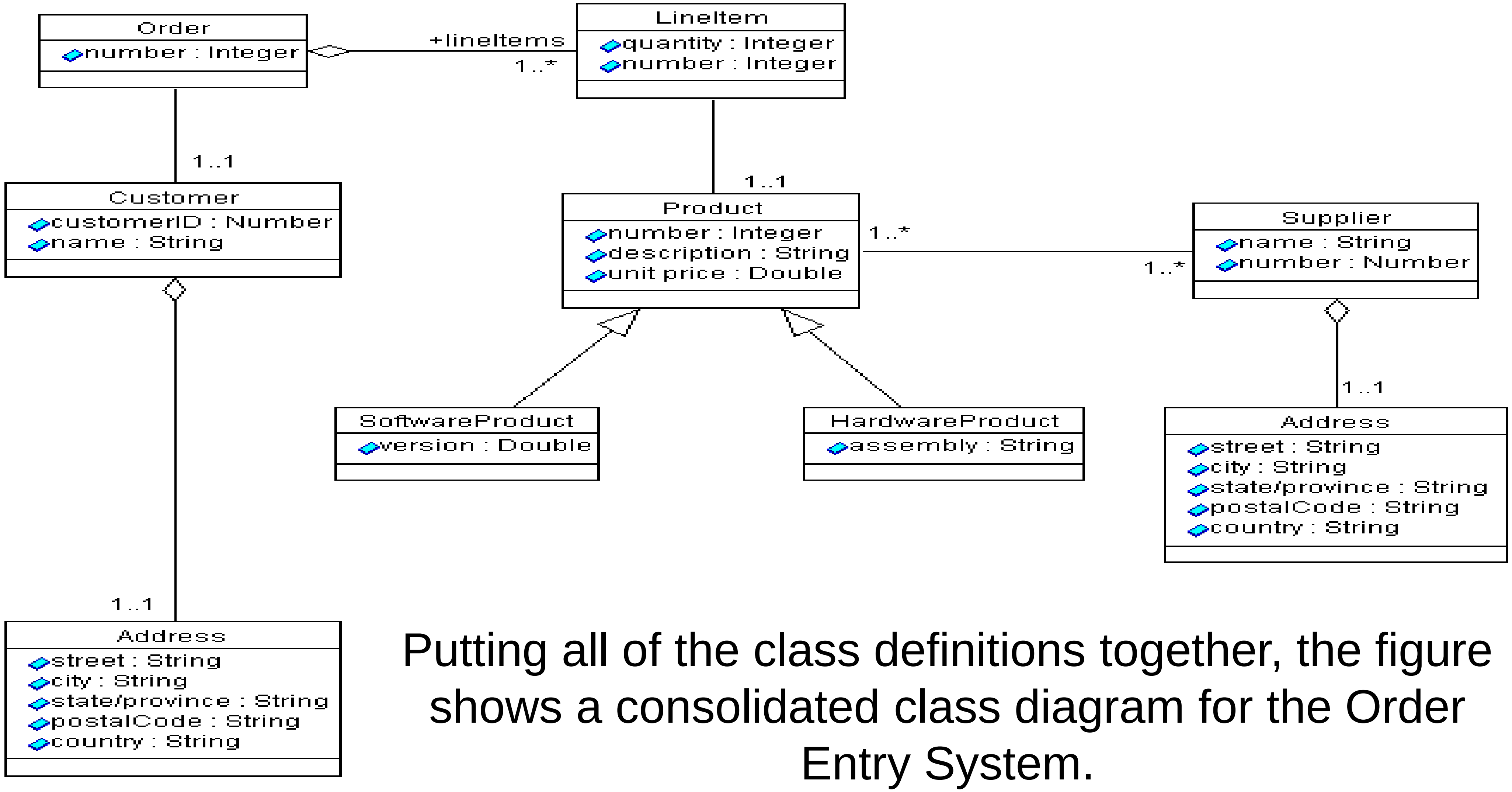
SW Product			**HW Product**	
Column Name	Data Type		**Column Name**	**Data Type**
Product_ID	**Number**		**Product_ID**	**Number**
Name	**Varchar**		**Name**	**Varchar**
Description	**Varchar**		**Description**	**Varchar**
Price	**Number**		**Price**	**Number**
Version	**Varchar**		**Assembly**	**Number**

Class Generalization from the Data Model

SW Product			HW Product	
Column Name	Data Type		Column Name	Data Type
Product_ID	Number		Product_ID	Number
Name	Varchar		Name	Varchar
Description	Varchar		Description	Varchar
Price	Number		Price	Number
Version	Varchar		Assembly	Number

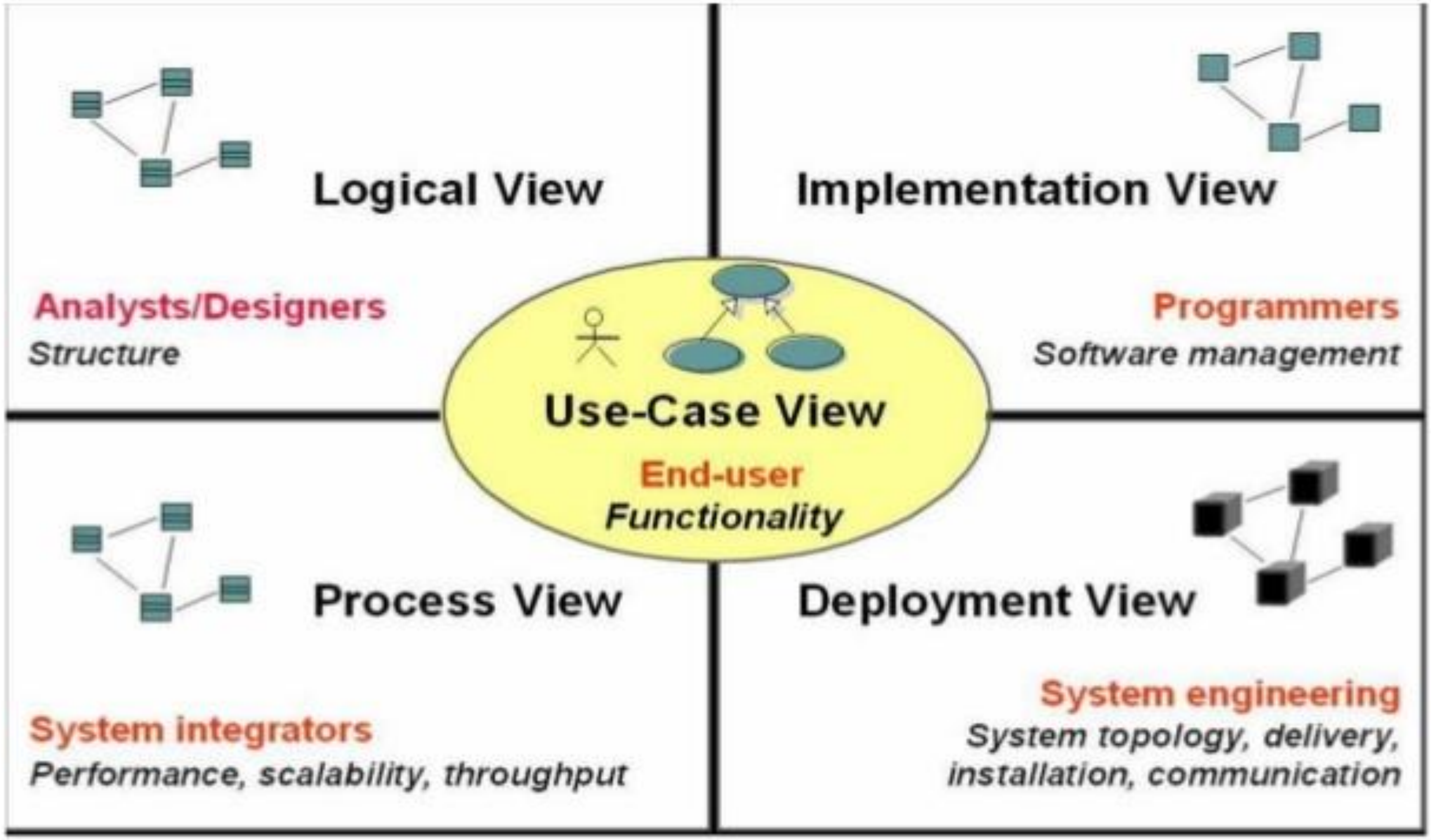


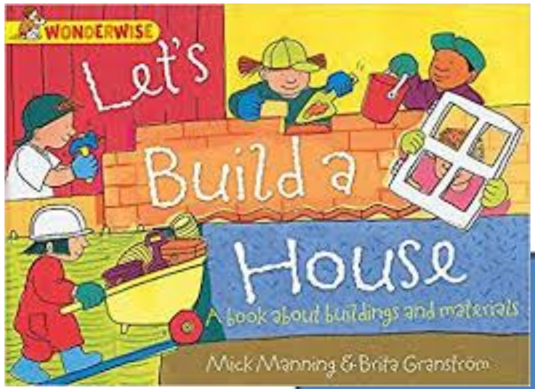
Finally – an Object Model

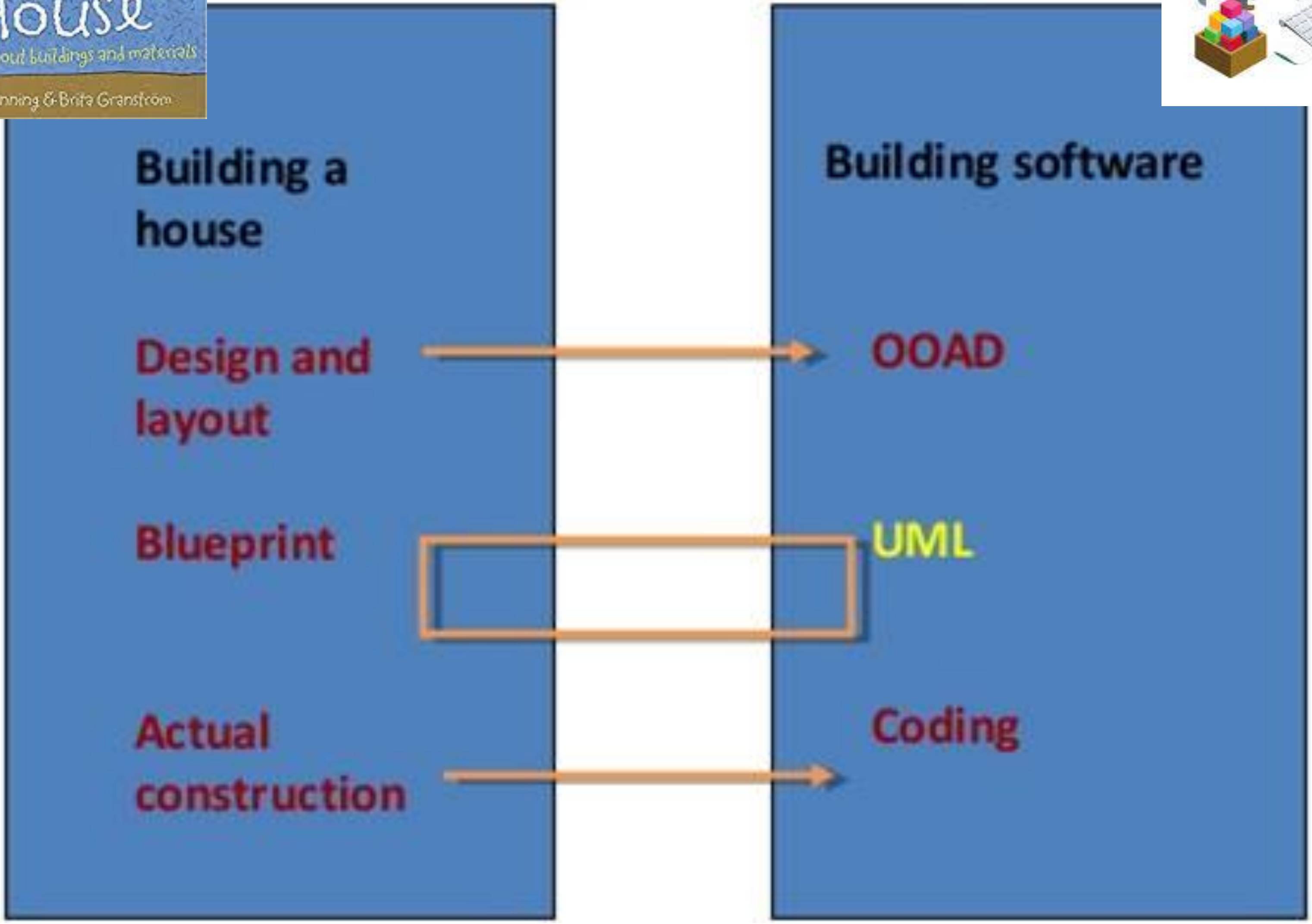


Putting all of the class definitions together, the figure shows a consolidated class diagram for the Order Entry System.

RUP 4+1 View


Logical View
Analysts/Designers
Structure
Implementation View
Programmers
Software management
Use-Case View
End-user
Functionality
Process View
System integrators
Performance, scalability, throughput
Deployment View
System engineering
System topology, delivery, installation, communication


WONDERWISE
Let's Build a House
A book about buildings and materials
Mick Manning & Brita Granström


Building a house
Design and layout
Blueprint
Actual construction
Building software
OOAD
UML
Coding

time to say
goodbye