

CEISAR White Paper on Foundation

1	Objectives of CEISAR White Paper on Foundation	4
1.1	What is CEISAR?	4
1.2	Why is Foundation so important?	4
1.3	Many questions	5
1.4	Contents of this White Paper	5
2	Synergy is Sharing Resources and Reusing Models	7
2.1	Why is Centralization so difficult?	7
2.2	Synergy is Reusing Models and Sharing Resources?	8
2.3	Reusing first or Sharing first?	10
3	What is Operation Foundation?	13
3.1	Exchange and Building Foundation	14
3.2	Which elements are in Operation Foundation?	18
3.3	Which elements belong to Exchange or Building Foundations?	24
4	What is Transformation Foundation?	27
4.1	Reusable Roles for Transformation	27
4.2	Reusable IT Configuration for Transformation	28
4.3	Transformation Approach	28
4.4	Transformation Engineering Tools	30
4.5	Transformation Tools for Management	32
4.6	Information Model for Transformation	33
5	Summary of what really works today	34
6	How to measure Foundation Value	36
6.1	Value examples	36
6.2	Main Value = Increased Agility and Reduced Costs	40
6.3	How to convince top management	40
6.4	Define a Global Plan	45
6.5	First step in obtaining Foundation budget is optimization of current expenses	46
6.6	How to define investments on Foundation?	47
7	What is a good Foundation?	49
7.1	Characteristics of a good Reusable Black Function	49
7.2	A structure and not a flat list of White Components	50
7.3	Scalability and performance	51
7.4	Solution Reused for several Enterprises	52
7.5	Foundation Reused for several Enterprises	53
7.6	Add Specific Functions to Foundation	53
7.7	Enterprise Model and detailed Model	53
8	How to get a Good Foundation: Success Factors	54
8.1	Foundation effort is not the same for Exchange and Building Foundation	54
8.2	The difficulty of creating Building Foundation	56
8.3	Which Transformation Tools to help Reuse?	57
8.4	Quality and experience of Foundation architects	59
8.5	Foundation Customer is required	60
8.6	What Organization is most efficient for Foundation?	60
8.7	Planning for a Foundation approach	62
8.8	Foundation life cycle	64
8.9	Build or Buy Foundations?	65
8.10	Foundation evolutions	65
9	How can current Solution teams use Foundation efficiently?	67
9.1	Main difficulties	67
9.2	How to convince Solution Teams to use Foundation?	67
9.3	Which Governance?	67
9.4	Which new Solution Models must Reuse Foundation?	68
9.5	Organization	68

9.6	Reused Foundation allows a new Approach.....	68
9.7	How Foundation supports Solution teams	69
9.8	Solution Versions and Foundation Versions	69
9.9	What risk when Solutions depend on Foundation?	69
10	Foundation and Packages	70
10.1	Trend towards Reuse	70
10.2	Using a Package is importing its Foundation.....	70
10.3	Can Package Foundation become Enterprise Foundation?.....	71
10.4	How to select a Package if an Enterprise Foundation already exists?	72
10.5	Reuse by Package or Reuse by Foundation ?.....	73
11	An example of a powerful Building Foundation for an Insurance Solution	74
11.1	Why Foundation?	74
11.2	A Global Insurance Solution for different Companies	74
11.3	Build the Insurance Foundation	78
11.4	How each Company Reuses the Insurance Package to Build its own Model.....	78
11.5	What is the final Structure?	80
11.6	Value of Building Foundation.....	82
12	Exhibits	84
12.1	Exhibit "List of Questions on Foundation"	84
12.2	Exhibit "CEISAR Terminology"	85
12.3	Exhibit "Detailed CEISAR Cube"	87
12.4	Exhibit "Togaf Technical Reference Model"	88
12.5	Exhibit "Sharing" is not Reusing	91
12.6	Exhibit "Complement to Operation Foundation"	93
12.7	Exhibit "Sharing Operation Resources"	99
12.8	Exhibit "Complements to Transformation Foundation"	100
12.9	Exhibit "Sharing Transformation Resources"	101
12.10	Exhibit "How to make Decisions"	103

This document was prepared with the help of many experts who took the time to explain their vision and offer.

We would especially like to thank:

Air France	Laurent Mondemé
Axa	Bruno Gay
Crédit du Nord	Bertrand Ledu, Christine Crépeau, Christian de Flers
Total	Rosanna di Leo, Claude Fauconnet

Arismore (representing Togaf from Open Group)	Eric Boulay, François Maitre
Google	Patrick Chanezon
IBM	Marc Famiente
Microsoft	Bernard Ourghanlian, Steve Sfartz
Oracle	Didier Medal, Yves Lhérault
Orchestra Network	Pierre Bonnet
Sales Force	Jean-Louis Baffier
SFEIR	Didier Girard
SAP	Thierry Pierre
Softeam	Philippe Desfray

Writing conventions

Enterprise Architecture is a complex topic. To help understand it, CEISAR uses some terms. These terms are easily recognizable in the text: they all start with a capital letter.

Definitions of main terms are described in Exhibit "CEISAR Terminology".

The full CEISAR glossary is available on www.ceisar.org.

1 Objectives of CEISAR White Paper on Foundation

1.1 What is CEISAR?

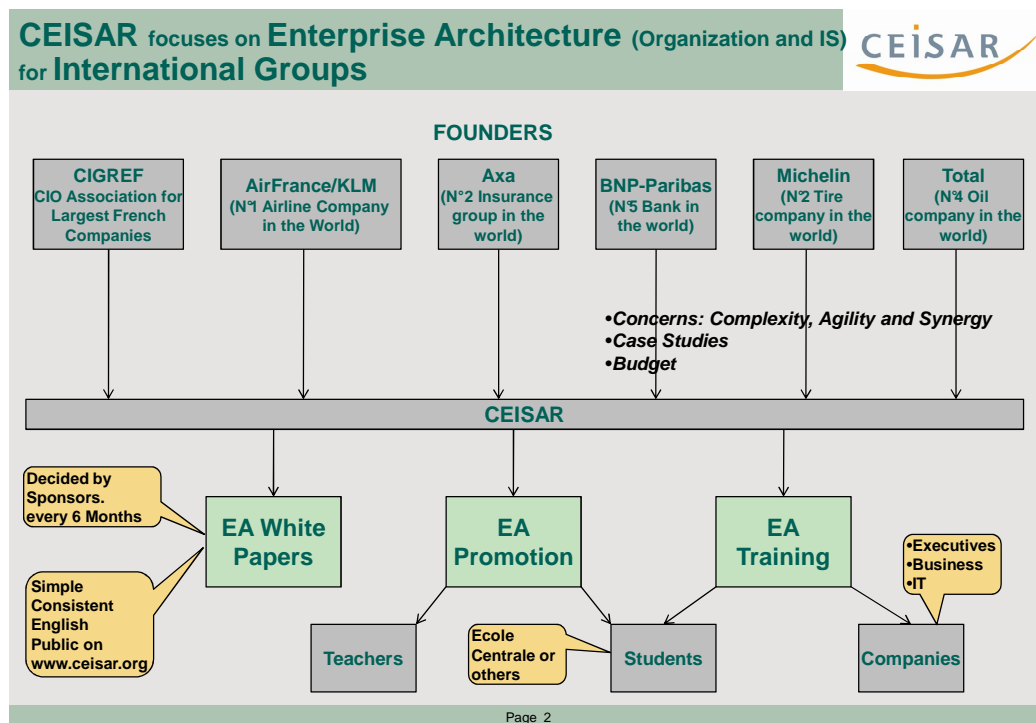
The CEISAR is the Center of Excellence on **Enterprise Architecture** based at **Ecole Centrale Paris**. Its role is to produce public white papers accessible on www.ceisar.org, to promote Enterprise Architecture and to offer EA training for Enterprises and Universities.

The CEISAR was founded by large International Companies: Air France, Axa, BNP-Paribas, Michelin and Total.

Every 6 months, the CEISAR produces a white paper on a topic defined by its sponsors.

For the period April-October 2009 the topic was "Foundation".

Foundation has been defined as everything which is **reusable by the different Solution Models**, like: Reusable Transformation Approach, Reusable Development Tools, IT infrastructure, Reusable role definitions, Reusable Information Models, Reusable Solution Models, Reusable Components, Reusable SOA Services...



1.2 Why is Foundation so important?

As presented in former white papers, the key concerns of Enterprises are:

- **Complexity**: number of Products, Processes, Partners, operating Countries, this is increasing so much that Enterprise Actors have more and more difficulty in understanding how their Enterprise Operates
- **Agility**: according to Globalization, time to market, increasing competition... deadlines to Transform the Enterprise should be shorter and shorter, but most Enterprises complain of their incapacity to move fast
- **Synergy**: to manage a decentralized Group made up of Companies supposed to define budget at Company level, get results, compare and make decisions based on financial information. It is much more difficult to also create synergy between countries, to share information on customers, to Build Solutions based on the same Components... Enterprises have difficulty in organizing synergy without creating bureaucracy and increased complexity.

For each activity of the Enterprise **Solutions** exist. For example, an Enterprise Operates a Human Resource Solution, an Accounting Solution, a CRM Solution... A Solution Model is not limited to an

Application Software, it also includes the Business Model and **groups Actor Roles, Processes and Software** for efficient Operations.

Foundation is defined as **Reusable Models**. It ranges from a common Business Language, to Software pieces or Enterprise Maps, to Customer Information Model or IT infrastructure Model. In the next chapter we will define more precisely a classification of Foundation elements to help the reader identify what could be Reused.

Foundation plays an important role in reducing complexity, increasing agility and managing synergy. It **reduces Complexity** because it structures the Solutions into independent pieces: software volumes can decrease sharply with Reuse: we will give some examples of high Reuse case studies: it is feasible to get 90% of a new specific Solution ready made through pre-Built Components!

It **increases agility** not only because there is less to do, but also because powerful Foundation automatically generates an elegant Solution structure which will be easy to modify and maintain. Moreover Foundation elements have already been tested and deployed so that testing and deployment are usually faster and less risky.

It helps **manage Synergy** because Foundation not only means Reuse of Functions, but also sharing of Customer information, transfer of good Practices inside Groups of several Companies, and enhanced staff mobility from one position to another one.

We will return to **Value** for Foundation in the 3rd chapter.

See article on SOA and Reuse from Martin Creaner, President, TM Forum at

<http://www.tmforum.org/TMForumIssue37/7826/home.html#feature?ctr=27110990>

1.3 Many questions

All our Sponsors want to improve their Foundation.

After the years of decentralization, autonomy, independence, of “small is beautiful”, come the years of Synergy. Enterprises are now under pressure to realize Synergy: economies of scale, cross-selling, reusing best practices between countries...

But it is not an easy task to organize synergy and few documents have been written which give a global vision of this topic.

This is why Sponsors have asked the CEISAR to produce this white paper.

We classified the main questions into 4 categories:

- **What** is Foundation?
- How to **decide** Foundation?
- How to **build** good Foundation?
- How to efficiently **use** Foundation?

(See details in Exhibit “Questions on Foundations”).

The topic turned out to be so large that CEISAR could not cover all aspects in a period of 6 months.

So, we asked our sponsors to narrow the scope.

They finally chose the following questions:

- **What is foundation?** Definition of Foundation and classification of its content
- What is the **value** of Foundations for an Enterprise; how to measure it?
 - Where does it **really work** today?
- What are the **conditions for success**?
- How do current teams efficiently **use foundation**: which governance and change management are required?
- What does Foundation become when choosing an external **package**?

With that limited list of questions, we nevertheless had the opportunity to address many of the others.

1.4 Contents of this White Paper

1.4.1 Foundation definition and classification and what works today

Mutualization is larger than Foundation: we must give a precise definition of Foundation: Operation Foundation and Transformation Foundation.

We will also classify Foundation elements so that it is easier to compare offers from Providers and present Foundation status in existing Companies.

Using our Foundation classification, we will illustrate what is commonly implemented and really working today in businesses.

1.4.2 Foundation Value and case studies

We will discuss what benefits Foundation brings, and we will try to illustrate via some examples the value obtained by some Enterprises.

Then we will discuss who has to be convinced and how to convince top management.

1.4.3 Success Conditions for a good Foundation

Once a Foundation has been decided upon, once budget and governance are there, how do we get a good Foundation?

Do we Buy or build a Foundation?

How to prepare a realistic planning?

Which are the main difficulties?

1.4.4 How to efficiently use Foundation?

Once the Foundation is available, it must be used. If a good Foundation is not properly used, it is worse than having no Foundation at all.

How to ensure that the Foundation is efficiently used by its customers: the Solution Builders?

1.4.5 Coexistence of Packages and Enterprise Foundation

Enterprises Operate 2 kinds of Solutions:

- Commodity Solutions provided by Package providers.
- Evolutive Solutions built with Enterprise Foundation

How can these 2 worlds coexist?

1.4.6 An example of Building Foundation

To illustrate what Building Foundation is, we will describe a Solution Package Built for Insurance based on a powerful Building Foundation.

Main message will be: "Foundation is not a side topic".

Good Foundation can reduce efforts to Build and Deploy new Solution Models by three. Foundation can represent a huge competitive advantage for an Enterprise in terms of time to market and cost.

It can also be an efficient way of creating synergy in large groups.

But it is difficult to achieve. It requires:

- long term strategy
- top management involvement
- new organization and new governance for Transformation teams
- selecting competent Business Actors and highly skilled IT Architects for Foundation teams
- experience in Foundation Architecture
- budgets to Build or Buy/Customize Foundation
- new Transformation Approaches

You can proceed gradually or rapidly according to your strategy and your budget.

You can use Exchange Foundation only to begin with, or also use Building Foundation.

You can deploy it on a limited number of Solutions.

But do not ignore Foundation.

2 Synergy is Sharing Resources and Reusing Models

2.1 Why is Centralization so difficult?

Enterprises must define what is **centralized** or **decentralized**.

They navigate between

- **more centralization** for visibility, scale economies, consistent information on customer or management, exchange of good practices or good products, move of people between Companies
- **more decentralization** for people motivation, proximity with the customer, fast decisions, smaller Solutions, less bureaucracy, subsidiarity

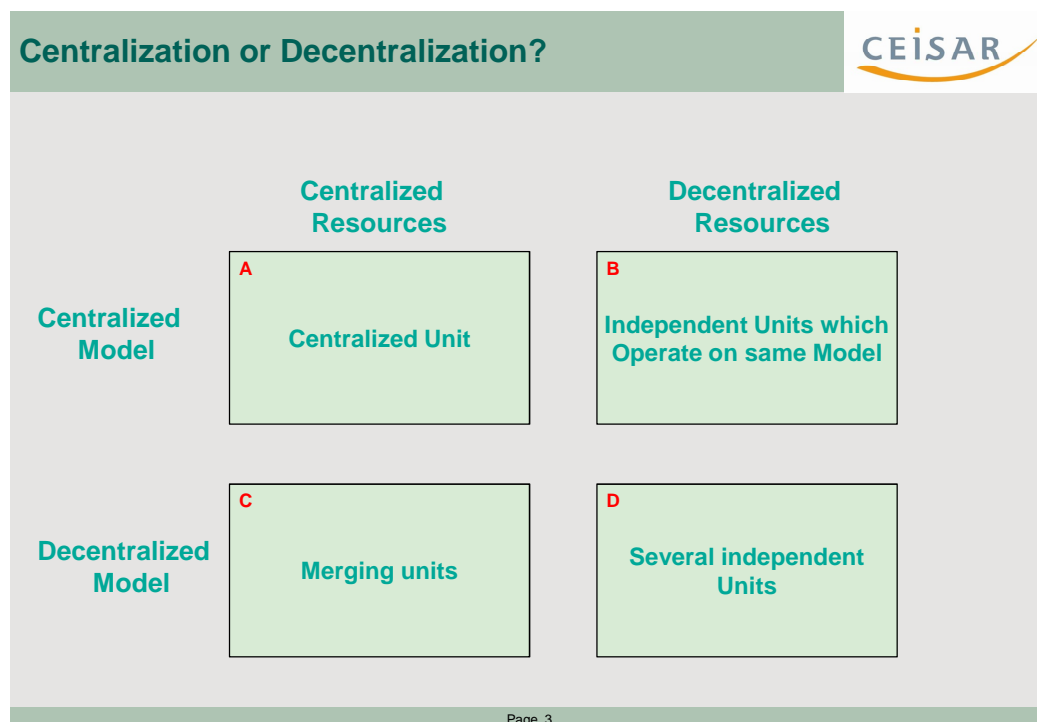
Difficulties arise with Centralization.

Questions

- Can we define a unique Model to Build new Products at Group level, and decentralize Product Design at Company Level?
- Can we define a unique Model to manage Human resources and decentralize Human resource management?
- Can we operate a centralized Call center which works for different Business Units?
- What is the advantage to centralize IT Operations while Models are different?
- How to centralize Process Models which adapt to each Company organization?
- How to centralize Solution Models which adapt to each Company specificities?
- What is the role of a centralized Architecture team
- How Software Packages and Solutions built with an Enterprise Foundation can coexist?

The difficulty comes from the fact that the single question “what to centralize or decentralize in the Enterprise” is **not the good question**: there are 2 questions and not 1

- centralize or decentralize **Models** (the Transformation): Information Model, Process Models, Function Models, Roles
- centralize or decentralize **Resources** (the Operations): Human Actors, Computer Actors, Information,



Example:

A Procurement, Call center, IT Operations

B HR, Branches

C Merge Units which Operate on different Models: happens when 2 companies merge; unstable, requires convergence on Models

D Independent Business Units: the Group follows financial information

You can for example:

For Human Resources

- Transformation: **centralize** Human Resources Model and define Processes, Functions such “How to hire new employees”..., “How to evaluate performance”,
- Operations: **decentralize** Execution of the Model.: team who manage Human resources are decentralized

For Marketing

- Transformation:
 - **centralize** “How to design a new Product” at Group level
 - **decentralize** “Build a new Product” at Company level
- Operations: **decentralize** sales of the Product at Branch level

For IT Operations

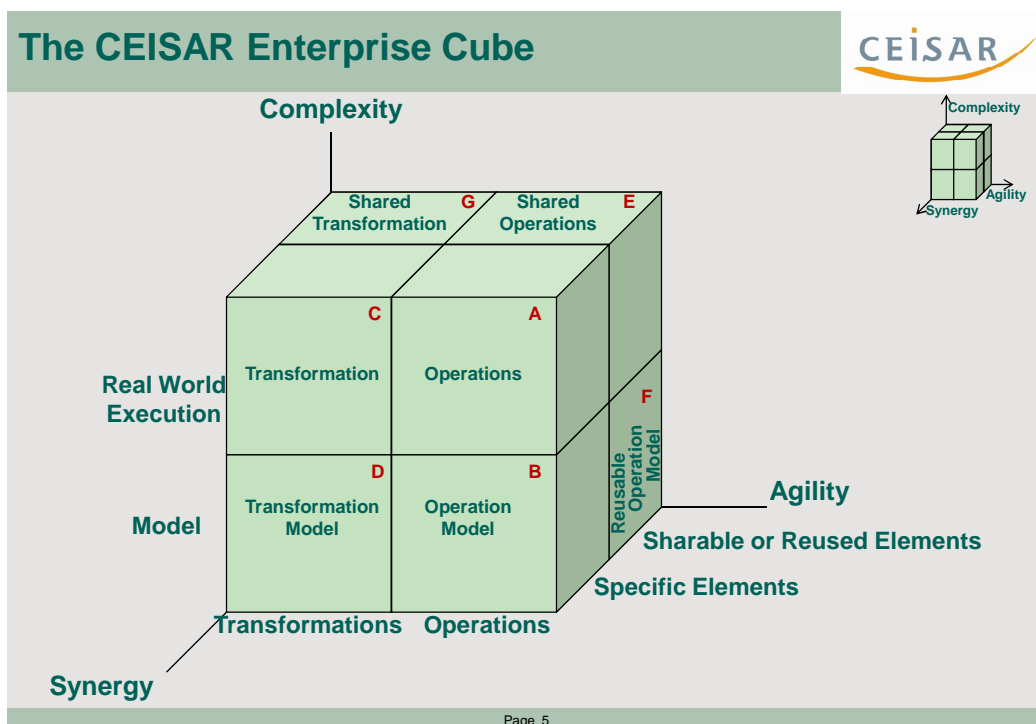
- Transformation: **centralize** definition of the IT Configuration Model : OS, Hardware, middleware
- Operations: **centralize** teams who manage IT Operations

Splitting Reusing Models and Sharing Resources will offer more adapted scenarios than just centralizing everything or decentralizing everything.

2.2 Synergy is Reusing Models and Sharing Resources?

We needed to classify different forms of Synergy to be able to exchange between Sponsors or with Providers.

The CEISAR Cube helped us to define 2 categories of Synergy: **Reuse Model** and **Share Resources**. Let's quickly recap on how the CEISAR represents an Enterprise:



In keeping with the key concerns (Complexity, Agility and Synergy) of large Enterprises, the CEISAR presents the Enterprise as a Cube:

- to reduce **Complexity**, we must formalize how the Enterprise works: we split “**real World Execution**” from its “**Model**” which represents this formalization: Roles, Processes, Software, Information Model
- to increase **Agility**, we must split **Operations** which is running the Enterprise every day, from **Transformation** which means Building or Modifying the Model on which Operations are executed. Operations is producing, selling, supporting Customers while Transformation means “Projects” to prepare new Models: new Processes, new Products, new partnerships
- to develop **Synergy**, we must define what is **Shared** in real world execution (like Teams, Information, IT Operations) and which Models are **Reused** (like Roles, Components, Solutions, Information Models)

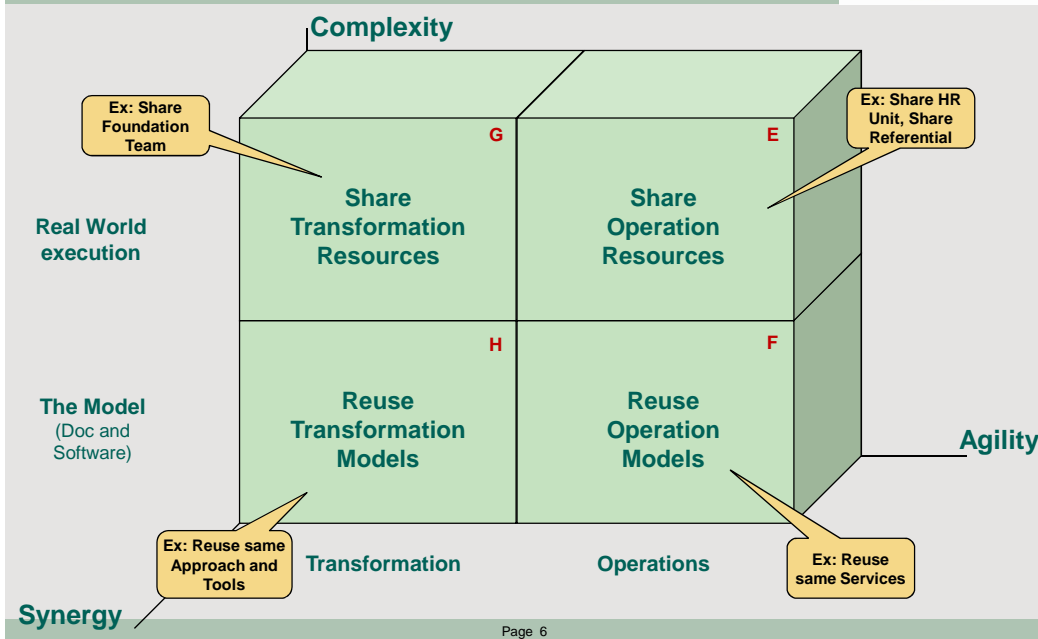
In this white paper we focus on Synergy which is represented by the 4 cubes behind (E, F, G, and H). Synergy is not only Reusing Models, it is also Sharing Resources.

F and H are “**Reusable Models**”: they represent the **Foundation**

- F is **Operation Foundation**; it includes all Model elements Reused in Operations:
 - **Operation Components** to Build internal Solutions,
 - **pre-Built Solution Models** (provided by Package providers or by a Group Unit working for several Companies of the Group)
- H is **Transformation Foundation**: it includes all Models elements Reused in Transformation like:
 - Project Approach
 - Tools Reused to Build Solution Models and Operation Foundation.

E and G are “**Shared Resources**” between Solutions

- E includes Shared Resources for Operations such as
 - Shared Units (like a centralized HR Unit which Operates for all Companies of a Group),
 - Shared information (like a centralized Customer referential which is accessed by all Solutions),
 - Shared IT Operation Units (like a centralized IT Operations Center which runs Solutions for different Companies of a Group)
- G includes Shared Resources for Transformation such as
 - Shared Foundation Team (like a centralized Architecture team which works for different Companies)
 - Shared repository of Components



Page 6

(See more detailed Cube in exhibit).

2.3 Reusing first or Sharing first?

Reuse then Share

Foundation is essential to Sharing Resources:

- All Companies of a Group cannot **Share** Customer Information in a single referential, if they do not agree to first **Reuse** the same Customer Information Model: What is a Customer? Which identifier? Which Attributes to define him?
- We cannot **Share** the same HR Unit between different Companies of a Group if we do not **Reuse** the same HR Solution Model.

This is why it is more efficient to first Reuse the same Model. Sharing Resources will come after if really necessary.

For example:

- Companies could Reuse the same Customer Information Model, but not Share the Customer Repository: each Customer file remains private inside each Company of a Group
- Companies of a Group could Reuse the same HR Model without Sharing the HR Department

In some decentralized Groups, it has been decided to **first centralize different Units** working differently because they apply different Models.

Then, as all Actors are under the same responsibility, it seems easier to Build and progressively deploy the same Model in the new Unit.

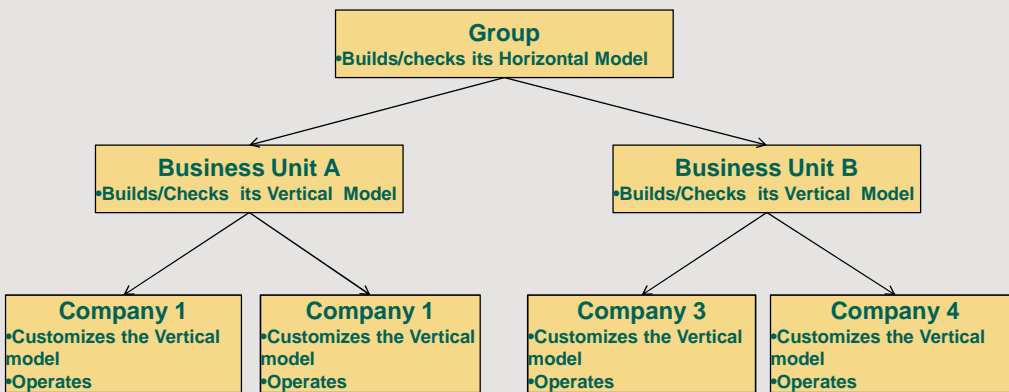
The exhibit "Sharing is not Reusing" details different combinations of Solutions.

For example, the following scenario describes the key principles in organizing a **Group** made up of different **Business Lines**: each Business Line owns **Companies** in each Country.

- Transformation: Centralize Building of Models
 - **Models for horizontal activities** such as Human Resources, Legal Accounting, Procurement, CRM are centralized at Group level: they can be customized at Company level if necessary
 - **Models for vertical Activities** specific to each Company are centralized at Business Line level: but each Company should be able to customize this Model
- Operations: Decentralize Execution of Models
 - all Operations for Horizontal or Vertical Activities are **executed by each Company**

- the role of the Group Managers is to Build Horizontal Models and check that they are applied
- the Role of the Business Line Managers is to Build the Vertical Models and check that they are applied by Companies
- Centralize some Units if necessary
 - for better negotiation, part of procurement activities are centralized at group or Business Unit levels
 - for lack of experts, legal Operations are centralized at group or Business Unit levels

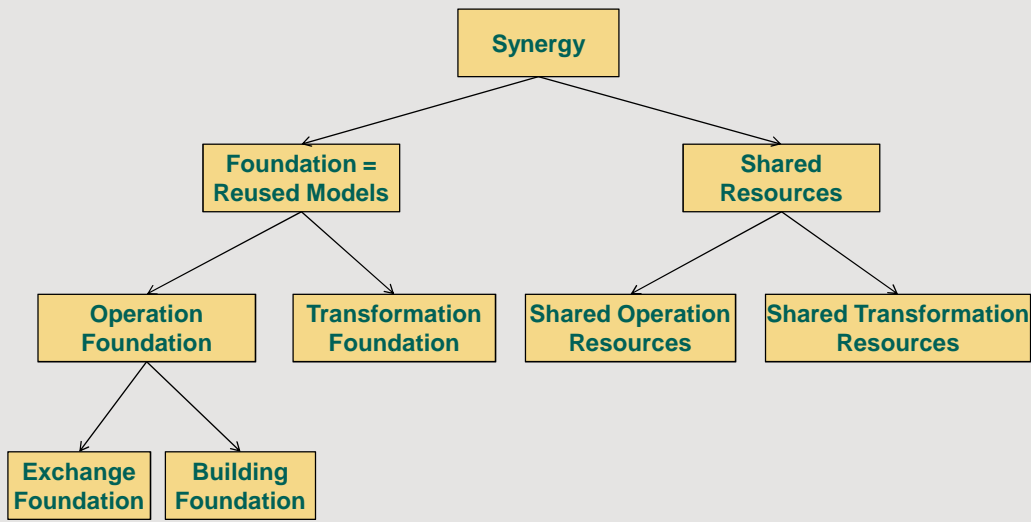
Centralization and Decentralization



Page 4

Many other scenarios may exist.

Now we will concentrate on **Foundations** which group the **Reusable Models: Operation Foundation** (decomposed into Black and Building Foundation) first, then **Transformation Foundation**. Those interested in **Sharing** can read the Exhibits “Sharing Operation Resources” and “Sharing Transformation Resources”.



3 What is Operation Foundation?

How did we identify Reusable Components?

We met with:

- **Sponsors:** Air France, Axa, Total and some other **large users**
- **large Suppliers:** Google, IBM, Microsoft, Oracle, SAP and **smaller Suppliers:** Orchestra Networks, SalesForce, Softeam, Wyde

Individual answers are delivered to our Sponsors. In this white paper, rather than giving individual summaries, we prefer to present a global overview by topic.

After analyzing these reusable Components, we finally classified Reusable Functions according to 2 dimensions:

- Components are customizable (White components) or not (Black Functions)
- Reusable Functions are Built with the same Transformation tools as Solutions

Box A welcomes SOA Functions and requires Middleware.

Box B is adapted to Solution Builders who have the right to customize the called Functions (not frequent)

Box C represents libraries of non-customizable Functions which are linked as libraries provided by the Technology providers. Enterprises have always reused these Libraries.

Box D represents the Functions which can be customized with the same Transformation tools. This is a very powerful way of increasing Reusability.

Categories of Reusable Functions		CEISAR	
	Black Function = « Call »	White Function = « Build from »	
Exchange Foundation (Use different Transformation Tools for Foundation and Solutions)	A Interoperability Ex: SOA Business Functions	B Customizable Functions Ex: Functions provided by Component Suppliers which can be Customized	
Building Foundation (Reuse same Transformation Tools for Foundation and Solutions)	C Libraries of Functions Ex: UI Libraries or Data Access Functions (from Oracle or Microsoft or IBM...)	D Specialization Ex: Inherited Business Objects	

Page 9

TOGAF uses similar concept called "Building Block" which is considered more for Exchange Foundation than Building Foundation : see <http://www.opengroup.org/architecture/togaf9-doc/arch/chap37.html>

We decided to call "Exchange Foundation" all elements required for interoperability between Solutions: Solutions can be Built with different Transformation tools, but they exchange. It brings unicity of information, good structure, independence and decreases complexity. Example: call the Security Function.

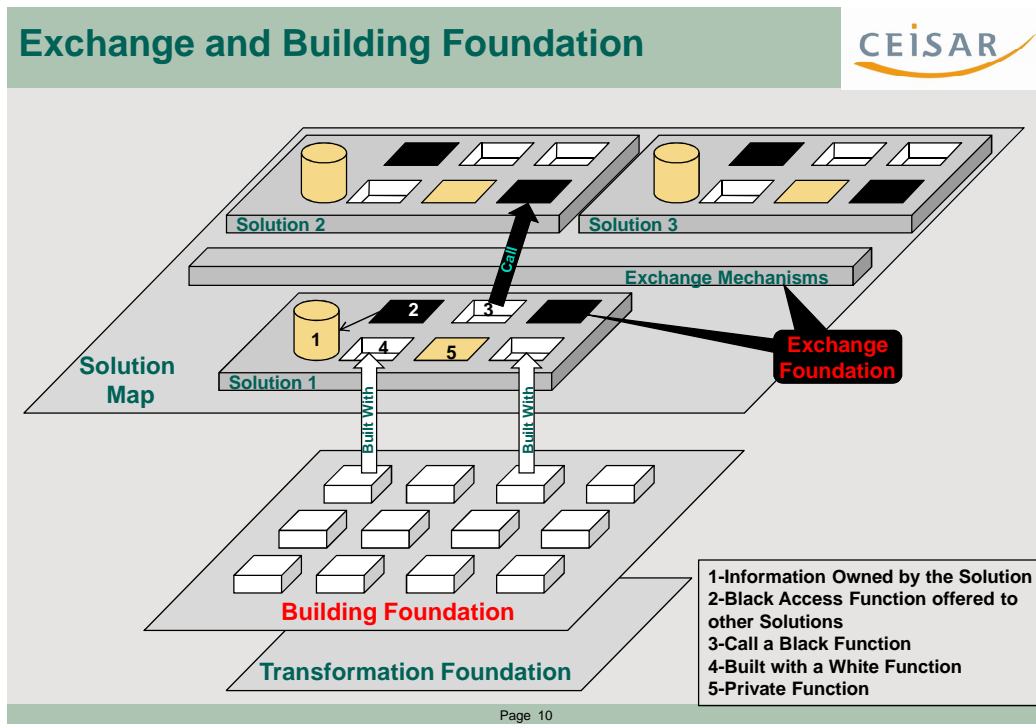
"Building Foundation" gathers all possible Reusable elements when Solution Models are Built with the same Transformation tools. It brings Agility, unique User Interface and decreases complexity if it is well Built (see below). Example: Life Insurance Contract is a specialization of Insurance Contract.

All enterprises should have an Exchange Foundation.

Enterprises for which Agility is strategic should have a Building Foundation.
 Before detailing Exchange and Building Foundations, we will discuss Reuse and Sharing.

Package Providers continuously improve their Foundation to reduce the complexity of their offer and increase its capacity to adapt the package to Customer needs.
 Buying a Solution Package means importing its Foundation; coexistence of **Package Foundation** and internal **Enterprise Foundation** is one of the main difficulties facing Enterprise today: we will come back to it in the last chapter.

3.1 Exchange and Building Foundation



A **Solution Model** formalizes a set of Processes and/or Functions: it formalizes Human procedure, software, information, into a Solution Model which not only contains Software but also all Human Documentation useful to execute Business activity.
 An **Enterprise Architecture** is made up of Solution Models: CRM Solution Model, HR Solution Model, Pricing Solution Model... It is represented by the “**Solution Map**”.
 Each Solution Model owns its Information, represented by the yellow cylinder on the slide.
 Each Solution Model may be executed one or several times. For example the same HR Model may be executed in each of the 10 Companies of a Group: in this case there are 10 **executed Solutions** for 1 **Solution Model**. We focus on Solution Models.

3.1.1 Exchange Foundation

When Solution Models are independent, they are easy to manage: decision, execution, evolutions are decided, executed and checked at a decentralized level: no coordination is required. Governance is very simple.
 But it means Information duplication, double Information entry, redoing the same thing several times, and impossibility of exchange...
 So it is necessary that these Solutions **interoperate**. To succeed interoperability, it must be decided which Solution **owns** what **Functions** and which **Information**. For example it can be decided that the Customer file will be owned by the “CRM Solution” and the Function “Am I authorized” will be owned by the “Security Solution”.
 Then Each Solution must provide a “**Black Function**” to allow each other Solution to benefit from its Functions and Information. A Black Function (or SOA Function in IT language) is composed of an **Interface** (or a “Contract”) which defines how to call it and an **Implementation** which defines how it

works. It is important to hide complexity of Implementation to the caller who just knows the Interface and does not know how it is implemented: this is why we call it a “Black Function” like a Black Box.

*Note that it is a **Black Box** for the **Caller**, but not for the **Called Solution** which must Build it.*

*The same Solution can be together a **Called Solution** when it offers Black Functions to other Solutions or a **Caller Solution** when it asks for a Black Function to other Solutions.*

To give access to its own information the Called Solution proposes **Information Access Function** and does not authorize direct access to its Information: it allows changing of Information Model without changing Interface and disturbing all the Caller Solutions.

In the end, each Solution only offers Black Functions to external Solutions: some to access information, some to execute a specific Business Function, some to feed Solution with input Information.

The set of Black Functions coming from all Solutions is called **Exchange Foundation**.

Exchange Foundation ensures interoperability: no double data entry, unicity of Information, offer of Reusable Functions. Interoperability simplifies Enterprise Architecture and reduces the work required to Build a new Solution Model.

Each Solution can be Built with **different Transformation tools** as long as it can call Black Functions and offer new Black Functions to the community. It requires that a Foundation team defines for all:

- Which Solutions **own** what
- Which **Reusable Information Model** must be respected by all Black Functions so that Solutions understand each other: Business Concepts, identifiers, Attributes, types
- Which **repository** to help Solution Builders to identify useful Black Functions
- Which **IT Infrastructure** to intercommunicate (network, Middleware)

The Foundation team does not implement the Black Functions: it is done by each Solution team.

3.1.2 Building Foundation

Once Exchange Foundation is efficient you can go further and Reuse Building Foundation.

There are 2 forms of Reuse: Reuse by composition and Reuse by specialization.

Reuse by Composition means that you assemble your Solution Model by calling external Black Functions as we just discussed.

Reuse by Specialization is Building a Model part which looks like another one. This is called **Specialization** in the Object Oriented Approach: a Damage Insurance Contract inherits from an Insurance Contract which in turn inherits from a Contract:

- “a Contract” has attributes such as: Product, Subscriber, date of subscription, account for debit
- “an Insurance Contract” adds: you are only allowed to use Insurance Products and not all kinds of Products, Beneficiaries...
- a “damage insurance contract” adds: the Good to insure

You can specialize not only Information, but also Functions, User Interface, Types...

The system becomes efficient if a change at a father level automatically benefits the cascade of sons who inherited from the father.

White components can be:

- **UI Components** to Build UI interface
- **Information Access Mechanisms** to Build Information Access Functions
- **Type Definitions** and Functions
- **Business and Organization Objects** from which it is possible to specialize new ones
- **Process Components** to Build new Processes

3.1.3 Benefits are not the same for Exchange and Building Foundations?

Exchange Foundation



Visibility

Information Unicity
(Single Data Entry,
Shared Information)

Modularity

Building Foundation



Agility,
Time to market

User Interface
Consistency

Simplification

Page 11

Exchange Foundation allows Solutions to Interoperate, benefits are important:

- **Visibility** of Enterprise Architecture: current and future Solution Maps help to understand the Enterprise Architecture, to align Solutions to strategy, to manage a project portfolio
- **Single Information:** exchanges between Solutions allow defining of which Solution owns which Information. It is then possible through Exchanges, to query or to update Information owned by another Solution. Information is consistent and there is no more duplicate data entry.
- **Modularity:** it is possible to specialize Solutions by Functional domains: Accounting Solution, Business Intelligence Solution, Call Center Solution and concentrate specialized knowledge by Solution.

Building Foundation allows us to Build or Modify Solution Models with same Transformation Tools to take advantage of pre-Built components. It brings complementary benefits:

- **Agility** and time to market: gains can be much higher than with just Exchange Foundation. You can divide by 3 time and money for new Solutions if Foundation is powerful (see below)
- **User Interface Consistency:** all Solutions may be Operated with same User Interface, which reduces deployment efforts, increases productivity and facilitates staff mobility in the Organization
- **Global simplification:** the size of the Enterprise Model (easy to measure with Software) is dramatically reduced when Reuse rate is high.
- **Lower risk** : the more pre-tested software you use, the less risk you run of generating defects in your solution (it is also an advantage of Exchange Foundation but at a lower level)

3.1.4 What differences and similarities?

White Functions represent a different kind of Reuse than Black Functions:

- this is not “at Operation time, Solution calls a Model element provided by someone else”,
- this is “at Transformation time, the Transformer Builds Solution Model from Solution parts”: these Solution parts will be called “**White Components**”.

White Components are useful not only to Build Solution Models but also to Build Black Functions.

Reuse of White Functions requires Reuse of the **same Transformation Tools**, which is not true for Black Functions.

All Enterprises **must have an Exchange Foundation** to allow interoperability, while some Enterprises should have a Building Foundation - mainly those for which **Agility** is strategic. The choice is not between Exchange or Building Foundation, it is between “Exchange Foundation only” or “Exchange Foundation and Building Foundation”.

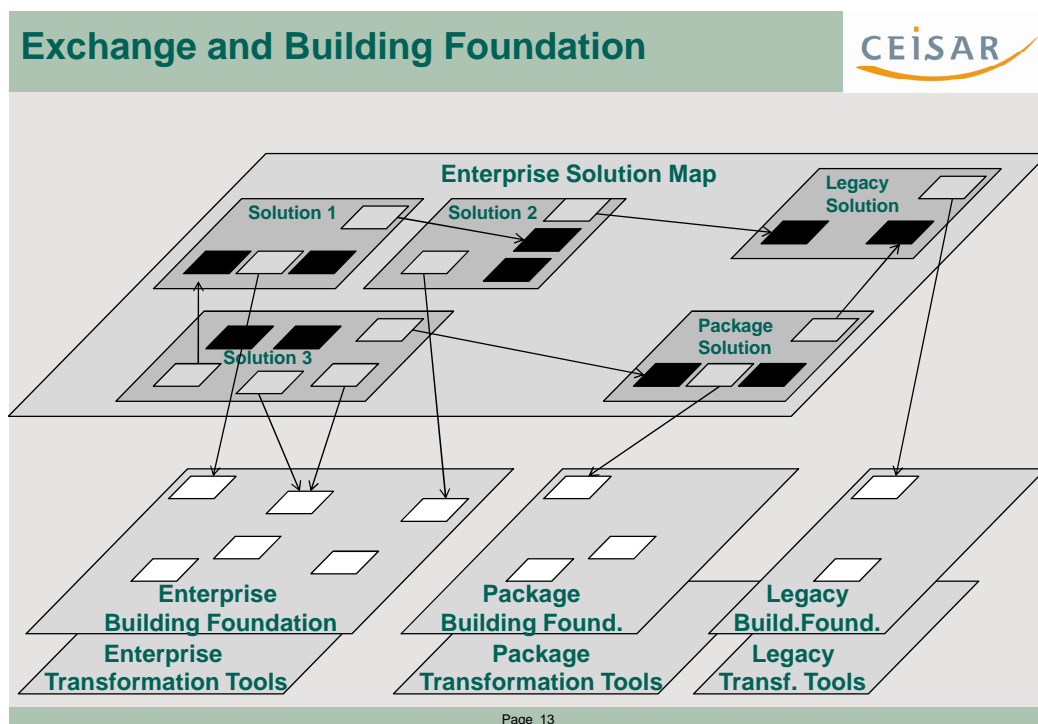
Remark: in France, the Approach for creating an Exchange Foundation is called “Urbanism”.

Governance of Building Foundation is more intrusive, as you impose Transformation Tools. **Exchange Foundation** can be used **progressively** by Solutions, while **Building Foundation** can only be Reused when a Solution Model must be created or deeply renewed.

Reuse is much more important for Exchange + Building Foundations, than for Exchange Foundation only: some attain an **80% Reuse rate**, which means that only 20% of the new Solution Model is to be Built.

A good strategy is to **start with Exchange Foundation**, because everything that is done for Exchange Foundation will be useful for Building Foundation: creation of the Foundation team, definition of a Reusable information Model, first level of Governance to respect Foundation... are all first steps for future Building Foundation.

Enterprises who choose Exchange **and** Building Foundation will keep **Packages for commodity Solutions**. Each Package comes with its own Foundation. It means that they will use Exchange Foundation for all Solutions, but **Building Foundation only for Evolving Solutions**.



In this example:

- an Enterprise has decided to Build Exchange and Building Foundation
- the Enterprise executes 3 kind of Solutions:
 - New Solutions (1,2,3) Built with Enterprise Building Foundation
 - Package Solutions Built with Package Building Foundation
 - Old Legacy Solutions which could benefit from limited Legacy Building Foundation
- there is a single Exchange Foundation for all Solutions, but several Building Foundations

3.1.5 Why “Black” and “White”?

The OMG recommends the use of these terms "Black" and "White"; they even go further with Clear-box assets and Gray-Box assets.

The variability and visibility of an asset is another key property of an asset. At the one extreme an asset can be invariable, that is it cannot be altered in any significant way. This is often the case for assets that are component binaries. Assets at this end of the spectrum are sometimes called **black-box assets**, since their internals cannot be seen and are **not modifiable**. At the other end of the spectrum are **white-box assets**. These assets are created with the expectation that asset consumers will edit and alter its implementation. White-box assets also typically include development artifacts such as requirements, models, build files, etc. Two other variations in between are clear-box assets and gray-box assets. **Clear-box** assets expose implementation details (via models, code fragments, or other documentation), however they cannot be modified. These details are exposed solely to help the consumer better understand the inner workings of the asset, so that the consumer can use the asset more efficiently. **Gray-box** assets expose and allow modification only to a subset of the asset's artifacts, usually through the parameters on the asset.

A **Black Function** is Built in 2 parts: Interface and Implementation.

The Solution calls the Black Function through the **Interface** (or the "**Contract**") and does not need to understand Implementation: this is why it is called "Black" like a black box.

A **White Function** is visible by the Solution Builder through the Transformation Tools: he can specialize it.

A **Clear-box Function** is a Black Function which allows its implementation to be read, but not changed.

A **Gray-Box Function** is a White Function which is customizable only by Configuration (and not software development).

3.2 Which elements are in Operation Foundation?

We propose now to list what is inside this Operation Foundation.

*If some readers are interested in going into more detail, they can read the Exhibit "**Detailed Operation Foundation**".*

If you Build a new specific Solution Model because you want, for example, to support the launch of a new specific Product, synergy may be obtained in 2 ways:

- you decide to **Reuse a pre-Built Solution Model** (like an Application Package provided by a Software editor, or a Solution Model Built by a Group for its different Companies) and **customize** it
- you decide to Build a new Specific Solution Model **Reusing Components** like GUI component, information access, business component, organization component or Solution Structure.

To Build a Specific Solution based on Reusable Components, you first must Build these Components which in turn may be Built by reusing other Components: for example the Security Component could Reuse Information Access Functions or UI Functions.

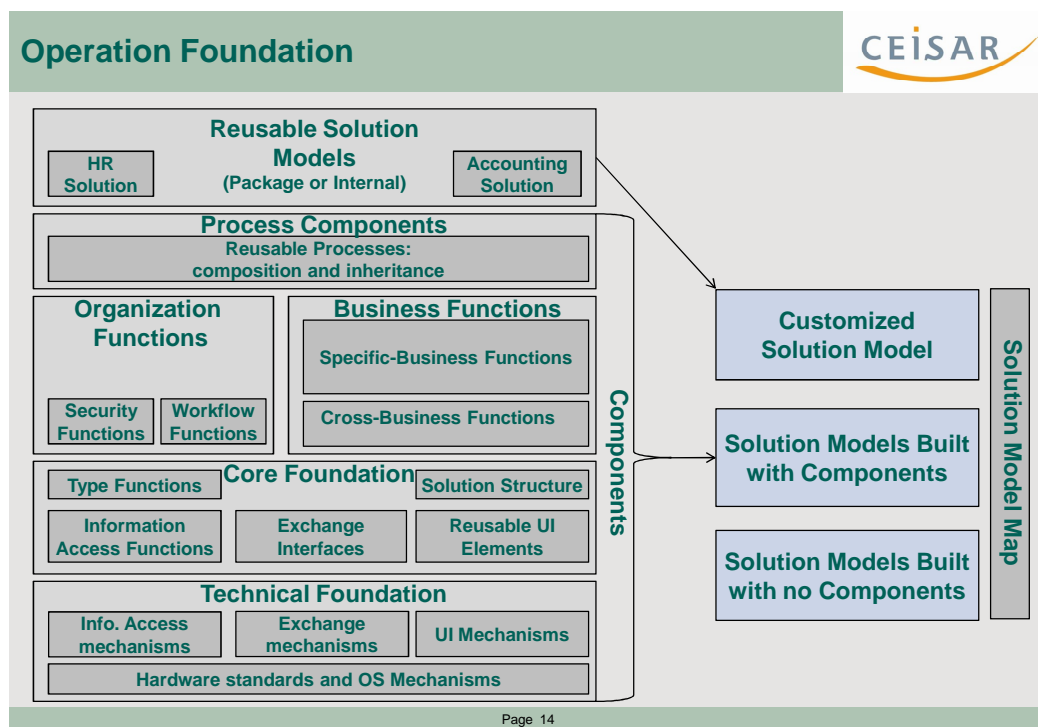
Togaf defines that "Components" or "**Building blocks**" must have generic characteristics as follows:

- A building block is a package of functionality defined to meet the **business needs** across an organization.
- A building block has a **defined boundary** and is generally recognizable as "a thing" by domain experts.
- A building block may **interoperate** with other, inter-dependent, building blocks.
- A good building block has the following characteristics:
 - It considers **implementation** and **usage**, and **evolves** to exploit technology and standards.
 - It may be **assembled from other building blocks** and It may be a **subassembly of other building blocks**: it means that we must not manage a flat list of Building Blocks but a structure of Building Blocks which Reuse each other
 - Ideally a building block is **re-usable** and replaceable, and well specified.
- A building block has a **type** that corresponds to the TOGAF content **metamodel** such as actor, business service, application, or data entity (this is why we include Reusable Information Model, Reusable Functions, Reusable IT Configurations... in Operation Foundation).

OMG prefers the term **Reusable Assets** (see <http://www.omg.org/cgi-bin/doc?formal/2005-11-02>) .

(for other Frameworks see : <http://www.pragmaticea.com/>)

We tried to classify all elements described by the different Standardization organizations into a single breakdown.



We propose to describe the different layers in a **bottom-up** and not a top-down order, because upper Components Reuse lower Components: we must understand the lower Components to understand how upper Components are Built.

3.2.1 Reuse Human Actor Roles for Operations

(For simplification purposes, this element of Operation Foundation is not represented in the slide).

Human Actors are employees, partners, prospects... Each of them occupies a **Position** in the Enterprise Organization. Each Position is located in the Organization chart and plays a **Role** like "Sales Man", "Assistant", "Branch manager"...

The same **Role definition** can be **Reused** for different Positions. It will help Solution Builders to assign Activities to Actors (**Workflow**) and to define Profiles of Rights and Duties (**Security**).

Remark: Organization chart is not part of Model, it represents real life, with real Actors.

What works today?

Most Enterprises have **well defined Operation Roles**.

Sometimes, Roles are too numerous which prevents the assigning of a larger set of Activities to Actors. If Foundation includes Reuse of UI elements and allows for standardization of user interface, if Foundation allows access to Information from everywhere, then each Actor will be able to do more because the effort to switch from one Activity to another is lighter: **"the more UI Foundation, the fewer Roles"**.

3.2.2 Reuse hardware standards and OS mechanisms

Same IT configurations can be Reused by different Solutions:

- one or a limited number of Operating Systems
- Hardware Model: Reuse one standard Work Station, one type of Server
- Network Model: Reuse same protocols for information exchanges (data or voice)

Remark: this is just Model description and not the description of all IT infrastructure (CMDB in ITIL terms) which is described in "Shared IT Infrastructure".

What works today?

They exist for most Enterprises.

3.2.3 Reuse Access mechanisms to Information

Each Information Access Function must be Built by Reusing powerful **Access Mechanisms** which lighten tasks such as: access **Business Objects** rather than tables, manage **Versioning**, allow **dynamic attributes**... (see exhibit for more details).

3.2.4 Reuse Information Model and Access Functions to Information

Some Information Models can be reused by the different Solutions. It enables us to:

- define exchange formats between Solutions (Black Functions) and **save time** when Building Solutions
- **Share Information** (see exhibit "Sharing Operation Resource").

Before Reusing Access Mechanisms to Build Reusable Access Functions, the Enterprise must define its **Information Model**.

The Information Model is broken down into **2 parts**: Information Structure and Attributes

- the **Information Structure** includes a **Business Glossary**, **Business concepts**, **identifiers**, **Relations** between these Objects (Ex: an Insurance Contract Relates to the Customer, the Product, the Account...), **inheritance** between Objects (ex: a Life Insurance Contract inherits from an Insurance Contract). The **Business Glossary** describes main **Business Entities** and their relations: it creates a common business language and helps define precise specifications. It is completed by **Maps for Entities**, also called "Entity Relation Model": which describes main Business Entities, their relations and inheritance. This is the first Map to build.
- Once the structure is ready, it can be progressively filled with **Attributes**, according to needs coming from Functions.

Note that the Information structure is key: changes of concepts, or Relation cardinalities are expensive, while adding Attributes to a stabilized Information Structure is not so difficult.

Industry Information **standards** must be used as much as possible: each industry is defining its Information Model. Take advantage of these standards when defining your own Information model (see exhibit for more details).

Enterprise must also define **Reusable Types**: how to represent a date, a name, an amount, a text... It defines how to present the Type to the Human Actor and how to store the Type for the Computer Actor. Types are not considered as important topics in most Organizations: people have not understood that the huge cost of "year 2000" or "euro conversions" were mainly linked to absence of Reusable Types in organizations.

Once

- Access Mechanisms are available
- The Information Model is defined
- Types are defined

it becomes possible to Build Functions to access Information.

The Operation Foundation includes **Reusable Information Access Functions** to Actors, Addresses, Accounts, Products, Contracts, Organization...

Comments on what really works today:

- Very few Enterprises have defined their business glossary, their Business Entity relation Model, their Types; many have defined Table contents for Shared Repositories
- Reusable Access Functions to Shared Repositories (like customers, organization, directory, product catalog) have been Built.

3.2.5 Reuse Exchange Mechanisms between Solutions

Solutions **exchange** with other Solutions: synchronous or asynchronous, for read Information or write Information or feed inputs or execute Functions.

The exchange can be executed using a Middleware which converts information (using XML, PDF, flat file, mapping...), finds the Target Solution and sends. It can be also executed by a simple call inside the same executable (like a DLL).

The Solution Builder must call the external Function without knowing if a middleware will or will not be used.

The choice is made by the Foundation team which hides complexity of middleware to Solution Builders.

Comments on what really works today:

All Enterprises have well defined Exchange Mechanisms through Middleware definition.

But many Reusable Functions do not require Middleware: you do not Build a Web Service to check Date validity, you generally use a much lighter mechanism.

Function Interface must be independent from exchange mechanism.

3.2.6 Reuse Exchange Functions (the “Adapters”)

Exchange Mechanisms allow us to Build **Interfaces** (or “**Adapters**”) between Solutions. They must be publicized to help Solution builders when they want to connect their Solution to other existing Solutions.

What works today

Most of our **sponsors** provide internal Adapters to help Solutions Builders when they want to connect their Solution to other existing Solutions.

But as they use independent tools to Build Solutions and to document Adapters, some have difficulty in synchronizing updates in Software and in Repositories: some Project Leaders prefer to contact Function Providers directly and not the Foundation team which generates overload: an integrated set of Transformation Tools would help to synchronize Information (see after: Transformation Foundation).

3.2.7 Reuse User Interface Components

Reusing the same consistent and powerful user interface (presentation, navigation) for all Solutions allows more efficiency. It can be done by defining **documented UI policy**. This policy is more easily applied if **UI Components** are offered which implement this policy.

Reusing pre-built UI elements (by composition or inheritance) also helps save time when Building new Solutions.

Comments on what really works today

Look and feel standard **documentation exists** in most Enterprises.

But Reusable **UI Elements do not exist**: User Interface Building is still a specific task with no Reuse. User Interface Building is a very costly Transformation activity; it requires many iterations before end user is satisfied. Improving Reusability in this domain is an important productivity factor.

3.2.8 Reuse Functions related to Types

Each attribute relates to a Type.

The Attributes “Birth date”, “subscription date”, “Value Date” all Reuse the same “Date” Type.

For each Type we can define Rules: check Functions, presentation Functions, internal representation.

Define reusable Types like: Enumerated, Date, Amount, Number, name, text, table, tree, image, video, sound...means that attached Rules are also Reusable.

For example the Date Type should offer

- check Date validity
- compute the number of working days between 2 dates
- present a date in European or US format
- ...

More complex Functions are related to Types like:

- word processing Functions: provide a reusable Text Type which allows embedding of Text Attributes inside Objects and provides Text editing functions to the end user.
- table processing Functions
- Video, Audio processing Functions

All these rules are simple, but they are Built hundred and thousands of times in large systems and contribute to the overall complexity.

Comments on what really works today

Types are not Reused in most Enterprises.

This is mainly due to the absence of powerful Type Mechanisms in most Transformation Tools.

3.2.9 Reuse Solution structure

Different White Components may offer several Implementations:

Solution Structure contains those Implementations that have been chosen for the current Solution.

- Preferred Security Functions
- Preferred UI presentation
- Preferred Desktop
- Preferred Types
- Preferred Language
- ...

Comments on what really works today

Exists in some Packages.

3.2.10 Reuse Organization Functions

These Functions are linked to Organization. They are highly Reusable. They provide help for assignment of Activities (Workflow) and security.

Comments on what really works today:

- Single sign-on is in place or will be in near future in most Enterprises.
- Security Functions: they exist in all Enterprises, but very few Enterprises have a unique Security Function
- Workflow Functions exist in some Solutions, but are never generalized as a unique Reusable Function for all Solutions

3.2.11 Reuse Business Functions

They can be reused as full available Functions (Black Function) or as a skeleton or pattern (White Component) which accelerates Function Building.

Business Functions are split between **Specific Business Functions** which belong to domains such as: Bank, Industry, Insurance, Telecom, Utility..., and **Cross-Business Functions** which are the same for all Business domains.

Comments on what really works today:

- Many Enterprises would like to Build and use Business Functions. **Very few have done it** because it is difficult. SOA is just emerging.
- Success appears to exist in 2 cases:
 - either **Business Information Model is perfectly defined**: then it allows companies like Amadeus to provide Business Services highly Reusable by its customers. Amadeus is progressively increasing its offer by adding new powerful Business Functions: it started with Reservation Functions (“Global Distribution System” or GDS), then proposed Airline internal IT functions such as Inventory or Departure Control Functions (DCS)
 - or **inheritance features** isolate what is Reusable from what is specific

3.2.12 Reuse Process elements

It is possible to build new Processes using Process Patterns or Process composition.

Process Patterns are used when different Processes look like each other. For example “Subscribe a Car Insurance Contract” and “Subscribe a Home Insurance Contract” are different but have a lot in common. Each Process call Functions like: get subscriber Information, select offer, choose options, compute price, bill, compute commission and enforce the Contract. The common part can be Modeled in a Process Pattern, which is the base on which each specific Process is Built.

Process Composition is used when a Process calls other Processes: for example the Process “Welcome a new employee” calls Processes such as “Deliver a security card” or “open a new email account”.

What works today?

Process patterns are not used by Enterprises.

Process Composition is used by some Enterprises: they may compose new Processes by calling existing Sub-Processes.

3.2.13 Reuse Solution Models

Up to now, we have described Components which are Reused to Build new Solution Models.

But, the Solution Model is not always Built internally, it can be an External Solution Model Built by an external team: a Package Provider or a Group team which Builds a Solution Model reusable by the different Companies of the Group.

In both cases, the Reused Solution Model must be **customizable** for each Company: choose currency, information ownership, UI standard presentations, pricing rules, language...

Customization can be executed by **configuration** or by **extension**.

- **Configuration** can be done by parameters, Rule Engines, Workflow Engines: it requires structured skills but not Software developer skills
- **Extension** is done by software development, mainly using inheritance mechanisms; it requires developer skills

Remark: Reusing a Solution Model does not necessarily mean that associated Business Units are centralized. In a Group of 10 Companies, the same HR Solution Model can be Reused by 10 different HR Units which Operate by independently.

What works today? Packages are widely used for Commodity Solutions. They can be used for Evolving Solutions if they have a strong Foundation allowing customization and extensions.

3.2.14 Reuse Solution Map

The **Enterprise Model** formalizes a Global Model for the Enterprise: we then use the word “Map”.

- **Enterprise information Map** has been described above.
- as Solution Model describes Business Processes and application Software, the **Solution Maps** define 2 domains:
 - **Maps for Processes**: describes classification of Processes from Business Domains. This Map focuses on Business Processes independently of current Organization.
 - **Maps for Applications**: describes Solutions Software Model and their exchanges.

Solution Builders Reuse these Maps to

- better understand the Enterprise Architecture
- ease relations between Business Actors and IT Actors
- identify Processes they must handle
- define precise Solution perimeter

There are 3 kinds of Solutions:

- **Specific Solutions** Built **without** Enterprise Components (like Legacy Solutions)
- **Specific Solutions** Built **with** Enterprise components (for new evolutive Solutions)
- **External Solutions** Built with Enterprise Components (if Built internally) or Package Provider Components (for Commodity Solutions).

Maps for Entities are stable Enterprise Models.

Maps for Processes and Maps for Applications shift according to Organization changes and delivery of successive Solutions.

What works today?

Enterprise Information Map exists neither in Enterprises nor with Providers.

Process Map exists for most of our Sponsors, but their Map represents Organization Processes and not Business Processes. Package Solution Suppliers also define this Process Map. It sometimes goes beyond Process level and also describes breakdown of Processes into main Functions.

Application Map exists in most Enterprises. It is sometimes called "Application Cartography" or "Urbanism" in France.

3.2.15 What about Referentials?

Remember that Information Model includes: definition of Entities, Relations between Entities, Attributes and Types.

Reusing the Model means that Reusable Information Access Functions must be provided. Referential Model is one of the first Foundation elements to Build.

"Referentials" generally means: Reuse Information Model **and** Share Information.

For example, a Bank wants to keep an updated file on Stopped Payment Accounts available for all subsidiaries, or a Group wants to share the Profiles of all its Employees, so that they can use any work-station in any Company of the Group.

Sometimes Companies just want to **Reuse Information Model** but not to Share Information. This is the case when the same Solution Model is deployed in different Companies, but each Company keeps its own Information.

Sometimes, it is a **mix**: for the same Business Entity, Companies of the same Group can Share just part of the Information. Let's take an example: for Customer Information

- use the same Customer Model; Information is shared in 2 parts: identification information (name, address, birth date, email, telephone) and comments on Customer
- Share identification Information
- do not Share comments on Customer

Sharing Information requires Rules: who is responsible for updates (the owner), who can access the Information: use the Security Function defined above.

When Information is Shared it can be achieved in 2 ways:

- a single centralized data base to which everyone has access: when real time update has to be offered to users
- replication from a master data base to local data base: when independence of local Operations is required

Replication allows each local Company (or factory) to Operate independently from availability of a central database.

A good example is given by the Oberthur Card System: They have factories to make smart cards in different countries (China, France, US, UK, ...).

They decided that each factory should Reuse the same Product Catalog, but that each factory should Operate locally without being dependent on the availability or performance of a worldwide data base: replication has been deployed. Each time there is a Product modification, the master Solution sends updates (and only updates) to Factories which subscribed to Product Modifications. It has allowed the company to manufacture the same products all over the world.

3.3 Which elements belong to Exchange or Building Foundations?

We now classify these elements into 2 domains:

- Components which allow Solutions to intercommunicate: the Exchange Foundation
- Components which require Reuse of the same Transformation Foundation: the Building Foundation

3.3.1 The Exchange Foundation

Exchange Foundation allows the different Solutions to interoperate: a Solution may access Information owned by another Solution or execute a Function provided by another Solution.

Solutions Models can be bought or Built independently with different Transformation tools. The Foundation team must define a **clear structure** of **Solutions** (breakdown into Domains, Areas...) and **clear Interfaces between Solutions**. The Perimeter of each Solution must be defined so that exchanges between Solutions are minimized: low coupling is required.

It mainly requires what is Black on the slide above:

- **Exchange Mechanisms** (Middleware)
- **Exchange Interfaces** or Adapters based on these Exchange Mechanisms: synchronous question-answers, synchronous updates, Asynchronous feeds (see above)

To define clear Interfaces, the Reusable **Information Model** must be clearly defined: **Business entities, identifiers, attributes, types**. But Access Functions are owned by each Solution; this is why only part of Information Access Functions is blue.

The Reusable Types must be documented: but the specific Transformation Tools of each Solution does not allow us to automatically Reuse **Types**.

Business Functions and **Organization Functions** can be offered to Solution Builders.

When a Process chains sub-Processes modeled in different Solutions, a Workflow engine based in one Solution may call these different **Sub-Processes**.

3.3.2 Building Foundation

Solutions are Built with the same Transformation Foundation: a pre-defined set of tools and Approaches. It allows us to increase Reuse possibilities: Reusable Types, inheritance of pre-built Classes, UI Components, workflow mechanisms replicated in different Solutions for task assignment, Process patterns... If the Foundation is mature, Reuse rate may reach 80%, which means that there is just 20% of work to do to Build a new Solution Model. In return, the Foundation must be powerful and supported (see below).

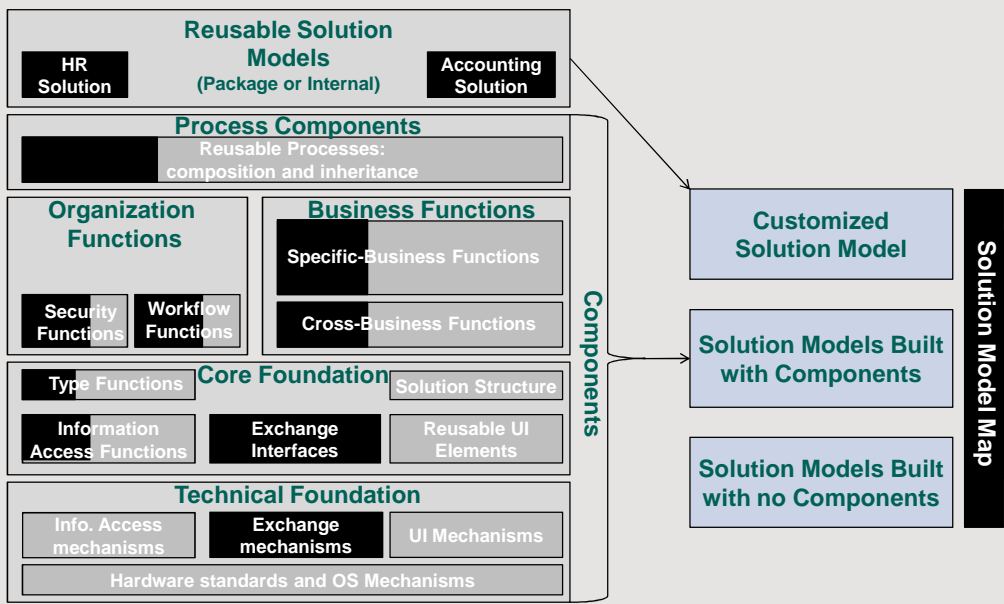
Building Foundation includes:

- **Transformation approach** and **Transformation Tools** are inside Building Foundation but not inside Exchange Foundation, as each Solution can be Built independently; they just need to be able to communicate: the only constraint is that Solution Builders **must Reuse Black Functions** (must be included in all Transformation Approaches) and require a **Repository tool** to retrieve these Black Functions
- **Hardware Standards** and **OS Mechanisms** are not part of Exchange Foundation: each Solution may choose different hardware and OS
- **Exchange Interfaces (Adapters)** and **Mechanisms** must be part of Exchange Foundation
- **Information Access Functions** and **Information Mechanisms** are not part of Exchange Foundation, but Business Entity Model must be part of Exchange Foundation to ease exchanges: same Business Concepts, same Attributes, same relations
- **Reusable UI Elements** and **UI mechanisms** are not part of Exchange Foundation
- **Type Functions** are not part of Exchange Foundation; but Type definition must be part of it to allow exchanges
- Some **Organization Functions** like Security Function or single sign-on are part of Exchange Foundation, but other Organization Functions such as “Assign next Activity to next Actor” or “Manage a To Do list”, or “update Calendar” are not part of Exchange Foundation
- **Business Functions** callable via SOA or other inter-solution mechanisms are part of Exchange Foundations, but Business Functions reusable by inheritance are not.
- **Inheritance of Processes** are only available in Building Foundation.

3.3.3 A mix of both approaches

When Enterprises decide upon a Building Foundation approach, they still prefer to buy Solution Packages for commodity Solutions. These external Solutions import their own Foundation.

It means that these external Solutions will be integrated through Exchange Foundation, while internal Solutions will benefit from full Foundation.



This slide summarizes the main differences between Exchange and Building Foundation:

- **Transformation approach** and **Transformation Tools** are inside Building Foundation but not inside Exchange Foundation, as each Solution can be Built independently; they just need to be able to communicate: the only constraint is that Solution Builders **must Reuse Black Functions** (must be included in all Transformation Approaches) and require a **Repository tool** to retrieve these Black Functions
- **composition and inheritance of Processes** are not part of Exchange Foundation
- Some **Organization Functions** like Security Function or single sign-on are part of Exchange Foundation, but other Organization Functions such as “Assign next Activity to next Actor” or “Manage a To Do list”, or “update Calendar” are not part of Exchange Foundation
- **Business Functions** callable via SOA or other inter-solution mechanisms are part of Exchange Foundations, but Business Functions reusable by inheritance are not.
- **Type Functions** are not part of Exchange Foundation; but Type definition must be part of it to allow exchanges
- **Information Access Functions** and **Information Mechanisms** are not part of Exchange Foundation, but Business Entity Model must be part of Exchange Foundation to ease exchanges: same Business Concepts, same Attributes, same relations
- **Exchange Interfaces (Adapters)** and **Mechanisms** must be part of Exchange Foundation
- **Reusable UI Elements** and **UI mechanisms** are not part of Exchange Foundation
- **Hardware Standards** and **OS Mechanisms** are not part of Exchange Foundation

4 What is Transformation Foundation?

Now, let's define what is inside **Transformation Foundation**.

If some readers are interested in exploring this in more detail, they can read the Exhibit "**Detailed Transformation Foundation**".

If some readers are interested in Transformation **Shared Resources**, they can read the exhibit "**Operation Shared resources**".

Transformation can be defined as Transformation Actors (Human Actors and Computers) executing Transformation Actions (Projects, maintenance) with Transformation Information.

In Transformation Foundation we group:

- Reusable Transformation **Roles**: project Manager, Business Analysts, Developers
- Reusable Transformation **IT Configuration**: hardware, OS and network which are used
- Reusable **Methodologies**, Good practices, and **Tools** to support them
- Reusable **Information Model** for Transformation

All of them can be Reused by the different Companies of a Group or by different teams in a Company.

4.1 Reusable Roles for Transformation

First question is: "Do Enterprises define Reusable Roles?"

Many Roles can be defined for Transformation Purpose like: "Project Leader", "Project Pilot", "Business Analyst", "Designer", "Developer", "Tester", "Business Architect", "Technical Architect". An Enterprise may decide to define a single list of Roles: it will help to describe the precise assignment of Activities to Actors.

Example of Reusable Roles:

- **Sponsor**: the one who defines the Problem, funds the project, make decisions and checks obtained value
- **Project leader**: the one who manages the Solution Building team
- **Business analyst**: the one who defines requirements, tests, accepts, builds user documentation and trains users
- **IT developer**: the one who designs, programs, tests Software
- **Business Foundation Building Actor** (sometimes called "**Business Architect**")
- **IT Foundation Building Actor** (sometimes called "IT Architect")
- **Support Foundation Actor**: trains, coaches, checks Solution Actors

What works today:

Do Enterprises define Reusable Roles?

They all have defined Transformation Roles. But the list of Roles can be **long** (40 different Transformation roles with one of our Sponsors), especially for those who have decomposed the project process into many steps. It appears that number of Roles and number of Transformation Tools are linked.

Second question is: "What is the consequence of Foundation on Role definition"

Consequence of Foundation on Project leader role:

Project leaders must take care of **management** tasks: they define plans, workload, budget, resources, they follow advancement, manage exceptions and changes...

But when a **Foundation is available** they are also responsible for Reuse, which means they must understand the Solution structure and functionalities offered by the Foundation. Their responsibility is not only to manage the project but also to deliver a Solution which takes advantage of Foundation.

In most Enterprises the Project Leader is simply managing the project. He or she does not Build or check Solution Architecture. Many Project leaders we met are asking for training to be able to play this new Role.

Consequence of Foundation on Business Architect role:

We noticed that the Role of the Business Architect was not uniform. We identified 2 key Roles:

The “Business Map Architect”

Concentrates on Solution Map and group Solutions in domains and areas.
Aligns this Solution Map to Business Strategy.
Defines Business Architecture rules: how Solutions must interact through Interfaces...
Defines the 3 year plan, the budget and manages the project portfolio.
Does not enter into what are considered as details, is not involved in project. Does not want to be considered as a super Business Analyst.
Does not Model Information and detailed Processes.
Does not care about Reuse. Does not care about SOA Functions, which are low level.

The “Business Builder Architect”

Plays the same Role but **also** Builds Solution Maps and Interfaces and check that Solutions are consistent.
Is **much more involved in details**.
Masters Information and detailed Process Modeling.
Is involved in all Phases of Projects, checking that Architecture Rules are well applied, Coaches Business Analysts.
Helps identify Reusable Functions and Repository Information.
Checks that Solution Architecture Reuses Components.

From what we observed, we think that the **2nd Role is more efficient** in guaranteeing a good Enterprise Architecture.
EA requires **continuity** between Business Architects, deliverables and projects. Maps which describe present and future Solutions are useful for understanding EA, but do not really contribute to Project success. Maps and Projects become desynchronized if they do not use round trip tools: the risk is that Project Leaders directly coordinate without Business Architect help.

New relations with sub-contractors

The Cooperative Approach requires new rules to **subcontract** developments because sub-contractors must **reuse Enterprise Foundation**.

Note that there is a conflict if the sub-contractor decides to Build its own Foundation to be as efficient as possible.

- the subcontractor’s teams must be trained, supported and coached by the Foundation team
- the delivered Solution must be **checked**: acceptance is not only based on delivered Functionalities, but also on the quality of the Solution Architecture
- establish **smaller contracts**: one by version;
 - Using the same Foundation helps the Enterprise to **understand the Solution Model**, even if its development is sub-contracted: large parts of the Solution are implemented through Foundation and Solution Architecture is implicitly defined by the Foundation.
 - Foundation brings **independence vis-à-vis sub-contractors** and **costs** progressively **decrease** as Foundation increases.
 - the supplier can be the same for each version as it knows preceding versions, but it can also be **different** if preceding supplier is not efficient enough

What works today

Most large Enterprises apply the Contractual Approach and not the Cooperative Approach.

4.2 Reusable IT Configuration for Transformation

Describes IT Configurations for Transformation teams: which work-stations, servers, and communications.

4.3 Transformation Approach

The same Approach can be reused by the different Transformation teams who execute Projects. It covers all Transformation activities, Solutions and Foundation:

- define problem

- analyze
- design: Processes, Functions, Information
- program
- Build test cases and Test
- document for Operation Actors and Transformation Actors
- integrate
- accept
- deploy: train, install, reorganize....
(see CEISAR White Paper on Agility)

What works today:

Do Enterprises reuse the same Approach, same Roles and same Tools?

All sponsors have defined an approach reusable by all Transformation teams.

Some Sponsors use different approaches for Business Analysis and IT Developments: cooperation would be much more efficient if same Approach and same language were used for all intervening parties. It means that the chosen Approach must avoid words not accessible to Business Actors.

Most Providers support a unique Approach. Some, like IBM, have defined several Approaches for different sizes of projects: but for a given Project only one Approach is recommended.

Using this reusable approach brings high benefits such as:

- more security for planning and workload evaluation
- better traceability
- better communication between the different Actors

The same trend is observed with Providers. For example:

- **IBM** is announcing an end-to-end modeling methodology called “CBMSOMA”, that Provides a roadmap for agility from strategy to execution. It combines Component Business Model™ (CBM), Business Process Management Models and Service-Oriented Modeling and Architecture (SOMA). This consistent approach links business strategy to processes and IT implementation. The CBMSOMA method is delivered as an IBM engagement from its consulting bodies.
- **Microsoft** proposes SOM which provides an extensible and customizable modeling framework including core business capability, process, service, and entity models to address specific industry, organizational, and customer needs. SOM takes advantage of Microsoft Services Business Architecture. SOM focuses business goals, priorities, and values and then properly aligns them with the definition and implementation of services using a Common IT architecture roadmap as defined by the Microsoft Service Oriented Architecture Maturity Model. With SOM, modeling is no longer simply a tool for systems architects and developers. SOM is the bridging technology that begins with key business drivers and ends in the development of agile software.
See : http://download.microsoft.com/download/f/8/5/f8503098-b1b9-455e-bcf6-fbe3fcf9d3f4/Service_Oriented_Modeling_Datasheet.pdf

How to improve Approach and Roles to help Reuse: from Contractual Approach to Cooperative Approach?

Present approaches are Contractual Approaches.

Sponsors are all thinking of improving their present approach. Let's present the main trends:

Looking for a Cooperative Approach

Their present Contractual Approach is based on a sequential process whose main item is the “Contract” which defines all Requirements. It is adapted to Commodity Solutions (see White Paper on Agility).

They now require a new **Cooperative Approach** (or **Agile** Approach) for Evolving Solutions. The objective is not to complete the Solution in the first version, but get it to a point where a set of capabilities can be tested and delivered to first users, and then to deliver other Versions at short intervals allowing to progressively discover needs and solve the compromise between what is desirable and what is possible: this Approach works better if a Foundation is available to guarantee Solution Flexibility and extension.

Foundation means smaller Projects:

Reusing same strong Foundation enables the breaking down of a large project into **smaller projects**: consistency is achieved by using the same Foundation and not by full detailed design of the large project before building it. First project can deliver Foundation V1 + Solution A. Second Project can deliver Foundation V2 + Solution B; third Project can deliver Foundation V3 + Solution C.

Solution B and Solution C do not need to be Modeled in the first project.

Governance to check usage of Foundations

As already described in the Governance white paper, Governance must check that Solution Builders do Reuse Foundation.

The conformity check must be done **before** and not after Solution Project approval.

If Foundation is not suitable for a given Solution, the Project leader must **provide proof**: he or she cannot ignore the Foundation which represents an investment made by the Enterprise.

Most advanced Enterprises have put this rule in place. But decentralized Enterprises have not.

4.4 Transformation Engineering Tools

4.4.1 Transformation tools are key

Transformation tools cover the full cycle: not only programming, but also specifications, Map modeling, design, tests, integration...

Many Enterprises think that Transformation Tools are **not so important**: they all deliver about the same overall productivity; or if they recognize some productivity difference, they consider that it just applies to the Programming phase which represents 20% of the global cost and can be outsourced to countries which offer low cost manpower.

Providers have a different view: they think that Tools have high consequences on Transformation productivity. They try to offer a single Transformation Environment organized around a single Meta Model, some also propose Business oriented tools:

- **Google's** strategy is to provide end customers with a single set of consistent tools which they can use internally. Google does not provide Business components. A list of main public Google components may be found at <http://code.google.com/more/#products-products-accounts>
- **IBM** is converging towards a unique Meta Model and assembles its heterogeneous tools inside the Eclipse Platform (which proposes only one tool for each need).
- **Microsoft** has made available to its client the professional tools that were used by its R&D teams. Team Foundation Server has a track record of productivity improvement among many enterprises (see <http://msdn.microsoft.com/fr-fr/teamsystem/aa718811.aspx>), especially when used with Component frameworks like ACA.Net by Avanade. But Microsoft is aiming at even more productivity with its Oslo project. The goal of "Oslo" is to provide a 10x productivity gain by making model-driven applications mainstream with domain-specific models, a new language, and tools. [http://en.wikipedia.org/wiki/Oslo_\(Microsoft\)](http://en.wikipedia.org/wiki/Oslo_(Microsoft))
- **Oracle** uses different Transformation Tools because many Providers have been bought and ascending compatibility must be respected for existing customers. But long term strategy is to converge towards a unique set of tools.
- Nucleus Research produced a report on **Sales Force**: using their consistent set of tools and components divides deadlines by 5 and costs by 2.
- **SAP** considers that Development Tool based on the Java environment is useful for solving part of their Transformation needs, but also use another Development Tool based on their own Transformation language ABAP to address high volume and performance problems. Yet SAP has decided to deliver a unique consistent set of Tools and Components, Netweaver, to accelerate projects (<http://www.sap.com/platform/netweaver/index.epx>).
- **Smaller players** also offer solutions. To give some examples:
 - **Orchestra Network** provides MDA tools to take control of master and reference data across the enterprise (<http://www.orchestranetworks.com/>)

- **Softeam** provides a modeling & generation support that covers the entire scope of the Enterprise modeling (Business & IT) in a single tool, and can be adapted to the enterprise application framework using MDA techniques (www.modeliosoft.com).
- **Wyde** provides a consistent suite of Tools based on a unique Meta Model (<http://tools.wyde.com/>)

It means that Transformation tools can have a much greater impact on Transformation Productivity than many people think: using integrated tools provides a much simpler environment with far fewer Roles.

From discussion with main Providers, we identified that the long term target was to provide **Model Driven Architecture** (MDA): Model Business and it compiles!

The main idea is to define a way to represent Business Models: this is the Enterprise **Meta Model** which covers all Models from Processes, to Business Entities, Reusable Functions, or Information Model. Different views can be given based on the same Meta Model: the Business view or the Software view, on Foundation or Solution, global or detailed.

Iterating between Business Model and executable Solution requires “**round-trip**” tools based on this same Meta-Model: modification in software automatically modifies the Meta Model and so automatically modifies the Business Model view. Communication between the different tools is much easier to achieve via a unique Meta Model than by Building interfaces between the Tools.

This round trip capacity helps iterations between the different phases: analysis, design, programming, testing, integration: this is a very important feature of applying the Cooperative Approach (Agile methodology).

There are sets of standards (UML2 for IT, BPMN for BPM) that are well recognized. The Problem is that there does not exist a unique standard to define an Enterprise Meta Model (UML2 , EMS...). Each Enterprise must make its own choice.

4.4.2 Powerful mechanisms which help Reuse

The quality and efficiency of a Building Foundation depends on Tools.

Many Components are only available if Transformation tools provide **powerful mechanisms** such as:

- Object Oriented approach: allows inheritance and polymorphism of Business Objects
- sophisticated relations help to modularize
- Business Entity access mechanisms: access to Business Objects rather than tables, mapping tool between Objects and Tables
- Powerful Typing
- Business Transaction mechanism
- Process Patterns
- Versioning for Models and Instances
- capacity to Build Processes from Process elements by composition (Sub-Processes) or inheritance (Process Patterns)
- integration of Rule engines
- multi language capacity
- Mechanisms to help Reuse
- independence with IT configuration models: OS, DBMS, Middleware
- consistency checking: to check if Foundation modifications are compatible with existing solutions
- impact analysis
- renaming: take care of not only Business Entity names but also **Function names**: experience has shown that Function Name is not so easy to establish. Should be sufficiently explicit so that Solution Builders understand what the Function does

4.4.3 External Rule Engine and Workflow Engine

Some Enterprises will have the opportunity to progressively rebuild their Solutions: as explained previously, they should rebuild new Evolving Solutions based on a Building Foundation.

But other Enterprises have not this opportunity: so what can they do?

First step is to centralize **Referentials** and offer Access Functions to all Solutions as explained above.

Second step: they can improve visibility and flexibility by extracting what often changes. One way is to extract Rules (or Functions) which often change and implement them with an external **Rule Engine**.

The Legacy system is adapted Rule by Rule: identify each Rule, replace its implementation in the legacy system by a call to the Rule Engine, and implement the Rule in this Rule Engine.

Once investment is done, it does not increase Reuse: the Model is the same. But it becomes much easier to identify Rules (visibility) and to manage these Rules without modifying the Legacy Solutions.

The downside is:

- overhead between the Legacy Solution and the Rule Engine may cause performance problems: they must be solved
- when the number of Rules increases, a new complexity must be managed: the set of Rules and their links with Legacy Systems.

Last step is to use an external **Workflow Engine** to chain Activities belonging to different Legacies.

It requires isolation of Activities or Functions inside each Legacy, addition of a new Transformation tool (the workflow engine) and synchronized modification of Legacies and Workflow models.

But, once in place, it will enable creation of end to end Processes and will externalize changes in Process Model without modifying the Legacy System.

4.4.4 What works today?

The existing portfolio of internal Solutions has been built and is maintained with **different**

Transformation tools: they may come from internal teams or external suppliers. Our Sponsors, which are large international groups, use many different set of tools (one sponsor uses 8 sets of tools in France alone: considerably more at the global level).

When a new Package is chosen, it **imports** its own Transformation tools.

It means that most large Enterprises have many Transformation tools.

To reduce this heterogeneity, at least for new internal Solutions, most Enterprises have defined a policy: they recommend usage of a limited set of tools.

But it seems that the Transformation Environment is still **very heterogeneous**, much more than the **Operations Environment** for which OS, DBMS, Middleware are much more standardized.

Most Enterprises still prefer to select “**best of breed**” tools rather than a set of integrated Transformation tools. It means that Actors are specialized by Tool:

- Designers use the Design tool,
- Programmers use the Programming environment
- Testers use the Test tools
- Integrators use Software configuration management tools
- Technical Architects use tuning tools
- Methodology Consultants use written procedures or specific tools
- Quality team uses design check tools or code check tools
- Managers use management tools
- ...

Some Enterprises have begun looking at a **limited set of consistent Engineering Tools based on same Meta Model** rather than **best of breed tools** which must be interfaced and synchronized for each change.

It is an opportunity for Transformation Actors to enlarge their roles: it reduces the number of roles.

Another advantage is that the Enterprise need not spend time selecting individual tools, checking compatibility with others, installing them and controlling ascending compatibility each time a new version is available for any tool.

4.5 Transformation Tools for Management

They help project management:

- Project repository and Project Portfolio management
- Project Planning
- Time Sheet and workload consolidation
- Evaluation Metrics

- Exception tracking

- ...

If Solution Engineering is **weak**, project requires a lot of management activities to compensate it: coordinate, test, integrate, synchronize, document...

If Solution Engineering is **good**, project requires far fewer management activities. If Engineering tools allow us:

- to reduce the number of Transformation Roles
- to organize Modeling around a unique Round Trip Meta Model
- to automatically check consistency of any update
- to organize repository of Components
- to automate part of testing activities
- ...

then there is no need to manage what is already achieved by Engineering tools.

Our advice would be to first focus on Engineering tools to ensure good engineering and good architecture of Solution, before defining Management tool requirements.

4.6 Information Model for Transformation

It describes

- reusable **Management Information Models** for Planning, project presentation, standard RFQ or proposals, Metrics for evaluation of Projects ...
- the **Meta Model**: all concepts such as Solution, Model, Functional Domain, Process, Business Entity, Function, Rule, Relation, Attribute, Type... useful in order to Model an Enterprise. It creates a Transformation language Reused by all parties
- **Repository Model** for Operation Foundations: how to store and version the different Components Reusable by Solution Builders (see list in Operation Foundation)

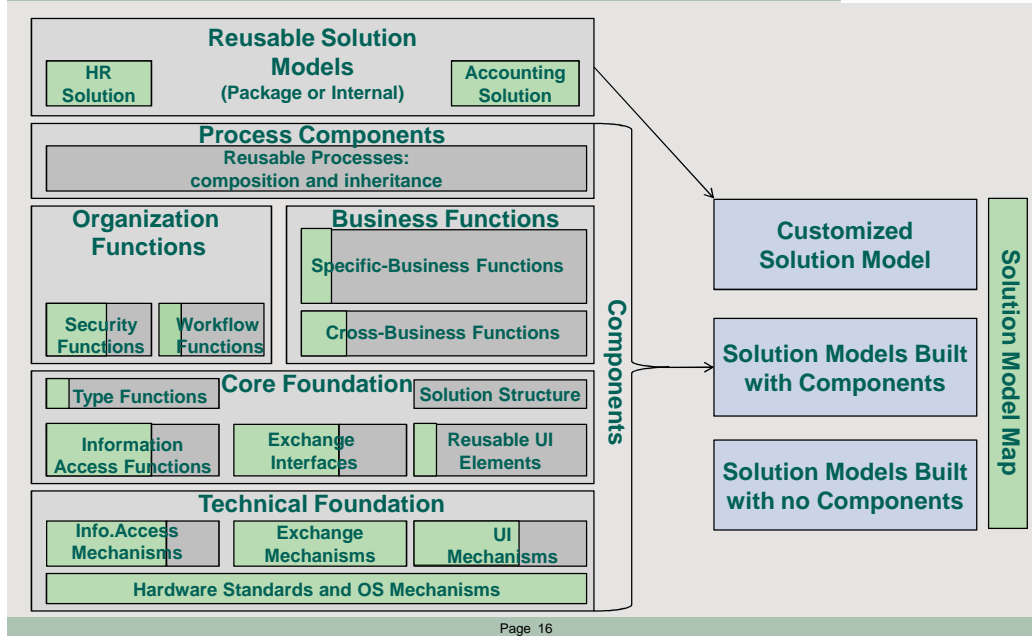
What works today?

Repository Models have been defined but Enterprise Meta Model is not well formalized: our Sponsors have not centralized a unique definition of what is: an Entity, a Process, an Activity, a Function, a Solution, a Component, a Project...

This Transformation Meta Model could be defined when the time comes to choose a new set of Transformation Tools

5 Summary of what really works today

Enterprise Foundation: what really works today



This diagram summarizes what is currently implemented in most Enterprises (in green).

Most Enterprises have developed Exchange Foundation. Very few have a Building Foundation.

What works:

- **Hardware and OS standards** have been well defined by most Enterprises
- **Information Access Mechanisms:** DBMS standard usage but no Business Object access Functions, no Business Transaction Mechanisms, versioning is not always managed
- **Information Model:** some Repositories are Shared like Customer File, Organization File. Very few have defined a Business glossary. Some have defined a Entity-Relation Model for main Business Entities.
- **Exchange Mechanisms:** most Enterprises have defined a standard middleware (EAI, Application server...) and they widely Reuse it to allow Black Function calls between Solutions
- **Exchange Interfaces:** list of Interfaces between Solutions is generally documented; they are not always up to date if tools are heterogeneous.
- **UI mechanisms:** mechanisms exist to build all kinds of user interfaces; but there are very few mechanisms which allow management of Reusable UI elements like Type presentations, inheritance of Windows, composition of UI elements
- **UI elements Models:** few Enterprises Reuse UI elements
- **Type Functions** do not really exist
- **Organization Functions** exist for Single Sign-on; Security Functions exist but there is not a single Reusable Security Function; Workflow and To-Do Lists are not generalized
- **Reusable Business Functions** are not very numerous: difficulty in specifying them, difficulty in splitting what is stable from what often changes; some Enterprises have started building SOA Business Components.
- Processes are not Built with **Process elements** by composition or inheritance (process Patterns)
- Enterprises have Built or Bought **Solution Models** Reused by the different Companies of the Group. This is mainly done for **Commodity Solutions** (no real competitive advantage, close requirements between different Enterprises), particularly true in the industrial domain.

- Enterprise Architects have developed **Enterprise Maps** to offer a global view of Entities, Processes and Solutions. It helps to understand the Enterprise Architecture, and align future Maps with enterprise Strategy. But :
 - these Maps are generally designed with **tools independent from tools to Build Solutions**: these Maps and the real world are not always synchronized
 - **Entity Map** is almost never available
 - **Process Maps** exist for Organized Processes, but do not exist for **Business Processes**: these Maps are attached to organization and must be modified when Organization changes.

6 How to measure Foundation Value

To convince top management to invest in Foundation, measuring Foundation Value is a must. The difficulty is that Value can only be checked after Foundation exists and has matured enough, while investment decisions must be made beforehand.

This is why examples or case studies are very useful.

Agility, however, is difficult to monetize. Everyone wants agility, but few are willing to pay for it; in the end, cost effectiveness is a much easier sell.

6.1 Value examples

We had **difficulty** in identifying case studies that provided proof that Reuse achieves high efficiency. It is difficult for a Project Leader to propose 2 evaluations of his or her project: with or without Foundations. People recognize that the cumulative Costs of Maintenance and Solution Evolution during the 10 years which follow is less expensive with a Foundation than without, but no numbers are available.

6.1.1 Air France

An SOA approach is currently in progress.

The objective is to simplify the Exchange Foundation by reducing the number of exchanges (or “Services”) between Solutions from several thousands to several hundreds.

It becomes easier for Solution Builders to find Services in a reduced list; but each Interface is more complex as it covers a larger perimeter. Total Modeling costs should decrease by several percent.

6.1.2 Imaging Solution

One of our Sponsors explained that 2 teams required an imaging Solution at the same time.

It appeared that the individual cost for each project was 1,440 man-days, which means that the total cost was 2,880 man-days.

The cost of a Reusable Solution was 2,600 man-days + 150 man-days by project, which means that the total for 2 projects was 2,900 man-days: exactly the same cost as for specific Solutions.

The decision was made to choose a Reusable Solution, because the marginal cost for future Projects is much lower: 150 man-days instead of 1,440 man days.

This case seems obvious: governance is easy, the first 2 Projects pay for the Reusable Solution; no investment is required from a Foundation team.

But very often, the same context does not induce the same decision: it happened because the following conditions were met:

- somebody identified that the same need was emerging in 2 projects: it means that **communication** had been organized **between Solution Projects and Foundation team**
- Project teams **accepted to Reuse** the same Imaging solution: management played its role
- A **Foundation team was in place** to take responsibility for the imaging Solution and to deliver it not only to the 2 projects in question, but also to future projects.

6.1.3 Crédit du Nord

This example is interesting because it helps us to understand the long term Value of a Foundation.

In 1983 Crédit du Nord was a bank with low productivity.

It was decided by a new top management to increase Productivity by 30%.

Instead of analyzing all main Business Processes, it was decided to Build a powerful Foundation which could allow implementation of any process optimization in a very short time.

The main **Foundation elements** were:

- Transformation Roles
 - The Developer had to be able to understand requirements, design, program and test: highly skilled developers
 - creation of a unique Foundation team

- Transformation Approach
 - iterative
 - minimum documentation and maximum prototyping
- Transformation Tools
 - a single Development Environment for each of the 3 levels: local software, mainframe software and server software
- Operation Roles
 - Operation Actors were able to execute many more Activities than before: unique work Station connected to all data bases and Solutions, standard user interface, data entry at source, all checks executed on line: this is the main reason for productivity improvements
- Operation IT Infrastructure
 - from 4 to 1 single mainframe server
 - a unique Work Station (1983 is the year of announcement of the IBM PC in Europe) for any role connected through LANs to the mainframe
 - specialized channel servers
 - automatic download of software modifications every week
- Information Access mechanisms and Functions
 - a unique Business Entity model
 - multi-enterprise mechanisms which later allowed to Operate other acquired banks
 - standardized Types: name, date, amount-currency, enumerated type, text... (Year 2000 and Euro conversions were easier than for other banks)
 - a unique Customer file
 - Reusable Functions to access main Entities
 - any Entry in the system was done through an Operation: unique identifier, tracking information (who did it, when), ability to suspend any Operation, abandon it, or transfer it
- Solution Exchange mechanisms and Functions
 - one synchronous and one asynchronous protocol for all needs
 - conversion Functions based on Types
- UI Functions
 - a unique user interface, a single desktop
- Organization Functions
 - a single sign-on
 - a generalized security Function
 - generate message to internal Actors: to group all messages by target actor instead of sending results by batch Solution
 - generate edition to Customers: to put all documents in same envelop and save stamps
- Business Functions
 - multi channel architecture: decomposition of Solution into Business Functions so that just navigation and UI had to be customized
 - office automation integration: text processing, spreadsheet and email systems were integrated with same UI, same database, same software configuration management
- a simple Enterprise Solution Map

All Solutions have been progressively rebuilt based on this Foundation. They were developed with about 50% reuse rate: half of the previous effort.

The simplicity of the Enterprise Architecture enabled the following **Results**:

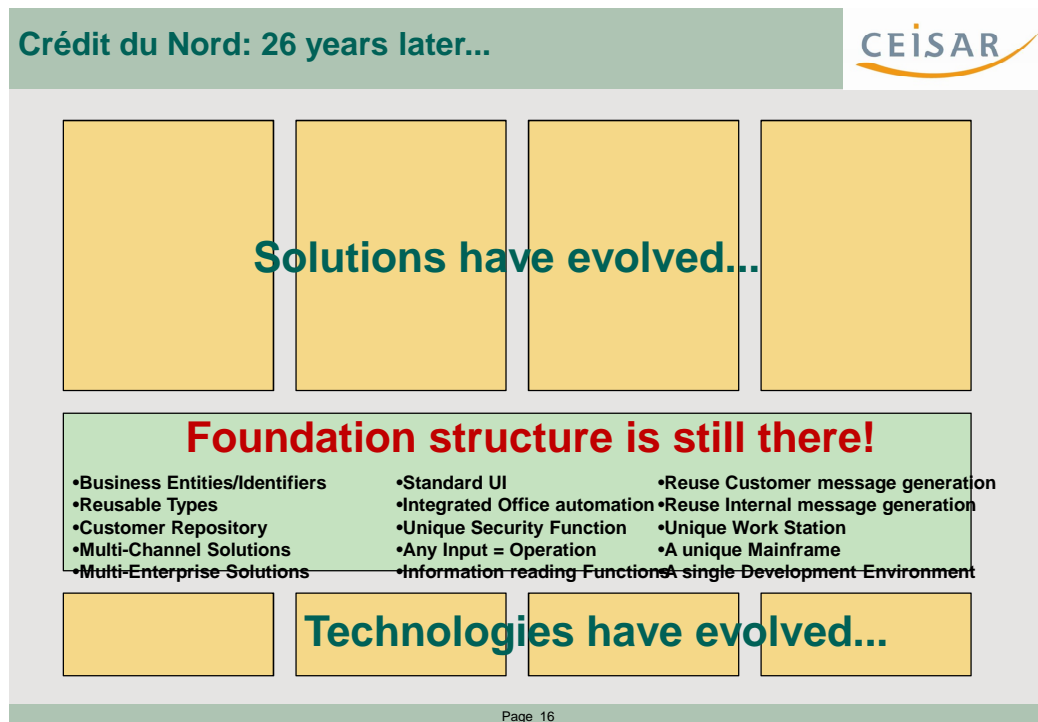
- drastically increased productivity: staff decreased from 10.300 people to 7.100 people; bank became profitable
- number of customers increased from 600.000 to 920.000
- Time to market increased: 3 times the market share on new banking products
- Quality of Service is one of the best among French banks according to annual Customer Polls.

But what remains **26 years later**?

The world has evolved:

- 5 successive CIOs managed the IS unit of Crédit du Nord

- successive technologies emerged
 - new Solutions were developed in the meantime
- But Foundation is still very powerful and keeps the Enterprise Architecture simple.



6.1.4 Google

Google explains that its own internal productivity is based on its Foundation: Approach, Tools and components.

Their Approach is based on:

- direct contact between the end user and the developer: there is no intermediary role. The Developer must be able to understand End User requirements, design a Solution, develop it and test it. It means that Google only hire highly skilled developers.
- small teams for each Solution: 5 to 10 people per team
- iterative process: first set of functionalities is delivered in first version, then following versions are driven by end users

For those familiar with our last white paper, this really reflects the Cooperative Approach and not the Contractual Approach.

6.1.5 Mobitel

Mobitel is a Mobile company from Slovenia.

They rebuilt most of their Enterprise Architecture with IBM.

They attained a 60% Reuse rate and accelerated “time to value” by 64%.

6.1.6 Sales Force

Nucleus conducted an analysis (document J29 of May 2009) of 17 Force.com projects and found significant savings in time to development and ongoing support costs. Nucleus analyzed existing Force.com application deployments and found that development was on average **4.9 times faster**. End customers, developers, and ISVs experience more rapid time to value, lower cost, and greater ongoing flexibility.

<http://nucleusresearch.com/research/notes-and-reports/force-dot-com-drives-faster-development/>

6.1.7 Insurance Package

A software editor provides an Insurance Solution for all Product lines (Life, Property and Casualty, Disability, Group Life, Health...), all Processes (CRM, Contract management, claims, billing, accounting, business Intelligence...) and all Countries.

To facilitate usage of Components by Solution Builders and decrease their workload, a powerful structure of 40.000 Business and Technical components has been Built: Solution Builder just see Interfaces of some Components, not implementations. Majority of Components are hidden. The consequences of high Reuse is that the cost to Build a Model for a new Insurance Business Line is about **5% of the total Foundation investment**. The majority of the Model is in Foundation and no longer in Solutions.
(see part 11 for more details)

6.1.8 Why does a Foundation Approach mean more complexity?

- Governance is more difficult: add
 - Foundation governance
 - Respect of Foundation by Solution teams
- Solutions become smaller if many Components are available, which means that the **Foundation is big**. To simplify the job of Solution Builders a large part of the components must be hidden to the Solution Builder, which means that the Foundation is a **complex structure**.
- The Solution Builders wants to use only what they need and not the superset of all needs of all Solutions: **Multi-Implementations** must be provided by Foundation teams for some Components.
- **Ascending compatibility** is necessary: how to guarantee that a new version of Foundation does not require modification of Solution Models using previous versions?
- Solution Builders must **know Foundation**, which is not necessary if there were no Foundation. They need training and support from the Foundation team.

6.1.9 Why does a Foundation Approach mean less complexity?

- **Volume** of Enterprise Model is **globally reduced**; to take an example: if Foundation allows us to reduce each Solution Model to 1/3 of what it was before, then the global Enterprise Model is at least **divided by 2**
- Each Solution is **smaller**: not only the Software size, but also the requirements; as a large part of the Functions are already inside Foundations, size of requirements is reduced.
- It is simpler to understand each Solution Model.
- One indirect consequence of powerful Foundation is that **each Solution Architecture is well designed**: it will be much easier to add features. It is well adapted to Cooperative Approach, prototyping and iterations.
- Powerful Foundation hides Technical complexity to let Solution Builder **concentrate on Business Modeling**: new versions of technical layers are only managed by the Foundation team.
- Components are **tested** by its first "customers". Once debugged, they do not need to be tested again for future Solutions.

In the end, is it more complex or not?

We could summarize our vision this way:

- **A bad Foundation is worse than no Foundation** (see next chapter which defines rules to Build good Foundation): bad Foundation kills the idea of Foundation
- If you get a good Foundation, then it reduces and structures the Enterprise Architecture, and **definitively simplifies it**, which has a huge impact on Agility, Quality and Cost reduction.
- Foundation Reuse generates the quality of the **Solution Architecture**
 - Main quality of a Solution is its **modularity**: a good modular Solution Architecture helps understand the Solution Model, avoids propagation of errors and facilitates evolutions.
 - **Reusing Components automatically brings modularity and Solution Architecture quality**.
 - As described above, Building Foundation adds White Components to Exchange Foundation: modularity is even better, but there is a constraint in terms of Transformation tools.
 - This is why Reuse helps the analyzing, designing and developing new Solutions. It helps Business Analysts who must analyze the Solution by Reusing Business Components and

avoids having to define what is already embedded inside pre-Built Components like User Interface, security, organization assignment...

Before detailing Value brought by Foundation, let's tell a little story about an experiment done with one of our sponsors (Total):

Total had defined global requirements for a Solution managing assignment of apartments to employees having moved to France.

We asked 3 smart students, having never developed software before, to Build the Solution with an available powerful Foundation.

- First, they succeeded in delivering the Solution.
- Second, it was done in 30 man-days, while the cost was supposed to be 90 days using a Contractual Approach (classic Waterfall methodology).

It means that when strong Foundation hides technical complexity, Building Solutions can be done by non expert people in a short time.

6.2 Main Value = Increased Agility and Reduced Costs

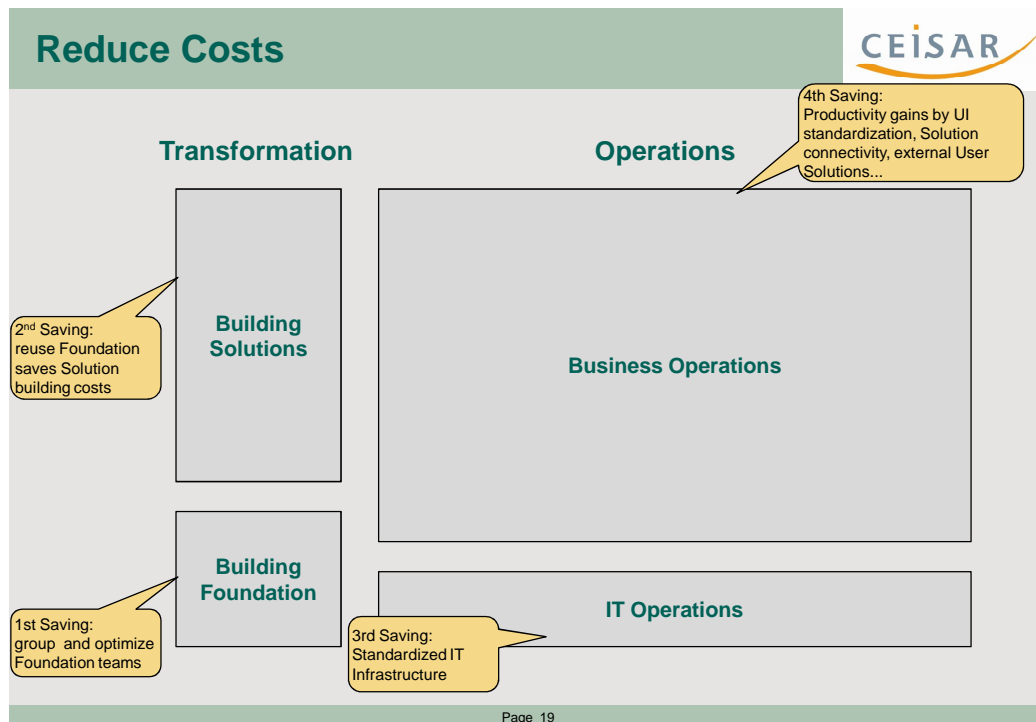
Reduction of Complexity has an impact on agility, quality of service and costs.

It **increases agility** because Reusable Components represent a large part of the work: not only for Solution Building but also for Solution evolutions.

It **improves quality of service** because Components are already tested and the Enterprise Architecture is simpler.

It **reduces costs**:

- **Transformation Costs**, because Reusing Foundation can divide by 2 to 5 the cost of building Solutions
- **IT Operation Costs** because standardization of IT infrastructure simplifies IT Operations
- but **main savings come from Business Operations**: User Interface standardization, implementation of end to end Processes thanks to Interoperability components, reduction of complexity, all contribute to productivity gains



6.3 How to convince top management

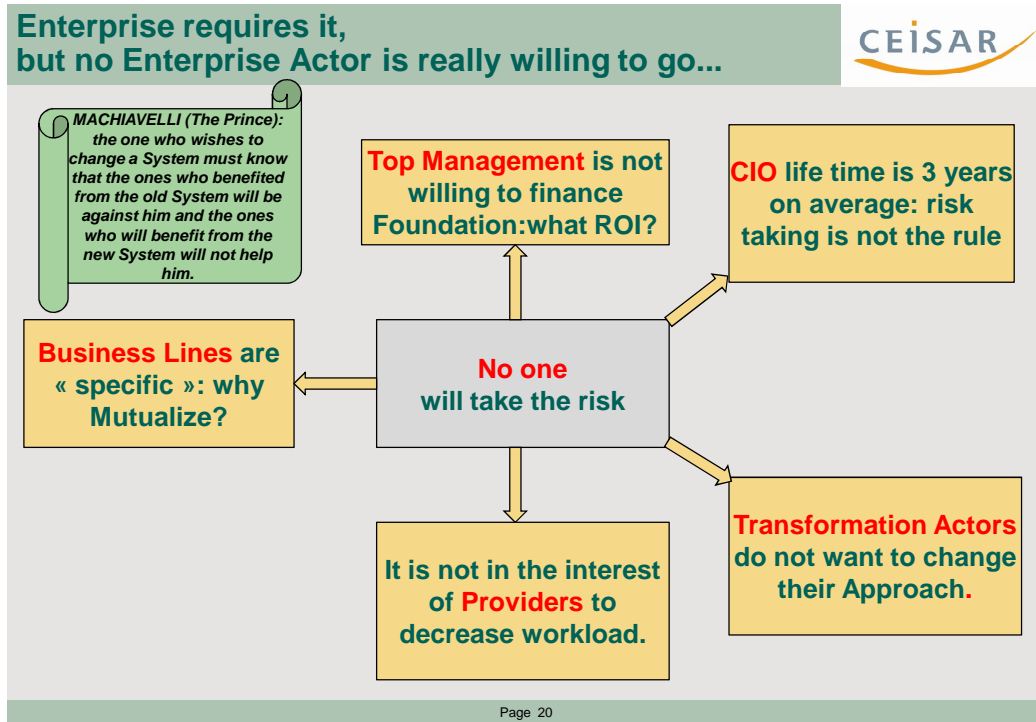
6.3.1 The 3 Scenarios

In April 2009, CEISAR carried out a survey in about 100 Enterprises.

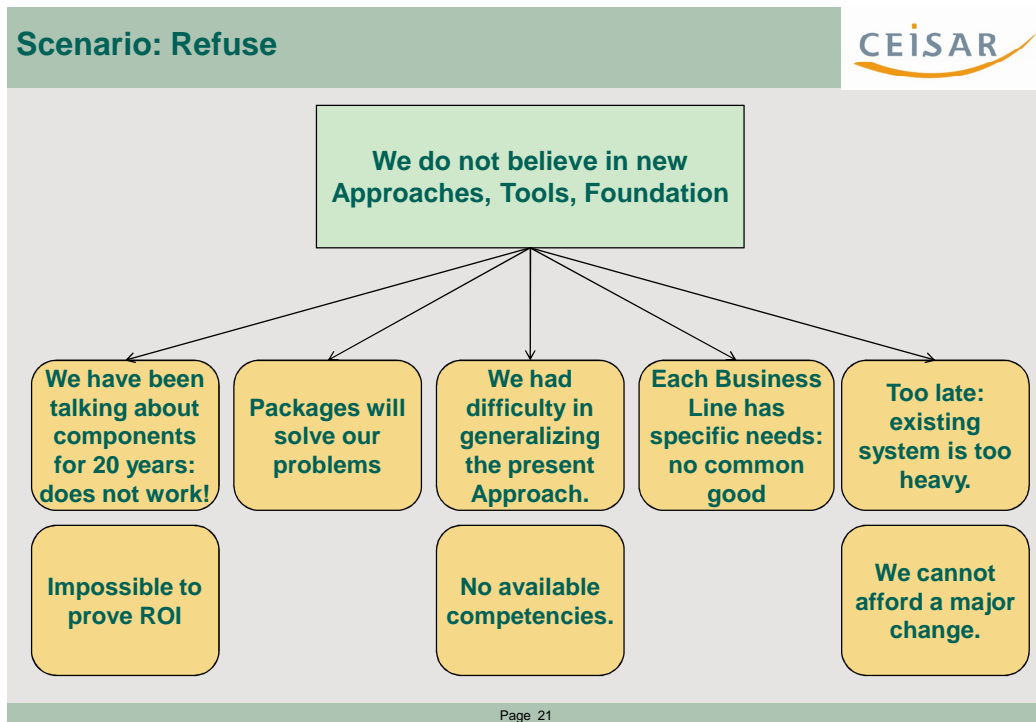
Question 1:

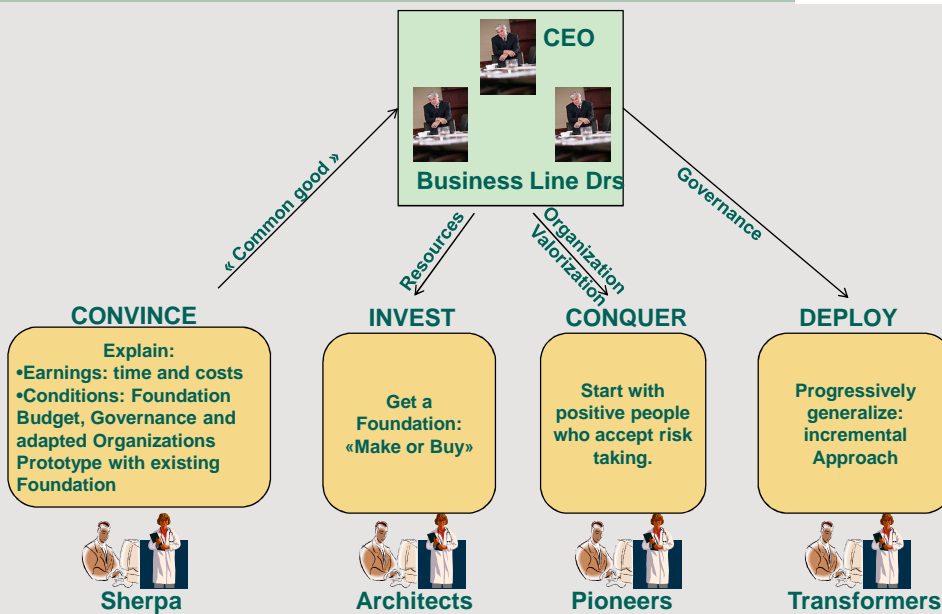
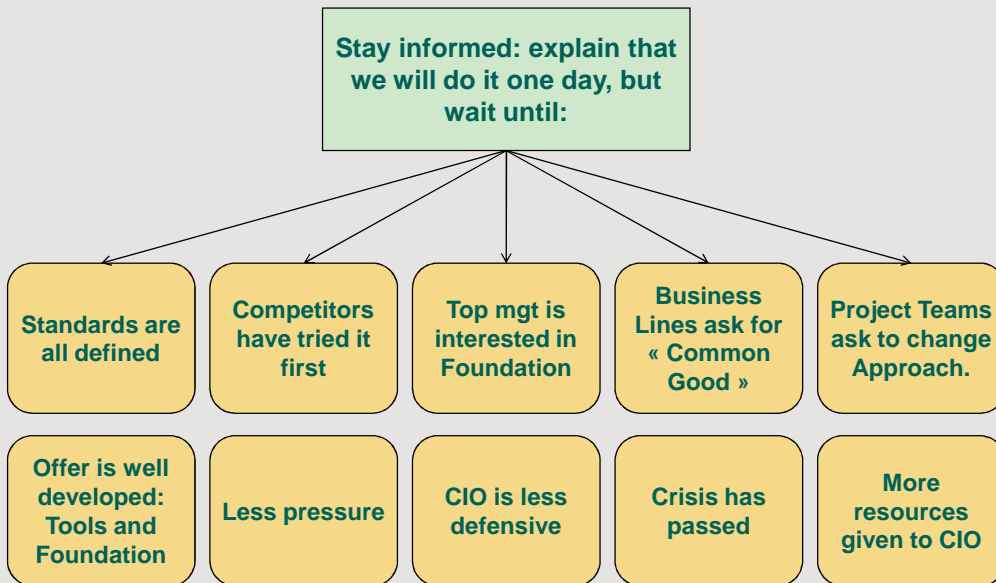
Powerful Foundation enables strong reduction (by half) of workload and timing? **79% answered "yes"**
 It means that Enterprises understand the value of Foundation.

But, creating a Foundation, using it, adapting the Approach represents a risk that no one wants to take, as summarized in the following slide. To set this change in motion, Top Management must be convinced.



We identified 3 main scenarios: Refuse, Wait and Go





We will only focus on the last scenario to help Actors who launch such an approach.

Question 2:

Is it possible to **convince your top management** to invest in Foundation? **56% answered "yes"**
 It means that a slim majority thinks it is possible to convince top management. But how to go about it?

6.3.2 Difficulty in proving Value

Proving Foundation Value is difficult, just like proving that a car manufacturer must invest in reusable parts to build new cars. It took 100 years to reach the present Reuse rate in the car industry.

The value of some Foundation items is easy to explain.

Examples:

- people understand that it is impossible to merge IT Operations if no **standardization of IT Infrastructure is done**
- people understand that it is difficult to exchange emails if different **email formats and mechanisms are used**
- people understand that it is difficult to have a **unique user identifier** if there is not a Reusable Solution to manage and store identifiers and passwords
- people understand that it is difficult to manage the **Customer** and not just the **Contract** if there is not a Reusable Model for Customer

Explanation is necessary to persuade, but it also **requires Sponsors**, in particular a Business Sponsor. As one of visited Enterprises explained, the Customer Repository became reality because the Marketing director understood the Value of such a Repository and decided to finance the associated project. Today 80 Solutions reuse this Repository!

But most Foundation parts are not required by a Business Unit. How to justify investment in:

- reusable business glossary?
- reusable Functions?
- reusable Process Patterns?
- reusable UI elements?
- new Transformation Tools or a new Approach?
- ...

Agility, is difficult to monetize. Everyone wants agility, but few are willing to pay for it; in the end, **cost effectiveness** is a much easier sell.

6.3.3 Convince Top Management

An Enterprise is broken down into Business Lines.

Each Business Line is evaluated on its own performance: sales, profitability, productivity...: they are not evaluated on their contribution to common good, they will never ask for Foundations.

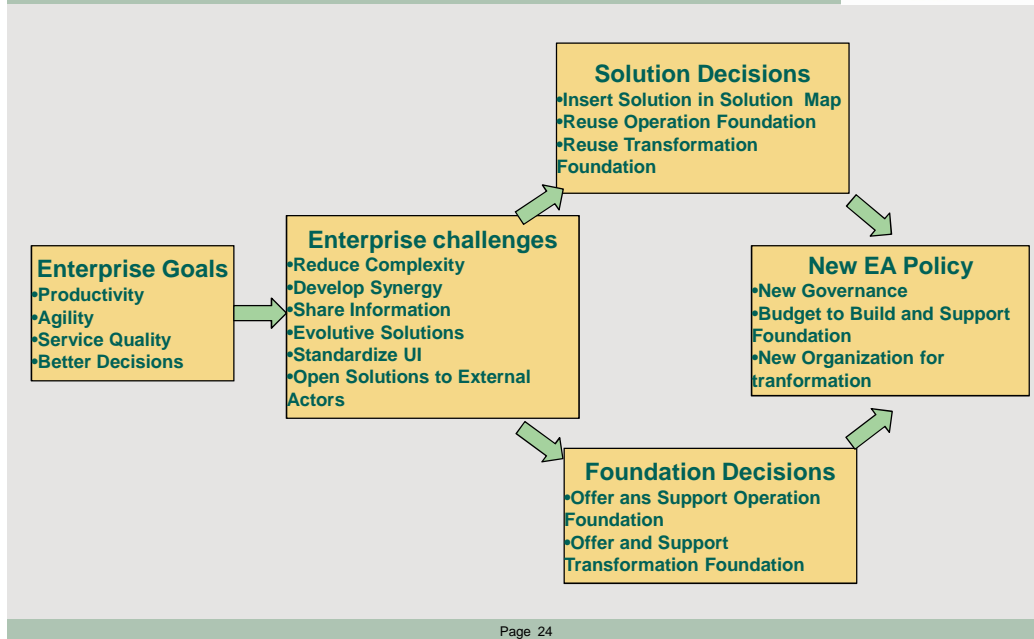
Common Good is under the responsibility of Top Management and not Business Lines.

To enforce a Mutualization process, Top Management must be convinced. If you succeed in convincing Top Management, they will define rules applied to all Business Lines which report to the Enterprise Top Management.

Yet if Top Management is not convinced, do not drop Mutualization, do it at **Business Line level**, and if Business Line Managers are not convinced, then apply it inside a **large project**: it will be less efficient but success will allow a return later to upper levels.

This is a very difficult topic because Foundation is an **abstract** domain for Top Management, far from concrete domains such as finance, marketing, sales, HR...

As Top Management is not really aware of Enterprise Architecture, Solution Map, Component Reuse... the only way to convince top management is to **align Foundation with Enterprise Strategy**.



To convince Top Business and IT executives, the process should be:

- define **what Foundation is** in Business language
- Start from **Enterprise Goals**: Productivity, Agility, Quality, Cross-Selling, Mergers and acquisitions
- Explain how difficult it is to achieve these Goals: identify the **Challenges**....For example to achieve the goal “higher productivity”, you must solve certain challenges “standardize usage of Solutions”, “share Repositories”, “interconnect Solutions”, “reduce complexity”, “automate end to end Processes”...
- Then explain how **Solutions reusing Foundation** solve these challenges: deduct a realistic **action plan** (see after) and a related Budget for **Foundation**
- and propose EA **decisions** to be taken by the top management to obtain such Solutions and Foundation:
 - **decide New Governance**: ensure that Solutions Reuse Foundation, define **indicators** to check Foundation efficiency (reuse rate, flexibility, modularity...)
 - **decide Budget** to build/buy and support Foundation
 - **decide new Organization** for Transformation: group into the Foundation team Business and IT experts who take care of Common Good (Components, security, methodology, IT architecture...)

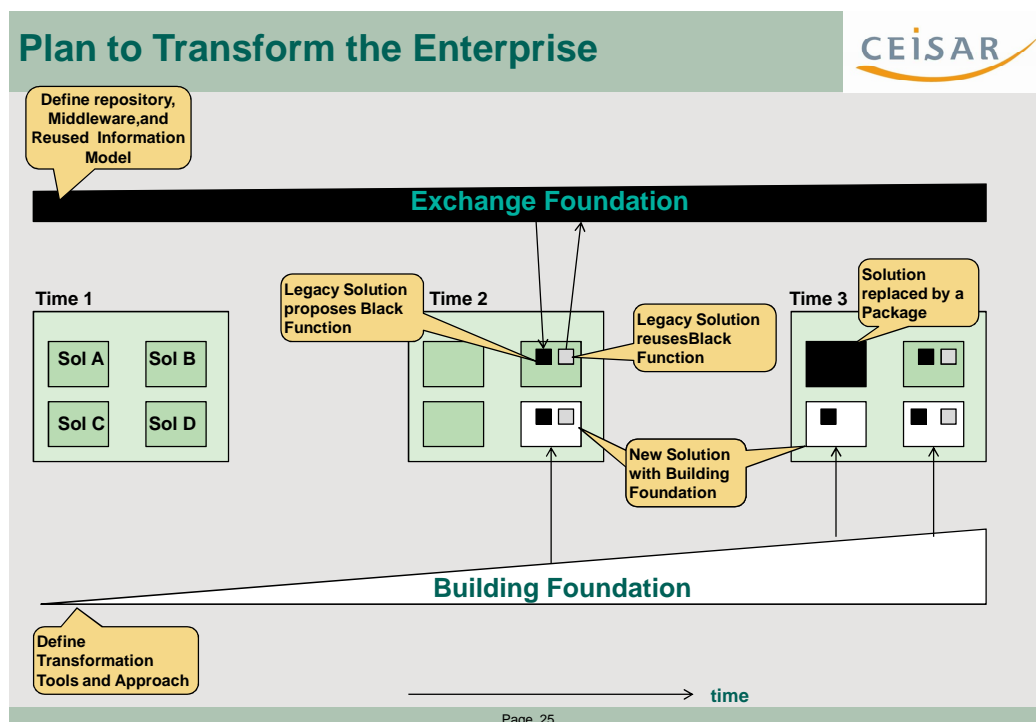
This new policy is difficult to implement if the Group has a decentralized culture: Business Lines will not easily understand that some of their decisions must be done in accordance to EA policy. It requires CEO involvement and support for several years: this is a long term approach which will deliver progressive benefits. If the Enterprise chooses to Buy and not Build Foundation, first results can be obtained faster: first Solutions can be Built **within a year**, which is important to convince the **internal Transformation teams** which generally represent the **most important brake** because the change effort is important for them:

- change Transformation Approach: from Contractual Approach to Cooperative Approach
- change Organization: creation of a Foundation team, merging Business and IT Actors in same Solution team
- change Transformation Tools
- learn what is available inside Foundation
- learn to not reinvent everything: Reuse what is already available inside Foundation
- accept a new Governance process which checks Foundation compliance
- ...

6.4 Define a Global Plan

The complexity of present Enterprise Architecture does not allow transformation of all Solutions at the same time: it will be a long haul to progressively Transform the Enterprise.

- define the **Target** in a 10 page document
 - How Target must contribute to reach **Enterprise Goals**: productivity, time to market, mergers and acquisitions, new Business Lines, new Countries, cost reduction...
 - **Map of Solutions**
 - International Solution? Multi-Enterprise Solution?
 - describe main exchanges
 - priorities to renew Solutions
 - Exchange and/or Building Foundation:
 - all Solutions must interact: requires **Exchange Foundations**
 - all future Evolving Solutions must be Built with same Transformation Model (same Approach and same Tools): allows **Building Foundation**
 - Some Commodity Solutions will be replaced by Packages
 - Target **Organization Structure**
 - Transformation and Operations are split in each Business Line
 - Business and IT in same Transformation teams
 - one Foundation team for the Group or one Foundation team for each Business Unit?
 - Target **Approach**: Contractual or Cooperative Solutions
 - impact on hiring
 - Target **Transformation Tools**: only if Building Foundation
- Select the **most important current Transformation Projects** (2 to 5) still in the early stages
- Define what must be Reused among these different Projects, deduct **Building Foundation version 1**
- Create the **team to Build and Support Foundation**
- Carefully check that everything converges
- for other Solutions:
 - do not disturb their present evolution: just inform Actors that new Adapters are provided by the new Solutions
 - however, when Solutions require big investment, decide if they should or not be built with the Building Foundation, and check that they respect the Target: all of them will reuse the Exchange Foundation



The slide above is a simple **example** of such a plan:

- The Plan must include Foundation Plan **and** Solution Plan.
- it defines present **Enterprise Architecture present status** (at times1) made of 4 independent Solutions
- it also define **Enterprise Architecture target** (at time 3) and **intermediate step(s)** (time 2) which represent priorities.
- **Exchange Foundation** starts by defining middleware, repository and Reused Information Model: then Exchange Functions will be progressively added
- **Building Foundation** starts by defining Transformation Tools and Approach: then its contents increases progressively. If the Foundation is bought outside, Building Foundation is more important from the beginning.
- at time 2:
 - Solution B has been Transformed: it **provides a Black Function** available by other Solution thanks to Exchanged Foundation, and **Reuses a Black Function** provided by another Solutions: note that Exchanged Foundation allows progressive Transformation of existing Solutions
 - Solution D has been rebuilt using the Building Foundation: it also provide and Reuses Black Functions
- At time 3:
 - Solution A has been replaced by a Package: according to its openness, Black Functions are provided (often exists) or Reused (not often allowed)
 - Solution C has been rebuilt using the Building Foundation: it also provides Black Functions

Foundation Plan depends on the Target:

- an Enterprise which mainly uses **Commodity Solutions** will require **Exchange Foundation**
- an Enterprise which increasingly uses **Evolving Solutions** will require Exchange Solution **and** Building Foundation
- Having very different Business Lines which exchange almost nothing means that Exchange Foundation will be reduced. But it does not prevent from Reusing the same **Building Foundation** in the heterogeneous Business Lines. About 2/3 of a Building Foundation are independent from the Business line: Technical and Cross-Business Foundations represent the majority of Reusable Components (see exhibit on Insurance Package which gives precise numbers)

Solution Plan depends on necessity to rebuild a Solution Model and on available Foundation. If possible, start with Solutions which include Black Functions reusable by next Solutions; for example Referential Solutions, then Core business Solutions, then Business Intelligence Solutions.

6.5 First step in obtaining Foundation budget is optimization of current expenses

Group all teams which work for “common good” inside a single Foundation team:

- technical architecture team
- methodology team
- quality team
- Enterprise Architecture team
- SOA team
- development tool team
- security team

Merging these different teams enables optimization of resources.

We also suggest grouping **all external expenses** concerning “common good”:

- software expenses: they are often spread out among different Solution teams: many companies pay for several design Tools, several accounting systems, several Business Intelligence Solutions...

- expenses for external experts: if you standardize tools and if you look for an integrated offer rather than “best of breed” offers, you reduce these expenses

6.6 How to define investments on Foundation?

But optimization of current expenses is not sufficient to pay for Foundation.

A Foundation Approach is a long term Approach: it requires a long term budget.

It is difficult to establish a single formula which computes what a Foundation should cost because Foundation can have distinct perimeters and timing according to Enterprise policy.

To help establish this budget we prefer to give some rules:

- Foundation budget must include not only Foundation Building but also **Foundation Support**: training product, documentation, consulting, coaching, hot line...
- Foundation Building comes with documentation and training products
- do not ask the first pilot Solution team to pay for Foundation: they will have to deal with the instability of first version of Foundation, they should rather be paid to be first Foundation users...
- Transformation teams should represent about **10%** of the total Transformation staff
- **Buying Foundation** will be faster and cheaper if the external Foundation offer has the following characteristics:
 - at least 50% productivity gains for Solution Transformation
 - allows **customization**: adapt proposed components without changing the overall structure: it is useful if you want to change UI Functions, Security Functions
 - allows **extensions**: Build new Components, particularly Business Components, or Interfaces with other Solutions
 - an integrated set of Transformation tools rather than “best of breed” products
 - allows business extensions
 - ascending compatibility: Solutions built with old versions of Foundation will run on new versions with minimum effort
 - produces Solutions which run on IT Operation standards (OS, Middleware, DBMS...)
- “Buy Foundation” will always require customization and extensions
- **Build/buy Foundation** elements when a real Solution Builder is ready to use it, avoid building useless Foundation.

We now propose some figures based on the following assumptions:

- staff includes Business and IT profiles
- Foundation is internally Built and supported
- with Business and technical Functions

Table to **identify savings** starting from a staff of 400 people in Transformation teams:

Reuse Savings	0%	20%	40%	60%	80%
Total transformation teams	400	355	295	235	175
Solution Team	400	320	240	160	80
Foundation team: Building	0	20	40	60	80
Foundation team Support	0	15	15	15	15

Table to **identify % for Foundation teams**

Transformation staff (Business and IT)	130	240	565	1100	2160
Solution Teams	100	200	500	1000	2000

Foundation Team	30	40	65	100	160
Building	25	30	40	50	60
Supporting (=5% of Solution Team)	5	10	25	50	100
% Foundation/Solution	30%	20%	13%	10%	8%

Support team size is proportional to number of people to support.

Building team is increasing with Solution team size, but is not proportional.

Cost for Foundation is **easier to justify in large organization**: % decreases with size.

7 What is a good Foundation?

Let's suppose that we have succeeded in convincing top management of the high value of Foundation. We must now deliver this Foundation.

Many people have tried but failed to create Building Foundation: they were unable to achieve it inside their organization because it is a difficult task. "Disappointment" was often transformed into "impossibility": I did not succeed because it cannot succeed. In the next section, we will discuss "how to Build a good Foundation"

Many questions arise:

- small or large reusable Components?
- evolution of reusable Components: how to ensure ascending compatibility?
- how to build Reusable Components for several Enterprises...

7.1 Characteristics of a good Reusable Black Function

7.1.1 Split Interface and implementation

Open Group: *"A building block's boundary and specification should be loosely coupled to its implementation; i.e., it should be possible to realize a building block in several different ways without impacting the boundary or specification of the building block. The way in which assets and capabilities are assembled into building blocks will vary widely between individual architectures"*.

A Function is divided into 2 parts: **Interface** (or "Signature", "Contract", "Specification"...) which represents how to call it and **Implementation**, which does the work.

The same Function may have a **single Interface** (how to call it) and **several Implementations** for 2 reasons: different requirements and progressiveness over time.

Let's take an example: the Reusable Security Function which checks that the current Actor can do what he tries to execute.

Interface could include: input = Functional domain + Territory + Amount and output = Authorized or Not authorized.

Different implementations can be successively Built:

1. Implementation which always answers "Authorized"
2. Implementation which just uses "Functional Domain": the Actor is or not authorized to use Contract Subscription Solutions, or HR Solutions...
3. Implementation which uses Functional Domain and Territory: the Actor is or not authorized to use Contract Subscription Solutions in a Region
4. Implementation which uses Functional Domain, territory and amount; the actor is or not authorized to use Contract Subscription Solutions, in a Region for an amount less than 10,000€.

These Implementations can be progressively Built according to Foundation team planning. But the Solution Builder may use the definitive interface from the beginning. It will allow delivery of successive Implementations without modifying Solution Models: Solutions just have to be re-tested

At a given point in time, some Solutions will be happy with Implementation 2 while others require Implementation 4.

Solution Structure allows us to define which Implementation is chosen by a given Solution: just define it once in the Solution Structure, and all Reusable Components used in the Solution Model will know which Implementation to use.

This is not only true for security Function, but also for UI implementations, Desktop implementation, Multi language implementation, and Error management implementation...

7.1.2 Versioning

As for Solutions, building Foundations means errors and iterations. It means that successive Versions of a Black Function will be delivered. Function name must be completed by Version number.

If the Interface does not shift while implementation shifts, the caller Solutions need not be modified when a new Version is delivered.

If the Interface changes we advise changing the Function name.

7.1.3 Quality

Component Quality mainly depends on quality and experience of Foundation Architects, Foundation rules and Transformation Tools.

7.1.4 Granularity

Actions may have different levels of granularity.

On one side of the chain we define a hierarchy of high level Business Processes such as:

- Transform the Enterprise
 - execute the CRM Project
 - define the Problem
 - evaluate the Solution
 - design Business Objects...
- Operate the Enterprise
 - manage Customers
 - Manage Life Insurance activity
 - define Products
 - subscribe a Contract
 - manage a Claim
 - Manage Staff

On the other end of the chain, we Model elementary Functions such as:

- acquire a Customer order
 - capture Customer information
 - Control Customer risk
 - capture order lines
 - enter product and quantity
 - control availability

Which granularity to choose?

- small or large Function?
- do we limit Function reuse to automatic Actions executed by Computers or can we also deliver Functions which require user interactions?

Our recommendations:

- large Functions are made of small Functions: do not hesitate to Build small Functions (equivalent of one page of source code)
- offer high level Business Services, as “register a passenger” in an Airline company, or “sell a complementary service” which require Human interactions and are not only question-answers as in an SOA approach
- define Business Functions independent from Organization, so that changing an Organization only requires assembling the **same Business Functions** in different Activities. It may impact on user interface: each Function is associated to its user interface: presenting one window by Function can be cumbersome for the end user: this is why it is sometimes necessary to redesign some windows for higher productivity.

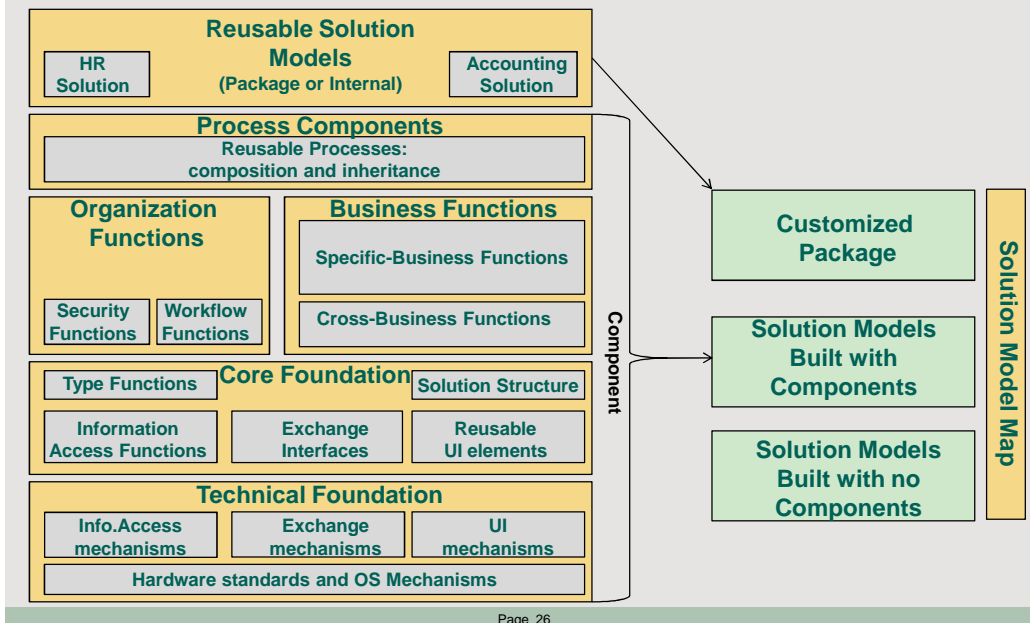
7.2 A structure and not a flat list of White Components

The Business Function “Compute Price of the Health Product” could reuse different Functions:

- “Read conditions in Health Product”,
 - with Health Product which inherits from “Insurance Product”
 - with “Insurance Product” which inherits from “Product”
- “Read parameters in Contract”,
- “Read Information on Customer”
- “Ask information complements” through a user interface
 - which in turn reuses UI patterns...

A Foundation is composed of thousands of embedded Functions.

They can be classified by layers, as we presented Operation Foundation in previous chapters.



Open Group: “Every organization must decide for itself what arrangement of building blocks works best for it. A good choice of building blocks can lead to improvements in legacy system integration, interoperability, and flexibility in the creation of new systems and applications”.

It means that there is no standard architecture of “Building Blocks”: **each Enterprise must Build its own.**

Building **White** Components must be done **from the base.**

Building **Black** Components must be done **from the top**, starting from the “Client” request of the Component: start from the Interface then develop the Implementation.

If you use Object Oriented Transformation tools and a Foundation of Business Classes, which is highly recommended to facilitate reuse, systematically create a Class which inherits from each Business Foundation Class even if no specific modification is required. It will allow you to add eventual specificities in future versions.

7.3 Scalability and performance

In terms of performance, breakdown into several IT parts means:

- advantage: it is **easy to identify** which Function spends run time energy so as to concentrate on their optimization
- disadvantage: multiplication of Functions produces an **overload of calls** to other Functions

Our experience is that Reuse requires tuning, but good performance has always been reached if the following principles are applied:

- do not use middleware to call all Functions: you do not use middleware overload to call a simple Function such as “Check Date validity”; select the Functions that really require middleware because they can be executed on another server and use simpler mechanisms for local Functions
- keep Object access mechanisms separate from their translation into relational data base access: it will enable the building of several scenarios of mapping “Business Object to Tables” without modifying Business Solution Model; Build your Business Solution once and Build mapping scenarios several times

7.4 Solution Reused for several Enterprises

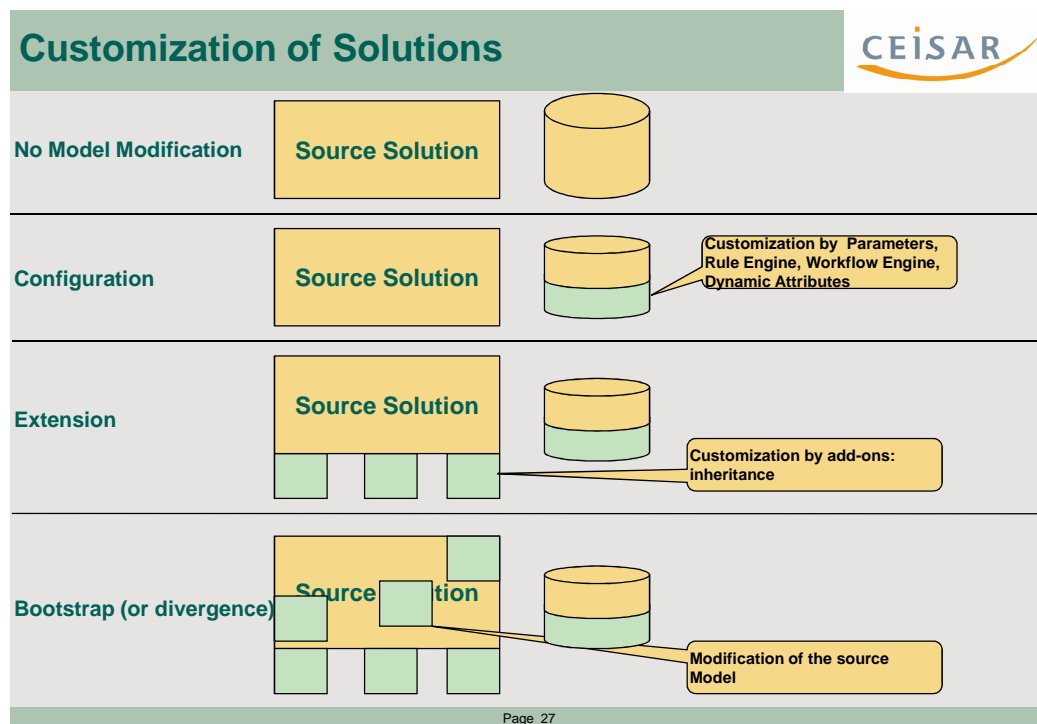
Reusing the same Solution for several Companies inside a Group has many advantages:

- Total Transformation cost decreases
- Same processes can be deployed
- Information is consistent

Maintained Solutions are Reusable if they can be Customized: Reuse will increase with Customization ability.

Let's take an example: each HR Unit of each Company of a Group Operates using the same HR Model provided by the Group

- **No customization:** each Company Reuses the Source solution Model as it is; if new Versions are delivered to Companies, they include tools to Migrate Information if necessary. The Source Model is not Modified; different instances of this Model can be Operated with different Actors and different Data Bases
- **Configuration:** parameters, rule engine and workflow engine allow modifications which do not change the Software part: this is the most common way provided by most Package Suppliers. Ex:
 - "change language"
 - "change tax Rules" thanks to **Rule Engine** at the right place in the Model,
 - add **dynamic Attributes** such as "subscribe or not to a retirement plan" (which has value only in some countries)
 - assign Activities to Actors according to the Organization chosen by each Company, thanks to the **Workflow Engine**
 Configuration Information must be isolated from Source Model so that new versions are easy to deploy.
- **Extension:** inheritance allows specializing of Business Classes; separation of original Solution Model and Customized part can be maintained independently: ascending compatibility is easier. For example, "Company 1 Employee Class" is inherited from "Source Employee Class" which allows addition of Attributes or Functions. Nevertheless, new Solution Versions are more difficult to integrate than with simple Configuration: tools should be delivered by the Provider to help Information Migration.
- **Bootstrap:** each Company gets the Source HR Model and diverges: no maintenance is required from Source Provider (Package Provider or Group Provider)



Recommendations to Build/Buy Solutions Reusable by several enterprises:

- explain that Reusing same Model does not mean that each Enterprise loses its **specificities**

- **Information modification:** if some **Attributes** have to be added for an Enterprise or if some **Types** must be adapted, use inheritance
- **Rule adaptations:** if using a Rule engine, do not forget “Enterprise id” as one of the parameters of each Rule
- **isolate Organization Modeling** from Business Modeling: Activity assignment must be externalized by Workflow Engine; different Organizations are supported by the same Business Model
- **Interfaces** to existing Solutions: same Interface with different implementations

7.5 Foundation Reused for several Enterprises

The same Foundation may be reused by different Enterprises: it happens if an Enterprise buys external Foundation, or if a Group of several Companies Builds a common Foundation. Each Enterprise must then adapt the Foundation.

Recommendations:

- **language:** rather than redesigning all screens or printouts, favor a mechanism which asks for a translation dictionary and automatically produces updated windows; it is then possible to update the windows which require adaptations (for example, because translated fields are longer than before); this approach enables us to only translate a dictionary of terms. Changing language applies not only to different Countries, it also applies to Companies from the same country which use different business wordings: “Contract” or “Policy”? “Customer” or “Member”?
- **Information ownership:** for all Business Objects which could be owned by each Enterprise (like a Customer file or a Contract file): check identifier structure

7.6 Add Specific Functions to Foundation

Solution Builders may Model a new specific Function which could be Reused by other Solutions.

Recommendations:

- Do not let Solution Builder directly distribute its Reusable Functions to other Solution Builders: it must be delivered by the Foundation team
- Before delivering this Function, the Foundation team must:
 - check that its design fits with Foundation standards: multi Enterprise Function, reuses other Functions...
 - prepare Documentation and training products
- Maintenance of this Function must be done by the Foundation team and not the Solution team which first Built it, so that each Solution team only has one provider: the Foundation team
- At the end of each Solution project, identify which specific Functions could be Reused.

7.7 Enterprise Model and detailed Model

Some Enterprises oppose Enterprise Model and detailed Model.

Enterprise Model represents an overall view of Processes, Business Entities or Solutions, through Maps at different periods: the gap between Maps helps to define long term evolutions.

Detailed Models represent the detailed Actions, information and Actors through Models and Software. The difficulty arises from the discontinuity between the 2 worlds. If they are not updated at the same time divergences may appear: Solution Builders do not use Enterprise Maps because interfaces between Solutions have evolved.

The best situation would be to have a **unique Meta Model** which offers different visions (global and detailed) according to each Actor requirement.

8 How to get a Good Foundation: Success Factors

Efforts to Build Foundation depend on different factors:

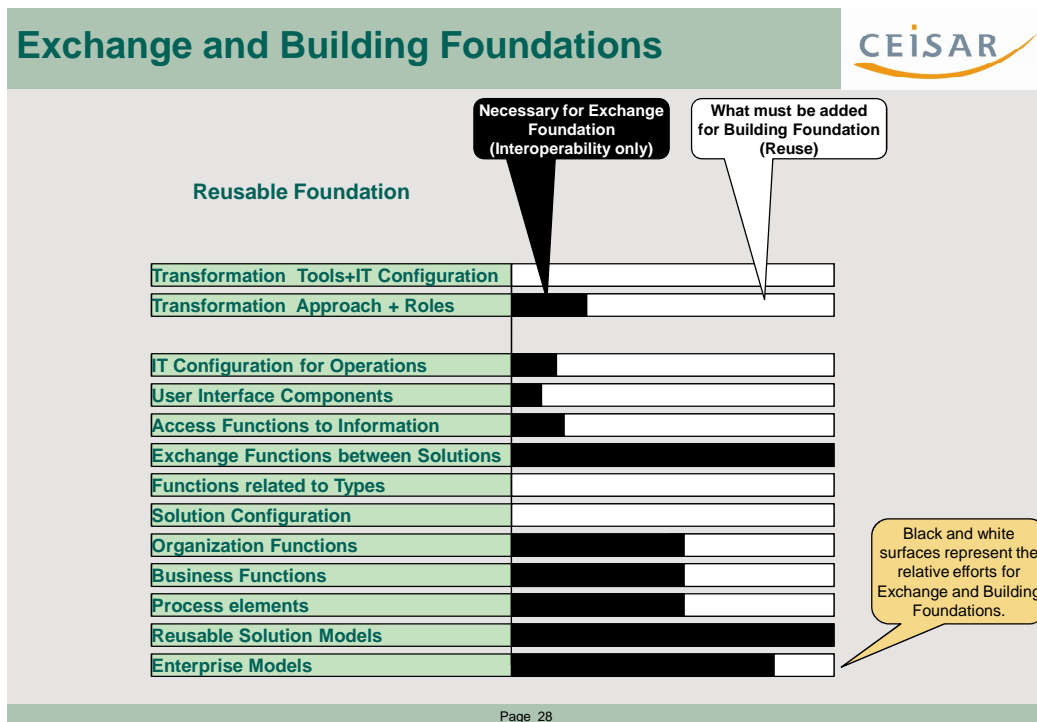
- Exchange or Building Foundation?
- Quality of Transformation Tools
- Resources: Quality and experience of Foundation Architects and Budget
- Customer pressure

One way to accelerate Foundation availability is to buy external Foundation: what recommendations can be made?

Between a full Foundation now and no foundation at all, there may exist different levels. It is possible to start Building Foundation in small steps:

- group all teams working for common good into one Foundation team
- define a Business Glossary, and a Business Entity Model
- define a single Middleware
- implement the Cooperative approach on some evolutive project
- experiment with some modern Transformation Environments
- ...

8.1 Foundation effort is not the same for Exchange and Building Foundation



- if an Enterprise decides on an Exchange Foundation, the Foundation team must Model the black rectangles which allow interoperability between Solutions
- if an Enterprise decides on a Building Foundation, the Foundation team must Model the white rectangles with same Transformation Tools

Let's comment on each line:

Transformation Tools + IT Configuration for Transformation

If an Exchange Foundation is applied, each Solution team may define its own Transformation Tools and IT configuration for Transformation: the Foundation team has no recommendations.

If a Building Foundation is applied, the Foundation team must decide on a set of transformation Tools and define the adapted IT Configuration for Transformation.

Transformation Approach and Roles

If an Exchange Foundation is applied, the Foundation team must define part of the Approach so that Built Solutions intercommunicate:

- Solution Team must Reuse the Information Model (Entity Definitions, Relations, Attributes and Types)
- Solution teams must Reuse Business and Organization Functions

If a Building Foundation is applied, the Foundation team must define a more detailed Transformation Approach which favors Reuse of White Components (user interface components, inheritance of Business Objects, Type Functions, Process Patterns) and takes advantage of the Transformation Tools (iterative approach, test tools, documentation tools, integration tools...)

IT Configuration for Operations

If an Exchange Foundation is applied, each Solution may choose its own Operation IT Configuration: the only constraint is that all Solutions which intercommunicate must Reuse the same Middleware.

If a Building Foundation is applied, IT Configuration for Operations is constrained by what generates the Transformation tools. Some sets of Transformation tools allow generation of Models which Operate on different IT Configurations.

User Interface Components

If an Exchange Foundation is applied, User Interface look and feel standards can be **documented**, so that Solutions usage is consistent. But standards are not always applicable: they are constrained by Transformation Tools or by the Package provider.

If a Building Foundation is applied, User Interface is Built Reusing same UI Components (Windows pieces, window inheritance, Type presentation, navigation patterns...): standards are easier to apply.

Access Functions to Information

If an Exchange Foundation is applied, it is possible to Build Reusable Mechanisms to access Information belonging to another Solution. This Mechanism Reuses same middleware.

If a Building Foundation is applied, many other mechanisms can be Reused: Business Object access rather than Table access, identifier generation, versioning, dynamic Attributes, navigation through Relations, Business Transactions, replication Functions...

Exchange Functions between Solutions

If an Exchange Foundation is applied, exchanges between Solutions must be well defined. It enables a first level of Reuse through Black Functions to access Information, update Information, receive flows from input Solutions or feed output Solutions.

If a Building Foundation is applied nothing more is required, except to facilitate Solution Projects by providing an Exchange Function repository integrated with the set of common Transformation Tools.

Functions related to Types

If an Exchange Foundation is applied, no work is required.

If a Building Foundation is applied, provide Reusable Functions related to Types:

- Reusable Types for Date, Time, Currency, Zip Code...
- all Attributes which Reuse "Text Type" benefit from text editing Functions,
- all Attributes which Reuse "Hierarchy Type" benefit from tree Functions,
- all Attributes which Reuse "Table Type" benefit from table manipulation Functions,
- ...

Organization Functions

If an Exchange Foundation is applied, Authentication and Security Functions can be Reused as Black Functions.

If a Building Foundation is applied, Workflow Functions can be proposed to dynamically assign Activities to the right User.

Business Functions

If an Exchange Foundation is applied, Black Business Functions are proposed by the Foundation team: Compute price, generate accounting entries...

If a Building Foundation is applied, White Business Functions are proposed to Build Business Functions by inheritance.

Process elements

If an Exchange Foundation is applied, exchange mechanisms can be used to launch Process elements which are modeled in different Solutions.

If a Building Foundation is applied, Process Patterns can be Built by the Foundation team to facilitate Process Modeling when Processes resemble each other.

Reusable Solution Models

If an Exchange Foundation is applied, Foundation Team may propose Solution Models which are Reused by different Enterprises. These Solutions interact with other Solutions.

If a Building Foundation is applied, nothing more is necessary.

Enterprise Models

If an Exchange Foundation is applied, Foundation team Builds Process Maps, Entity maps, Solution Maps

If a Building Foundation is applied, add Maps of Reusable White components.

Exchange Foundation can be a first step. It is simpler and already delivers good Value (see above).

Building Foundation can be progressively Built and applied, according to opportunities: each time a new Solution Model has to be Built, it creates requirements for new Transformation Tools and White Components.

8.2 The difficulty of creating Building Foundation

Creating Building Foundation is a long and difficult task.

The example of a single Insurance Package for all Business Lines and all Countries described in the last chapter, is a good example of the ability to Build Solutions based on powerful Building Foundation: a new Business Line Model represents only 5% of the Building Foundation Model!

But you cannot reach that level of Reuse in one step. The key difficulty is abstraction: how to discover what is common among Business Models which look so different?

When Business Operation Actors present their activities, they always focus on what is specific to their Business Domain, they never present what is common to other Business Domains. The task of the Transformation Actors is to progressively discover what can be Mutualized.

The Building Foundation Model is broken down into different layers, from Technical Components to Business Components.

Start with technical components to progressively ascend in Business layers, because upper layers Reuse lower layers: you need to understand Functions and mechanisms offered by lower layers to properly design a new layer:

- Model Types before Modeling Reusable Access Functions
- Model Reusable Access Functions before Modeling Cross-Business Functions
- Model Cross-Business Functions before Modeling Specific Business Functions

The market offer is growing: it will allow Enterprises to base their Building Foundation on pre-Built layers. Be careful to select a Building Foundation offer:

- which works efficiently: best way is to Build a real Prototype of Solution
- which is customizable
- which is easy to use for Solution Builders
- which manages ascending compatibility: Solutions Models should not be modified when a new version of Building Foundation is available

The last criterion is the most important, but it is difficult to achieve in first versions when the Model is not yet stabilized.

Some Enterprises use **copy/paste** in first versions: the Solution Modeler copies the Building Foundation and diverges. It saves time and money for first version of Solution Model, but does not take advantage of future versions of Foundation: full maintenance is carried out by the Solution Modeler.

8.3 Which Transformation Tools to help Reuse?

Building and reusing a Building Foundation requires Transformation Tools.

Many **White Components** described above are **only feasible if Tools are powerful enough**:

inheritance, rich typing, versioning...

8.3.1 Powerful features

The Tool features which help Foundation are the following:

- a **single Meta Model** which represents Business and IT Models for Solutions and Foundation
- **Versioning** to identify successive elements of Foundation and facilitate ascending compatibility: comes with automatic comparisons between different Versions of Models
- Sophisticated **Relations**: Relations between Business Entities must be defined as “simple or multiple”, “owner or not”, “versioned or not”, “both sides or not”: the sophistication of relations allows us to connect Foundation Entities and Solution Entities
- **Object oriented** features to Build White Components (inheritance and polymorphism); also useful for User Interface: Windows inheritance and polymorphism
- **Powerful Typing** to help Build type Functions
- **Integrated Rule Engine**: to allow isolation of what often changes
- **Integrated Workflow Engine** to Reuse same Business Process in different Organizations
- **Persistency engine** for **Business Entities** using standard functions from DBMS, to Build Business Functions
- **Straightforward incorporation of** external components
- **Work Group** features to allow Solution teams and Foundation team to work in parallel and synchronize easily
- **Software configuration manager** able to manage thousands of small Functions related to each other
- **Component Repository** to store Foundation elements
- This set of tools must **permanently check Model consistency**: impossible to delete a piece of Information if this Information is used by any Model; impossible to change a Function Interface if old Interface is still in use: helps to manage evolutions of Foundation and obtain **impact analysis**...

8.3.2 Represent different granularities

The tool must represent a Solution or a Foundation.

It also must represent an Enterprise Model made of Different Solutions and one Foundation.

This can be done at Group or Company level (check vocabulary)

8.3.3 The Meta Model

A tool is based on Concepts like “Business Entity”, “Process”, “Function”, “Actor”, “Event”, “Block”, “Class”, “Service” ...

The definition of each Concept and their relation to each other is defined by the Meta Model.

Each Enterprise uses its own Meta Model.

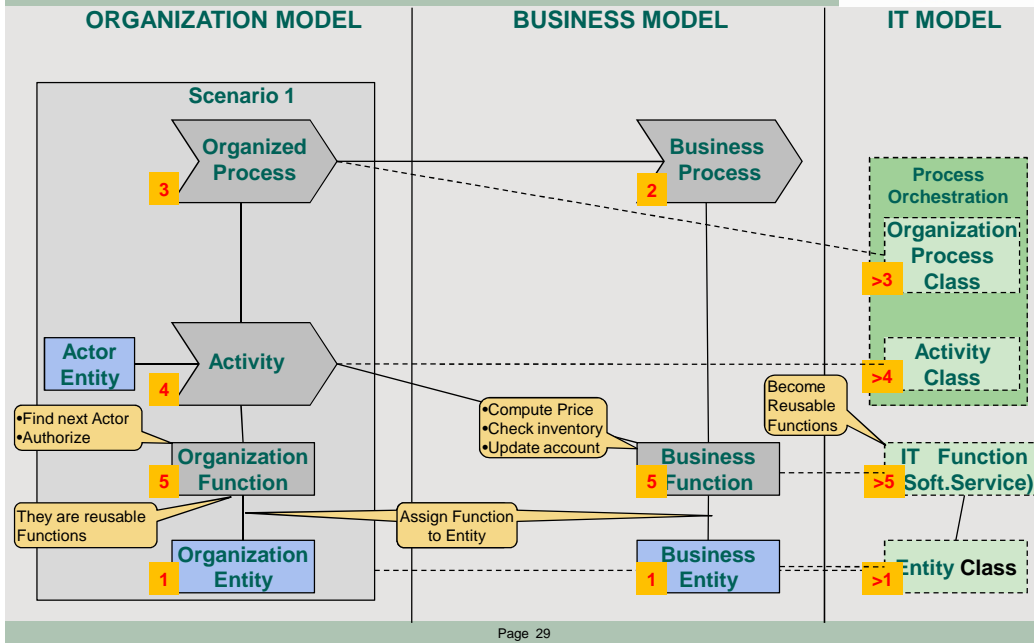
Normalization standards bodies (like UML, Process Management, ...) define concepts, sometimes too many.

Our first goal was to define a basic Meta Model which fits different companies.

Working with CEISAR sponsors (Air France, Axa, BNPP, Michelin and Total), helped to define a common Meta Model.

This Meta Model is essential for creating Foundation and Solutions: as the reader may have noticed, Reuse of the same words “Solution”, “Foundation”, “Function”, “Business Entity”, “Business Process”, “Organization Process”... is necessary to explaining Foundation.

Even if you use other terminology (such as “Black Function” for “Service”, or “Solution” for “Application”), you still need to use a Meta Model for a successful Foundation Approach.



Let's summarize what was described in former white papers:

- a **Business Entity** is a representation of a real Business Object: a Contract, a Product, a Customer
- a **Business Process** is broken down into **Business Functions**: it describes successive Actions necessary to execute the Process independently of the chosen Organization.
- Processes are structured in a hierarchy of **Process Domains**
- **Business Functions** are grouped by **Business Entity**
- for each Organization scenario, one **Organization Process** is defined: for the same Business Process there may exist several Organization Processes
- Each Organization Process is split into **Activities**: an Activity is a set of Business Functions which are always executed together by the same Actor (Person or Computer) at the same time
- **Organization Functions** are added to each Activity: security, workflow, To-Do list Functions are not defined at Business Process level
- **Organization Functions** are grouped by Organization Entity
- Organization Processes, Activities, Functions and Entities are all related to **IT Classes** which implement the associated software

Some comments on the Meta Model:

- There is just **one** Meta Model for Organization designers, Enterprise Architects or Software Developers.
- This is a **round trip** Meta Model: when a change is made by a Transformation Actor (ex: Business Analyst, Developer), the Meta Model views of all concerned should automatically change.
- **Consistency rules**: any modification in the Model must be checked by strong consistency rules.
- Use **Standards** when they exist (UML, BPMN, ...)
- A **glossary** of terms must be defined before the Meta Model itself. As a concept is defined thanks to other concepts, check that its definition covers what is required.
- "Action" can be specialized in many different concepts like "Business Process", "Organization Process", "Process Domain", "Activity", "Procedure", "Operation", "use case", "task", ... We have defined a middle position: each Enterprise should be allowed to **extend the Meta Model** if necessary.
- If one standard Meta Model becomes well recognized, in the future Models will become exchangeable: exchange formats must be defined based on the Meta Model, and tools must

accept input and output of these exchanges.

Any Meta Model concept must be defined at least through:

- Inheritance
- id, version, name, text description
- relation with other concepts.
- Generators
 - A Meta Model comes with a **documentation** generator: enables extraction of that part of the Meta Model adapted to each role. Generation should be automatic and produced on different formats (word, HTML, XML, ...)
 - It must be also possible to generate other outputs like: Processes (BPML), code, data model

8.3.4 Cloud Computing: an opportunity to integrate Transformation Tools

Wikipedia definition:

“Cloud computing is a style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet. Users need not have knowledge of, expertise in, or control over the technology infrastructure in the "cloud" that supports them.^[3]

The concept generally incorporates combinations of the following:

- infrastructure as a service (IaaS)
- platform as a service (PaaS)
- software as a service (SaaS)

Cloud computing services often provide common business applications online that are accessed from a web browser, while the software and data are stored on the servers.”

Cloud computing offers Operation and Transformation Services.

Cloud Operation Services facilitate Enterprise Operations:

- no hardware to manage
- no network (except internet access)
- no scalability plan
- no IT Operation Software to install or upgrade on servers or work stations: just keep a Web Browser
- few Operation staff to save/restore Information, manage changes, launch batches

One remaining question is how to solve security problems: access, information ownership.

Cloud Transformation Services facilitate Enterprise Transformation: Operation advantages can easily be translated into Transformation advantages. The main benefit is that **Cloud Computing** can offer a **pre-integrated set of Transformation tools and reusable Components**. The Enterprise which chooses to Build Solutions based on Cloud Computing, still has to Build its own Business Foundation, but it will find a Transformation Foundation which will make life easier.

Salesforce.com began as a monolithic SaaS Solution. However, its fundamental Functions, useful and reusable across a wide range of custom Solutions, have been teased out and made available in the company's Force.com offering.

Arguably, even Amazon, the pioneer of cloud computing, began as a monolithic web Solution selling books. It then followed up with granular Functions that plug into its core, and finally provided the basic building blocks of their Solution—queuing, database, storage, elastic computing—as Functions for hire. This is SOA evolving into cloud computing.

8.4 Quality and experience of Foundation architects

Building Foundation is more difficult than Building Solutions because

- there exist several Customers for Foundation
- it is a 3 level Structure (Solution based on Foundation based on technical layer) and not a 2 level structure (Solution based on technical layer)

It means that a large part of the best Architects must join the Foundation team.

Required qualities: abstraction, structure, capacity to identify common parts in different specific Solutions, communication, pedagogy

Experienced Architects: the learning curve is long. It takes time to understand a structure of Components, to simplify Function interfaces, to Build Components with other Components. Hire experienced Architects if you want to avoid the usual pitfalls.

8.5 Foundation Customer is required

Foundation should be built **before** Solutions as Solutions Reuse Foundation.

But good Foundation is Built from its Customer, the Solution Builders' requirements: this avoids the Building of useless or over-complex Functions. Yet this means that Foundation is actually Built **after** Solutions. This in turn means iterations: thus a Foundation is progressively Built; successive versions are delivered to Solution Builders. It is a long process.

Black Components are Built starting from Business requirements.

Ex: find interface for the Function “am I authorized?”.

White Components are Built starting from Technical Functions, ending with Business Functions. We need to start with Types, Information Model, and go up in layers as described above. White Components are defined by Builders who progressively discover how to break down Solutions into Reusable pieces. The only way to shorten this process is to **buy an external Foundation** and Customize it.

8.6 What Organization is most efficient for Foundation?

If a Group is broken down into Business Units, an efficient target could be to isolate the Foundation team and merge Business and IT Modelers into the same teams.

Rule 1: each Business Unit splits Operations and Transformation

Operation Actors produce, sell, administer according to the Operation Model. They represent the majority of Actors. They generate the Enterprise Revenue.

Transformation Actors Build the future Operation Model or Modify the existing one. They design new Products or new Processes, define Procedures, develop the associated Software, train people and deploy.

If you mix Operations and Transformation, Operations is always a priority and reduces Transformation efforts.

It does not mean that Operation Actors are not involved in transformation: the Sponsor of a Project is an Operation manager; but it means that Building a Model demands different talents than those necessary for Operations.

Rule 2: Each Transformation team includes Business and IT people

The Contractual Approach is adapted to commodity Solutions: but evolving Solutions require a Cooperative Approach which requires that Business and IT Modelers be merged in the same project team to decide the best compromise between business hopes and IT possibilities. It is not “IT slaves, build a Solution to support all my Business requirements”, it is “Together, let’s define a first Version of a Solution which delivers useful Functionalities at a low price and which can accept future add-ons”.

Rule 3: a Foundation team is created at Group level which works for all Business Units

To guarantee that all decisions on Foundation are consistent, the Foundation team is unique. It includes all teams which work for Reusable Models such as “Technical Architecture Team”, “SOA team”, “Security team”, “Quality Team”, “Transformation Tool team”, “Ergonomics team”, “Process team”,... As for Foundation team, the Foundation team owns Business and IT Actors.

Rule 4: the Foundation team includes a “Support Team”

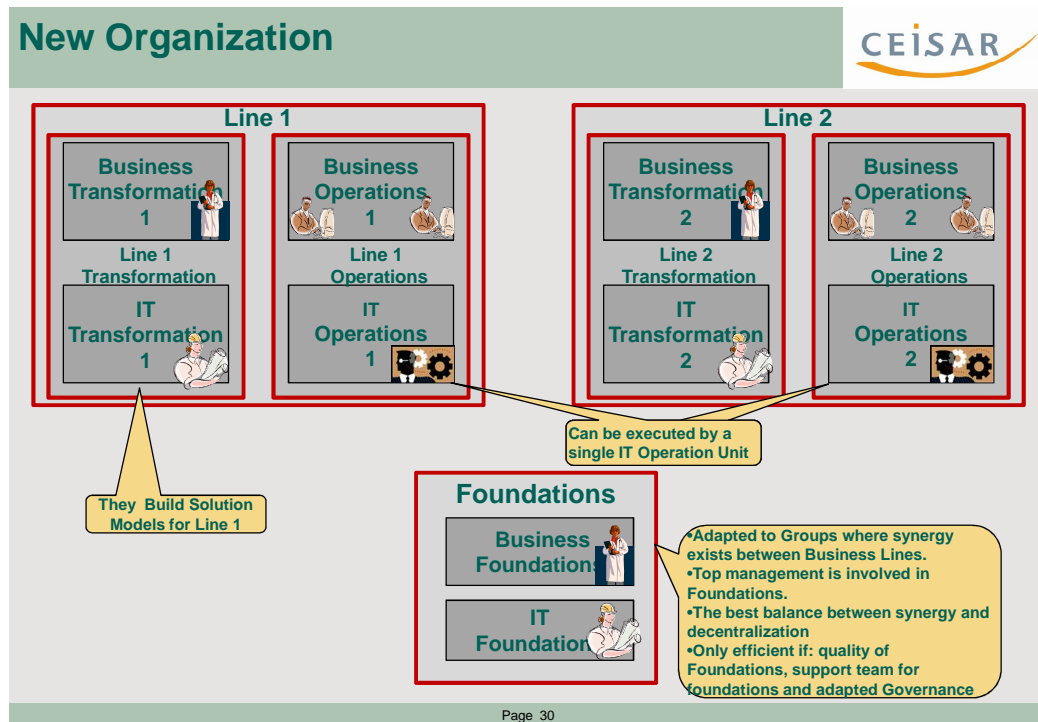
Building a good Foundation is not enough: Foundation must also be supported.

A Solution Team often requires help (training, coaching, checking) from the Foundation team: to facilitate relationships, the Foundation team is composed of a Building team and a Support team. The Support team will solve the majority of requests from Solution teams. If they cannot answer a question, they ask the expert of the Building team to solve the problem.

The Solution team is considered as a client of the Foundation team which must act as a software editor. A good way is to assign one Support Architect to each Solution Project: their role is to coordinate all actions to help the Solution Team.

With this scenario, the traditional IS department is shared in 3 ways:

- IT Operations joins Operations
- Foundation joins the Group level
- each Business Unit owns its Transformation team



Exchange Foundation:

Black Functions are Built by Solution Builders The **role of the Foundation team is light**: define the Reusable Information Model, define exchange Mechanisms, manage Black Function repository for all.

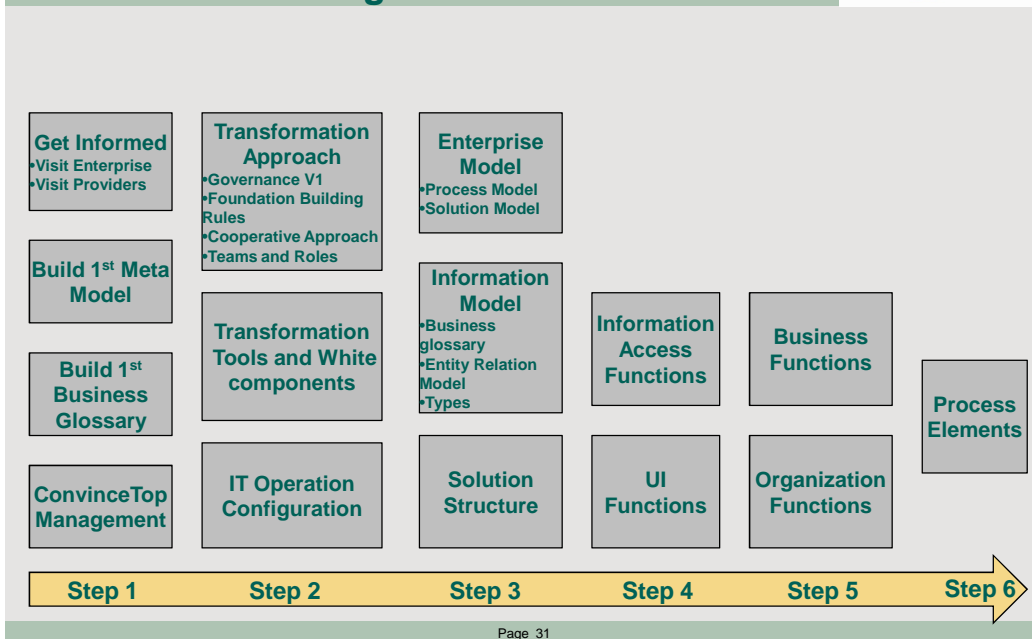
Building Foundation

White Functions are Built by the Foundation team; the role and the size of the Foundation team is much more important which is largely compensated by a decrease in size of Solution teams. Foundation team may become bigger than Solution teams.

8.7 Planning for a Foundation approach

8.7.1 A determined approach for Exchange + Building Foundation

An Exchange and Building Foundation Planning



Page 31

If there is a desire to genuinely make large investments in Foundation, the Process could be the following:

First Step is to:

- Choose or define the Enterprise **Meta Model V1**
- merge all teams working for common good into a single **Foundation Team** and split the Foundation team into: Building and Support
- **document** what already exists in Foundation
- develop a **Business Glossary** and start working on a Reusable Information Model
- **visit** Enterprises and Providers which have already developed a Foundation approach: understand the value they get, the efforts they made, the planning they followed
- get informed on **Transformation Tools** and **Approaches** which deliver high productivity
- prepare arguments and **convince Top Management** to invest in Foundation

Second step is to define:

- Transformation Approach
 - **Governance** requires defining how to check that Solution Builders reuse Foundation and how decisions for Foundation are made: a global approach
 - **Foundation Building Rules** include: Granularity, naming conventions, versioning, ascending compatibility, ownership
 - **Cooperative Approach** for Solutions, rather than Contractual approach
 - **Teams and Roles** for
 - Sponsors
 - Foundation Builders: Business and IT
 - Foundation Support
 - Solution Builder: Business and IT
- **Transformation tools** which support the Approach: this is the first engineering decision because it is impossible to start Building without tools. Define the list of **criteria** and check with other Enterprises using same tools. Once Transformation tools are defined, start Building **White Components**.
- and target **IT Operation configuration** (OS, DBMS, Middleware) which must be supported by Transformation tools

Third step is to Build

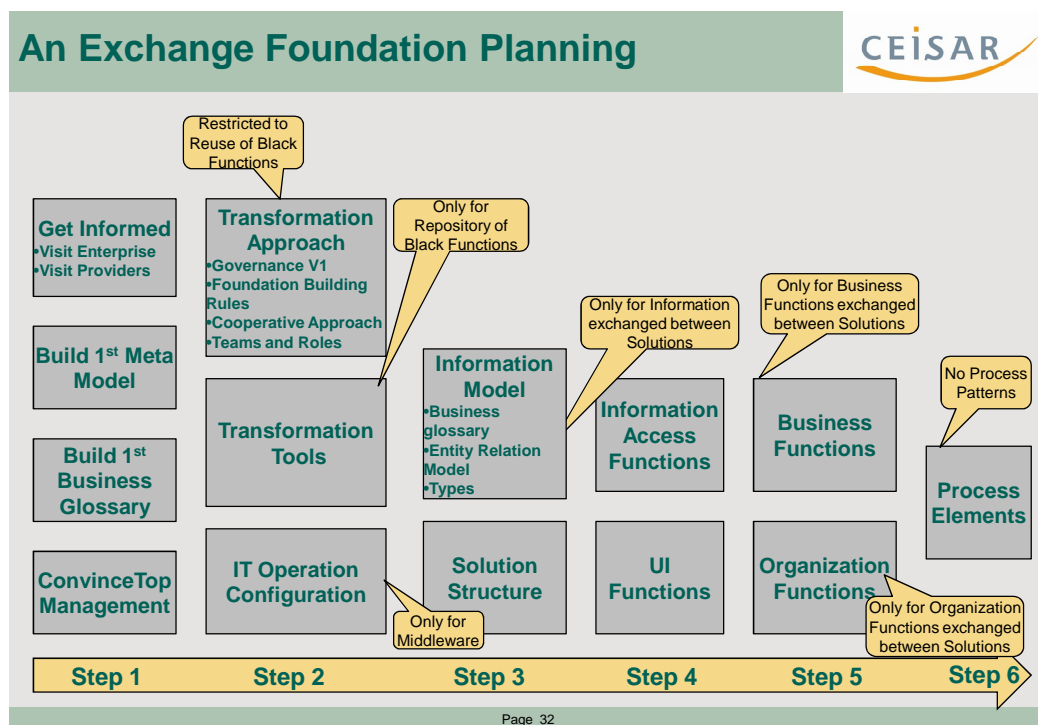
- **Enterprise Model:** Process Map, Solution Map
- **Information Model:** Glossary of Entities, Entity Map, Relations, Attributes and Types

Fourth step is to Build **Information Access Functions:** Business Object access rather than simple table access, versioning, mapping Entity/tables, Transaction mechanisms...

Fifth step is to progressively Build **Business and Organization Functions** which reuse Information Access Functions and **White Components:** inheritance of Entities, reusable Types, patterns for desktop, processes, skeleton for Software Services...

Last step is to Build **Process elements:** Process Patterns or Sub Processes which can be reused when Building Processes.

8.7.2 A determined approach for Exchange Foundation only



This planning follows the same interdependencies, except that this is simpler: no User interface Components, no Information Access Functions, no White Functions, no inheritance, no Type Functions...

Transformation Approach is limited to “How to Build and Reuse Black Functions between Solutions”.

Transformation Tools are limited to: how to help generate Interfaces for exchange Functions and repositories for Reusable Black Functions.

IT Operation Configuration is limited to Middleware which allow Solution exchanges: send/receive, synchronous or asynchronous, addressing Function, Conversion...

Enterprise Model and **Information Models** are the same.

Business and Organization Functions are limited to Black Functions exchanged between Solutions: no inherited Functions. It can be

- synchronous **Exchanged Functions:** Imaging, Security, Archiving, Editing, Call Center requests, Customer Access...
- or asynchronous Exchanged Functions like Flows between Solutions

Process elements may include Sub Processes but not Process Patterns.

8.7.3 A soft approach

When it is impossible to convince Top Management to invest in Foundation, it is not possible to apply the determined approaches just defined.

But it is possible to start the Foundation process at a slower pace.

Practical tasks could be:

- group all teams working for common good into **one Foundation team**
- define a **Business Glossary**, and a **Business Entity Model**
- select a **single Middleware**
- implement the **Cooperative approach** on some projects
- experiment with some **modern Transformation Environments**
- Carry out one or two pilot projects with an existing Foundation
- Describe Process Map and Solution Map
- Visit Enterprises which have implemented a Foundation approach

Starting these actions will help to convince oneself, to identify arguments, to build proof of concepts, to persuade internal Transformation teams, before returning to Top Management.

8.8 Foundation life cycle

There are 3 main phases in the Foundation life cycle:

- **Prototype Phase:** the objective is to prove Foundation efficiency to help make decisions on Foundation budget, Transformation organization and governance.
- **Building Phase:** the objective is to progressively Build Foundation by successive iterations. Ascending compatibility is not possible during this phase, because Foundation design will evolve several times before reaching stability.
- **Maintenance Phase:** Foundation is stable; ascending compatibility works; Solutions may take advantage of new Versions of Foundation with little effort.

8.8.1 Prototype Phase

Convincing top management that Foundation has a high long term value requires not only documents, explanations, PowerPoint presentations but also proof of concept: Build a Prototype of a Solution to prove that efficiency is there.

The proof of concept must be based on an available Foundation, but internal Foundation is not ready as we are looking for budget and time.

One way is to “rent” a Foundation to Build a prototype and obtain Budget, and then decide to Buy or Build a Foundation which can be very different from the original Foundation used in the Prototype Phase. Prototype is done to “sell” the Foundation approach and not a specific Foundation.

8.8.2 Building Phase

As for any building, errors are made when Building Components. During the Foundation Building period, the Foundation team must maintain the freedom to modify the Foundation design: a good structure of components is difficult to build and requires successive iterations. This is the **Foundation Building period**. Once the different layers are mature enough, Foundation will not evolve much. Interfaces are stable. only implementation is moving. This is the **Foundation Maintenance Phase**.

It means that there is no automatic ascending compatibility during the Building Phase. Each group of Solutions using the same Foundation version will have 2 choices:

- either consider the Foundation as a bootstrap which helps to deliver better Solutions, and **diverge**: no formal maintenance is required from Foundation team. Foundation team may help to evolve but does not guarantee full maintenance service as does a Software Editor.
- or **migrate** every 3 or 4 years from one Foundation version to another one: the effort is considerable and **no migration tools** are provided by the Foundation team

The first Customers of the Foundation must be **positive Solution teams**: first versions of Foundation will not be well stabilized, they have bugs, support is not professional, Governance is not easy to apply...

When the Building Phase ends, then generalize usage of Foundation to all Solution teams.

8.8.3 Foundation Maintenance Phase

Foundation stability can be recognized by the fact that its **Information Model** and its **Interfaces** no longer shift under pressure from the Solution builder.

Once the Foundation is stable, the relation with Solution teams is different: **ascending compatibility becomes possible**. For each new Foundation version, **a tool to migrate Information**, if really required, is provided: one can do it for Business Information but also for Configuration Information (parameters, rules built with Rule Engine). The migration work should be very light as the Information Model is stable: same Classes are offered, but some new Attributes can be added.

Patches are easy to apply if no modifications are made to Information Model and Interface Model.

8.9 Build or Buy Foundations?

The difficulty is not to Build a Reusable Component, but to Build a structure of thousands of Components which Reuse each other providing high agility, reliability, scalability, good performance, and simplicity of usage. So, building is the long way of obtaining it.

However, pieces can be bought outside, but integration is required to offer a simplified view of Foundations to Solution builders. Just adding Open Source pieces does not provide a full Foundation structure.

The best solution is to acquire an open, integrated set of Foundation elements which are customizable, extensible, and which produce Solutions on chosen IT Operation infrastructure. Providers are progressively Building such an offer: see examples of Sales Force, SAP, Wyde above.

If not found, then progressively build Foundation inside the Enterprise.

8.9.1 How to select good Foundation: prototype!

It is very difficult to verify global productivity gains from supplier's documentation, declarations or presentations: they all have very good tools and good Foundation.

But **productivity can be very different** from one offer to another one: make prototypes and compare.

Do not believe that:

- Programming only represents 20% of total costs and programming productivity is not so different from one tool/Foundation to another one
- for other activities (analysis, design, tests, documentation, change management...) productivity does not depend on tool/Foundation

But believe in:

- Finding a set of consistent tools/Foundation which cover all phases and not only programming
- if this set of tools/Foundation is "round trip", offers different views for each Actor, includes powerful Business Components, allows immediate testing, supports good Solution Architecture... then overall productivity can be very different.

Typically a Business Prototype takes 2 to 8 weeks, with 1 to 4 people: global requirements should be available before starting.

A Technical Prototype can also be Built

- to prove ability to Interface with other Solutions,
- to be compatible with standard IT Operation configurations already chosen by the Enterprise
- to support high volumes of transactions and Information

The Technical Prototype is done by another team, in parallel to the Business Prototype.

8.10 Foundation evolutions

Foundation must evolve as Solutions.

Evolution can be:

- a new Function is available, and old Functions work the same
- an old Function has same interface but a different implementation
- the structure of Foundation Functions has changed

When a new Version of Foundation is delivered, existing Solutions must take advantage of new Foundation: it means Solution modification.

- new **tests** for non regression: we must avoid regressions in Operations
- data base **migration** if Information model has changed
- **modification of Solutions** if Function Interfaces have changed

8.10.1 How to take advantage of a new Foundation?

A modification in Foundation may require some work for Solution adaptation.

To avoid having to modify Solutions for each new Foundation several techniques are available:

- use **configuration tools** for Solutions: Rule engine, workflow engine, parameters. They Produce customization which can be easily isolated from the Solution Software: new version of Foundation is easy to install
- for each Reusable Function, build a **definitive interface** even if implementation is very light in first versions. Definitive Interface enables us to desynchronize Foundation and Solution evolutions: if only Implementation changes, no modifications are required for caller Solutions, only tests must be executed.
- use **inheritance**: modifications in a father-class belonging to Foundation are automatically applied to son-classes belonging to Solutions. Be careful of impact on:
 - **user interface**: inheritance of Windows may help, but checks must be done on field overlap
 - **Information model**: if information is added or a type is modified in a father-class, it means migration work

Let's take an example of **successive implementations**. You want to provide a Security Function.

- identify the parameters on which the check must be done: Functional Domain, territory, Amount, Confidentiality Level...
- identify which answer you will get: it can be as simple as "Authorized" or "Not Authorized"
- Based on parameters and possible answers, define the definitive **Function Interface** and publicize it: all Solution Builders may use it
- develop Implementation 1 which can be as simple as "Always Authorized"
- develop implementation 2 based only on Functional Domain
- develop implementation 3 which is based on all parameters
- develop implementation 4 which adds storage of a message to management when an Actor decides to do something forbidden

Implementation 1, which does nothing, allows very rapid delivery of an Interface to Solution Builders.

Successive Implementations will be progressively delivered by the Foundation team, but Solution Models will not have to be modified.

8.10.2 Synchronize Foundation upgrades

Each time a new Reusable Function is available, naturally, Solution Builders would like to have it as soon as possible.

On the other hand, integration of a new Foundation Version is time consuming for the Foundation Team, and migration of a Solution towards a new version of Foundation is also time consuming for the Solution Team. One of our sponsors would like a 3-year interval between 2 Foundation versions!

Based on our experience, a good interval for the launch of a new Foundation Version is between 6 months and 1 year. In the meantime, we must be able to deliver patches every week or so: a patch corrects a bug, yet never changes the Information Model.

Full integration work is then required from all Solutions using Foundations before joint launching of the new Foundation with adapted Solutions. It is difficult to ask Solutions builders to execute such a task for each new Foundation Version unless it is a very **easy job**.

If it is not an easy job, we must avoid **synchronizing** Solution modifications.

9 How can current Solution teams use Foundation efficiently?

9.1 Main difficulties

Once a good Foundation is available it is difficult to convince Solution Builders to use it.

The reasons are numerous:

- Reusing Foundation means a **big change**: not only to learn Foundation but also to adapt the current Approach and to kill the “Not Invented here” syndrome.
- In a **multinational group**, great distances, different cultures and different languages do not help
- Reusing a Group Foundation means **less autonomy** and dependency on a Foundation Team which is not local:
- When there is a **merger** of 2 Companies: each company comes with its own Foundation (in this case, choose the best rather than mixing 2 Foundations)

Solution Teams will reuse Foundation if:

- they are **convinced** that it is good for the Enterprise and themselves
- the right **governance** is in place
- the right **organization** is also in place
- a new **Cooperative Approach** is applied
- they are **supported** by the Foundation team
- they **efficiently use** Foundation
- their Solution is not disturbed by **new versions of Foundation**

9.2 How to convince Solution Teams to use Foundation?

Foundation Approach **value** must be explained **by top management**.

It is important that Business and IT Solution Builders understand that the Foundation approach comes from the top management and not the Foundation Team.

Experience has shown that it is impossible to convince everyone.

- **Positive teams**: some Solution Builders will be happy and declare that they had been waiting for Foundation for years, they are the **positive teams**, they believe in the approach.
- **Negative teams**: some Solution Builders will be against the Foundation approach: they do not believe it is possible to save time and money with such an approach, they do not think that the Foundation team is capable of doing a good job, they think Solution + Foundation becomes too complex to manage, they think that they have overly specific requirements which cannot be solved with a common Foundation, they want to remain independent in their Business Unit.
- **Doubtful teams**: the majority of Solution Builders are in between: they like the idea, but do not know if it really works; they do not want to be the first to use Foundation.

Our recommendation is to start Building Foundation with a positive team, to prove to doubtful teams that it works, and then generalize to these teams. The negative teams will join once the initially doubtful teams will have used Foundation with success.

9.3 Which Governance?

Enterprise Architecture governance is required to be sure that Solution Builders do reuse Foundation.

The main problem is that it represents an effort for Solution teams:

- they must understand Foundation
- they must adopt the Approach
- they must not reinvent the wheel each time a Reusable Function is available
- they must accept to Reuse a Function even if they do not like it: they must ask the Foundation team for modifications, which takes more time than if they were doing it themselves
- they must spend some time delivering what could become a Reusable Function at the end of their Solution Project.

One way to check good usage of Foundation by the Solution team is to ask the Foundation Support team to **check Foundation usage** before definitive approval of the Solution Project. If the check is negative, Solution project decision should be postponed until Foundation team gives approval. If Foundation is not adapted, the proof must be provided by the Solution team and not the Foundation team.

9.4 Which new Solution Models must Reuse Foundation?

When a new Solution Model is required, it must Reuse the Exchange Foundation.

If a Building Foundation is also available, it must be decided if it is Reused or not.

Once a Building Foundation is available, the difficult questions are:

- When do I Reuse an already existing Solution Model (provided by a Package company or by the Enterprise) not Built with the Building Foundation?
- When do I Reuse Building Foundation (provided by a Package company or by the Enterprise) to Build the new Solution Model?

We summarize arguments in following table.

Reuse Existing Solution Model and Customize	Build a new Solution Model from Building Foundation
Commodity Solution	Evolving Solution
Small number of Users	High number of users
	Ability to move employees is important
Transformation Cost is high if: many Interfaces	Transformation Cost is high if: weak Building Foundation
Transformation Cost is low if: low customization	Transformation Cost is low if: powerful Building Foundation
Weak Business Transformation team	Strong Business Transformation team
Weak Building Foundation support	Strong Building Foundation support

Easy access to Foundation elements is a key factor.

Some of our Sponsors have defined “**Service Repositories**” which allow access to Black and White Functions in different ways (key words, hierarchy of Functions...).

When Foundation is rich (Thousands of Components), do not try to expose all Components, but progressively select the Functions really required by Solution Builders

9.5 Organization

See *Foundation Organization* above.

To summarize: a Solution team includes Business and IT Actors who are co-responsible for project success.

They benefit from help provided by a unique Foundation Support team.

9.6 Reused Foundation allows a new Approach

Contractual Approach is based on a sequential process whose main item is the “Contract” which defines all Requirements. It is adapted to **Commodity Solutions** (see White Paper on Agility).

Enterprises now require a new **Cooperative Approach** (or **Agile Approach**) for **Evolving Solutions**.

The objective is not to complete the Solution in the first version, but get it to a point where a set of capabilities can be tested and delivered to first users, and then to deliver other Versions at short intervals allowing to progressively discover needs and solve the compromise between what is desirable and what is possible: this Approach works better if a **Building Foundation** is available to guarantee Solution Flexibility and extension.

Reusing the same strong Building Foundation allows the breaking down of a large project into **smaller projects**: consistency is achieved by using the same Foundation and not by fully detailed design of the large project prior to building it.

Main differences when applying Cooperative Approach:

- Start with **Entity Model** (while present approach is based on Processes)
- design **Processes** independent from Organization
- mix Business and IT Actors in same teams
- **simplify Project Management:**
 - less Actors, less requirements, less meetings or reporting
 - more Architecture quality, more prototypes, more iterations
- progressive delivery by **iterations**

See http://www.scrumalliance.org/pages/what_is_scrum

Strong configuration possibilities mean better relations with Business

If Foundation provides strong configuration possibilities through Parameters, Rule Engine and Workflow Engine, it helps to build evolving Solutions.

Configuration change does not require expertise in Software Development: it requires understanding the Model structure. Business people may be trained to directly configure the Solution. They become able to directly modify pricing, adapt commissioning rules, create Products... It means less work for IT teams and faster reactivity: Business teams like it!

9.7 How Foundation supports Solution teams

Solution Builders will use foundation if a Foundation Support team brings them the help they need:

- Training
- Consulting
- hot line

A member of the Support team cannot answer all questions, but he can answer 80% of them: he calls on experts from the Foundation Building team only when necessary.

9.8 Solution Versions and Foundation Versions

When a first version of a Solution is Built, we try to Reuse classes and Functions as they are.

In former Versions, it can be necessary to specialize the reused class (to customize a Function or add Attributes): adding a new inherited Class is more complex than just updating an existing Class. This is why we always advise that you **create a Solution Class** which inherits from Foundation Class even if no modifications are required on first versions of Foundation.

More detailed answer

9.9 What risk when Solutions depend on Foundation?

If most important Solutions are Built with a Foundation, what is the dependency risk?

Solution Models Built with Foundations are much easier to understand because of their clean structure and their smaller size. If a Solution Architect leaves the Enterprise, it will be easier to maintain their Solution Model.

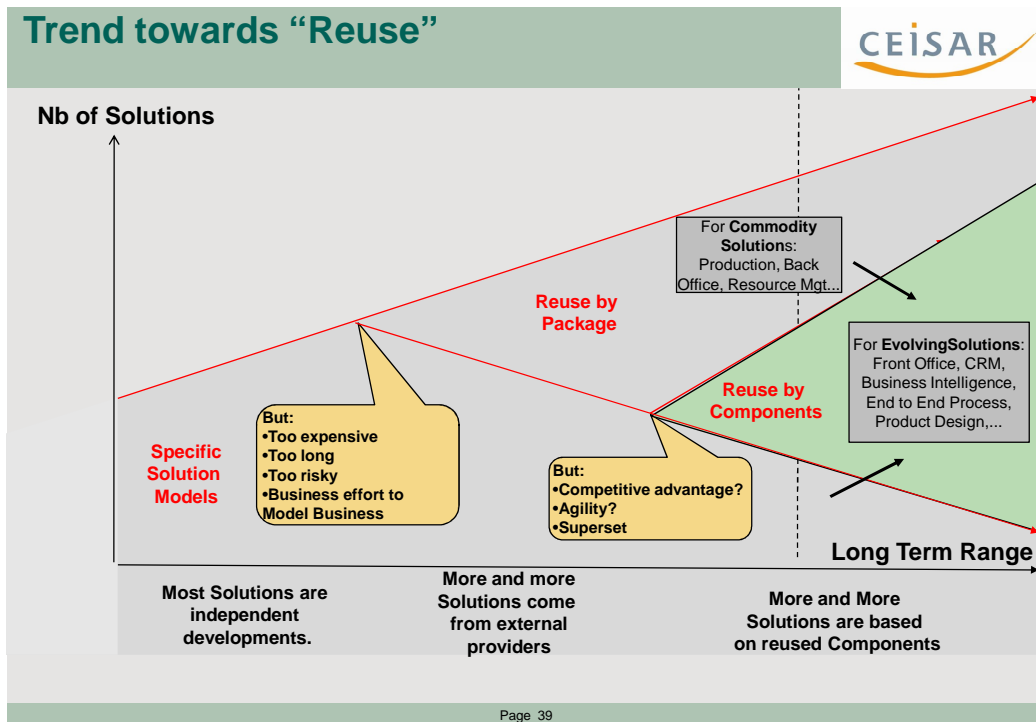
If Foundation is internally Built, it must be maintained. Foundation Architects may leave the Enterprise.

It requires checking the quality of Foundation, to ensure that documentation is clean and up to date.

If Foundation is provided by an external company, train some of your staff to be able to maintain the Foundation in case it becomes necessary one day, and obtain a legal guarantee that source code will be obtained. The real guarantee comes from good Architecture, good structure of Models, Foundation or Solution.

10 Foundation and Packages

10.1 Trend towards Reuse



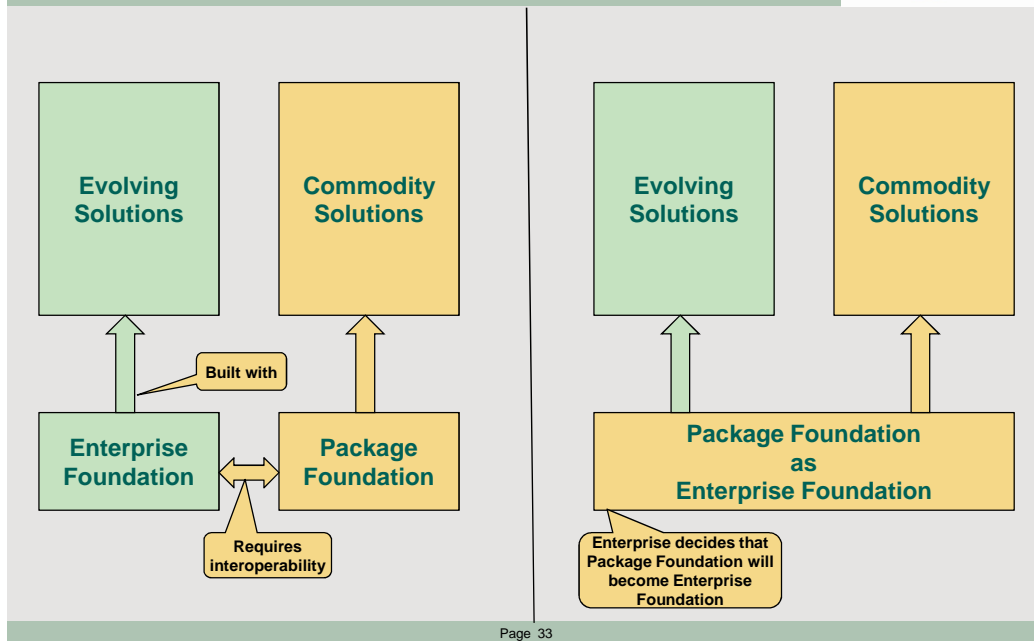
First step: Enterprises started to Build their own Solution Models. Multiplication of Solutions made it difficult. Projects became too long, too expensive and required more and more Business involvement to carefully define requirements.

Second step: a new industry of package providers (Oracle, PeopleSoft, SAP...) developed Packages for **Commodity Solutions** (ERP, HR, Accounting, Back Office...), for which requirements were close between Enterprises. It helped save time and money, requirements were ready made: Business Analysts just had to complement them.

But it appeared that this approach had reached its limits when Enterprises tried to Reuse Packages for **Evolving Solutions** (CRM, Front Office, Business Intelligence, End to End Process, Product Design...): it was difficult to obtain a competitive advantage from a Solution Model which was also available to competitors, agility was difficult to achieve when new Functions were required, and diversity of Requirements increased the volume of the Package which included requirements for all Enterprises.

Third Step: this is why “Reuse by Components” has become a new alternative for Evolving Solutions. As for “Reuse by Package”, the Enterprise does not reinvent the wheel, but it may assemble Components to Build new Evolving Solutions different from those Built by Competitors. In return, they require a Building Foundation and the related Approach. Package providers have understood this trend well and are preparing offers for Foundation (see above). The target Enterprise Architecture will be a mix of Packages for Commodity Solutions and Models Built with a Foundation of components for Evolving Solutions. The percentage will depend on the nature of the business.

10.2 Using a Package is importing its Foundation.



An external Solution has a Foundation, it can be large or small, but always exists: Operation IT Infrastructure, Middleware, Entity Models, Access functions, UI, Organization and Business Functions are part of the Package.

Package Foundation and Enterprise Foundation are different and must coexist.

Coexistence means at least that:

- **Information Model** must **map** on 4 levels:
 - Business glossary
 - Relations
 - Attributes
 - Types
- same exchange mechanisms must be used
- Package Supplier provides open Functions to **read package Information**: it allows Evolving Solutions to read Information owned by the Package
- Package Supplier provides Functions to **feed Package Solutions**: it allows Evolving Solutions to feed Information inside the Package

Some more mechanisms can also be provided, such as:

- Package Supplier provides Functions to **update package Information**: it allows Evolving Solutions to directly update Information owned by the Package
- Package Supplier enables **change implementation** of some of its **internal Functions**: it allows the Package to Reuse Functions provided by Enterprise Foundation
 - Ex: Package Solution Reuses Enterprise Repositories instead of its own files
 - Ex: Package Solution Reuses Authentication and Security Functions
 - Ex: Package Solution accepts participation in an End to End Process managed by an external Workflow engine

10.3 Can Package Foundation become Enterprise Foundation?

The Enterprise may decide that the **Package Provider Foundation becomes the Enterprise Foundation**. It is possible if

- the Package Foundation is adequate
- the Package Provider accepts to deliver its Foundation
- the Enterprise accepts to depend on the Package Provider for its Foundation

Integration of Package Solutions and Evolving Solutions is then easy because they are based on the same Foundation.

But Package Foundation is generally not delivered by Package providers for different reasons:

- Package provider wants to **protect** its software
- Foundation is **not packaged** as a Software product as the Solutions are
- **Revenue model is not clear**: customers are ready to pay for Solutions, yet they are not still ready to pay for Foundation

This may change in the future. Package Providers could propose not only Package Solutions, but also their own Foundation to customers because:

- Enterprise realizes that Building Foundation is a tough task: it costs more than they thought and takes a lot of time before reaching high Reuse. So why not to concentrate on Evolving Business Solutions **rather than Foundation**?
- Revenue on Solutions is much higher than revenue on Components. Package Suppliers do not identify high Business Revenues from Foundation. But Enterprise difficulties in Building Foundation will increase the value of available Foundation: **revenue Model** should be better for Package Supplier.
- Helping its Customers to Build its own Evolving Solutions improve the **image** of Package Solution providers who are often accused of locking Functionalities inside a closed Solution
- **Coexistence** of Evolving Solutions and Package Solutions based on same Foundation is easy to achieve

First target should be Enterprises which use many Package Solutions from the same supplier; they realize that the number of Evolving Solutions is increasing and do not want to invest in Foundation. They should naturally ask Package provider to also become Foundation provider.

The market is not mature today, but it certainly holds promise for Package Providers and Enterprises.

10.4 How to select a Package if an Enterprise Foundation already exists?

Generally, the main criteria for selecting a Package are:

- delivered functionalities
- cost and time for license, customization and deployment

Other criteria should be Enterprise Foundation **compatibility with Exchange Foundation**, at least for:

- **Entity Model**: business concepts, identifiers, relations
- **exchanges** between Package and Specific Solutions built with Enterprise Foundation
 - Information access from Package to Foundation Repositories
 - information access from Foundation to Package Repositories
- **IT Infrastructure**: must at least interconnect with same middleware.

If the Foundation team is not strong enough, the weight of availability of Functions will be stronger than the weight of easier integration or easier evolution.

One of our Sponsors has 4 different CRM Solutions. They explained that the first key choice was taken because the offered functionalities mapped with what was expected: the Foundation team explained that the selected Solution was not able to evolve without a very high adaptation cost; but these arguments did not suffice, and the product was chosen. When new CRM requirements emerged, a new Package was added because the first package was not able to satisfy them at a reasonable cost; today coexistence of 4 Solution Models for CRM with 4 different Customer files is a very cumbersome Architecture: it is difficult to make synchronized evolutions, it is far too expensive.

This kind of story is frequent and partly explains present complexity.

To avoid it:

- Define the Exchange Foundation and ask all new Solutions (package Solution or Evolving Solution) to respect this Foundation. It guarantees interoperability.
- The top management must approve these rules and their consequences on future choices
- Communication must be carried out on why these rules are good for the Enterprise

- For each key Solution decision, apply these rules before deciding on a new Package: it requires adapted Governance.

10.5 Reuse by Package or Reuse by Foundation ?

The efficiency of the Building Foundation has a strong influence on Package decision. If Building Foundation is powerful, then it can be faster and cheaper to Build a Solution rather than integrating an external Solution.

10.5.1 Reuse by Package

Any company which buys a package knows that time and money are required to integrate this package in their Enterprise Architecture:

- Package must be **customized**: by configuration (parameters, rule engine, workflow engine if they exist) or by extension
- Package must be **interfaced** to other Solutions built with Enterprise Foundation
 - Information access from Package to Foundation Repositories
 - Information access from Foundation to Package Repositories (if allowed)
 - Call of Package Business Functions by Enterprise Solutions
 - Call of Enterprise Business Functions by Package (if allowed)
 if the package is not open to exchanges, Information and Functions will be duplicated
- Users must be **trained** to use new Solution: new User Interface, new security system, new workflow, new editing system...
- IT Operations must also be **trained** to Operate the Package
- Globally the Enterprise Architecture becomes **more complex**: the package imports its own Foundation which is a complementary Foundation to manage
- Evolution of heterogeneous Solutions is more complex when they evolve: **synchronization** is difficult

10.5.2 Reuse by Components

When you Build a new Solution Reusing a powerful Building Foundation, you must pay the price of Building it, but:

- no customization is required
- Interfaces are ready made
- User training is light
- IT Operation training is light
- the Enterprise Architecture is based on same Foundation
- Evolutions and synchronization are much easier

This is why Enterprises which own powerful Foundations have a lower percentage of Packages. The trade-off between

- cost of Building Solution with powerful Building Foundation
- all actions required to integrate a Package

is more often in favor of Reuse by Foundation.

Same remark could be done with Components: if Building Foundation is powerful it will become less expensive to **Build a new Component** than **integrate an external available Component** which comes with its own Foundation (specific UI, data base, information models, types, IT infrastructure...).

11 An example of a powerful Building Foundation for an Insurance Solution

11.1 Why Foundation?

The goal of any Enterprise is to Deliver a Business **Product** for its Customer: the Product can be goods or services or both:

- for Insurance, it is the Benefit delivered to the covered Actor when a bad event happens
- for Bank it is “send money” or “get a loan”
- for Telecom it is “Call somebody” or “receive an SMS”,
- for Water utility it is “get water”
- for Car manufacturing it is “get a car” or “repair a car”
- for Transportation companies it is “transport a person” or “transport goods” or “rent a hotel room”....

To obtain Product, the Customer must Subscribe a **Contract**.

Deliver Product and Subscribe/update a Contract are defined by the Product Designer in an **Offer**.

The Core Business Domain allows **high diversity**: many similar Products, many similar Processes, many similar Functions or Windows.

This is why if you are a Group of several Companies, the difficult question is: “Must we Centralize or Decentralize the Business Model between different Companies of the Group?”

If you are a Package Provider, the difficult question is: “Can we Build a unique Model for different Business Lines and different Countries?”

Using “Exchange Foundation” it is **difficult to achieve** this goal:

- A **small number of big Black Functions** means that each of them will be a superset of this diversity: many parameters in the interface, complex implementation and high evolution. For example if you Build a unique Black Function “Compute Price” for a life insurance business which provides thousands of different products, this Function will be too big and complex to be really Reusable.
- On the other hand, Building hundreds or **thousands of small Black Functions** is difficult to manage and to Reuse. Business Modelers will be disappointed by the small granularity which does not fit with their main Business Functions.

We feel that working with a “**Building Foundation**” can be very efficient in this case.

As it is a complex topic, we prefer to give an example: an Insurance Solution Package has been Built using powerful Building Foundation. Let's explain how it works.

Same example could be Built for many other Business Domains who manage a large variety of Products (goods or services) like; Banks, Telecom, Utilities, Distribution, Media, Education, Transportation, Industry...

11.2 A Global Insurance Solution for different Companies

The Insurance Package Solution must cover:

- all Insurance **Product lines**: Property and Casualty, Life, Health, Disability, Group, Reinsurance...
- all **Processes**: CRM, Product design, contract management, claims, billing, accounting, commissions...
- all **Countries**
- all **Actors**: insurance companies, brokers, customers, prospects, partners

Some say that the Insurance Business is the same in different countries and different business lines. They would like not to reinvent the wheel: in all countries, you must prepare Offers, subscribe policies (or Contracts), bill, manage claims and manage distributors.

Some other say that Insurance products, tax systems, distribution channels, languages are not at all the same in different countries or in different business lines.

Both are undoubtedly true: we must objectively **identify what is similar from what is different** to Build a Customizable Solution.

11.2.1 What is similar: the Business Architecture

To present a simplified view of the Insurance Business: first Product Designers define the Offer, then Customers Subscribe Contract, then the Insurance Company delivers Benefits to Beneficiary when a bad event happens.

As for most Business domains (Bank, Utilities, Telecom, Distribution, Industry, Transportation...), Insurance is based on the **same Product structure** and the **same Process Patterns**. This Business Architecture must be customized Product by Product and Process by Process in each Company.

Product Structure

For the same Offer, several Benefits can be offered. For example the “Car Insurance Offer” includes benefit “Repair the damaged car” and “pay for hospital expenses”.

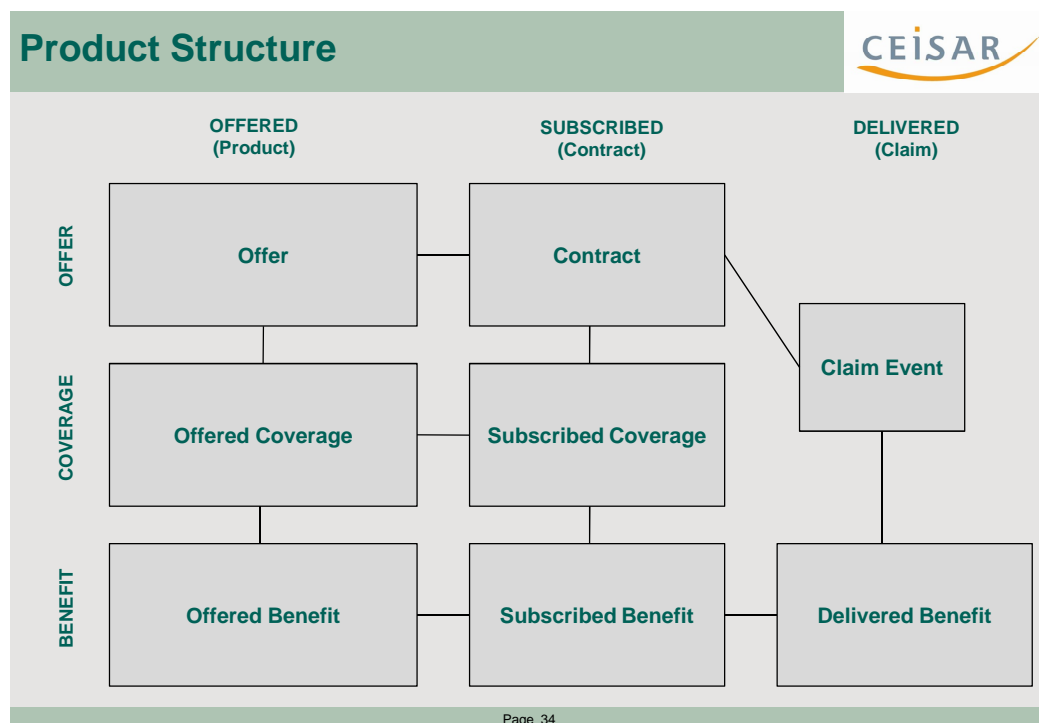
The number of Offered Benefits can be high: to simplify subscription, Benefits are grouped into “Coverages”.

So the Offer is broken down into Coverages which are in turn broken down into Benefits. The customer has just to select among a limited number of Coverages and not a large number of Benefits.

Defining these 2 dimensions

- Offered/Subscribed/Delivered (horizontal)
- and Offer/Coverage/Benefit (vertical),

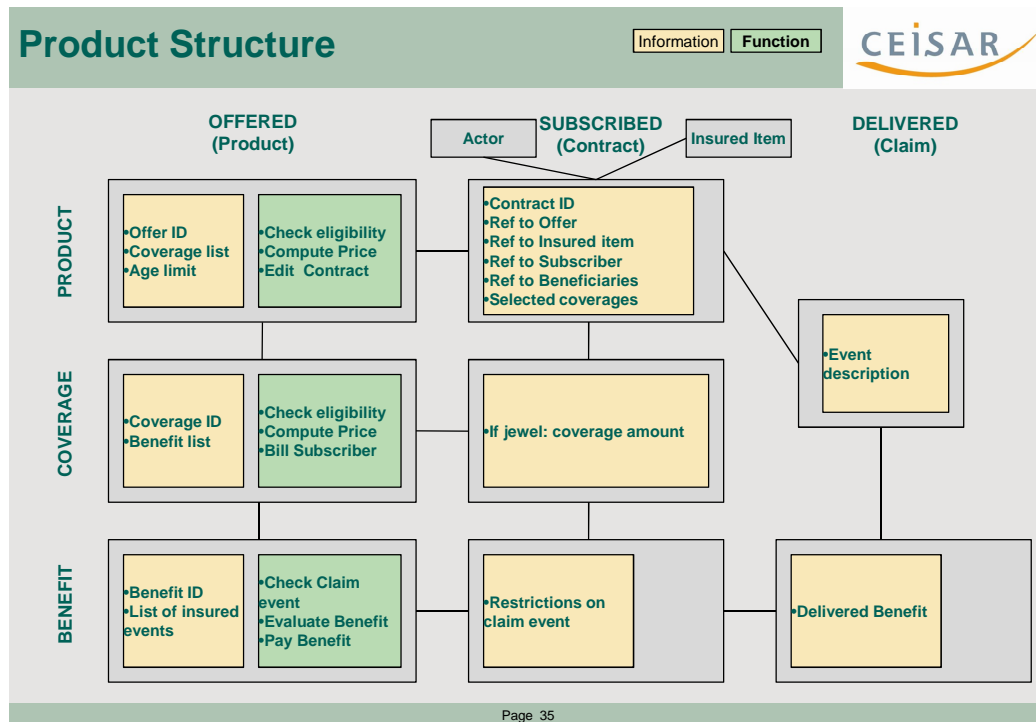
allows us to identify the following Insurance objects:



The links between objects are **Relations**: a Contract relates to an Offer, is broken down into several Subscribed Coverages and is Related to the Claim Event when it happens (to be precise we should present the UML diagram and define each detailed Relation, but we preferred to remain simple in this example). If Subscribed Coverages are not customizable at subscription time, Offered Coverages are sufficient, no Subscribed Coverage is required (same remark for Subscribed Benefit).

Fill the Product Structure

For each Product Line, we must **fill** the different boxes with **Information Attributes** and **Function categories**. We give some examples in the following slide.



Information Attributes:

Let's take the example of the "Offer" Entity: **Attributes** must be added

- each Offer requires an **identifier**; you could also add an Offer Name and a Version to separate each successive image of the Product after each Offer modification
- each Offer authorizes a **list of Coverages**: some are compulsory, some are dependent, some are incompatible
- we added the Attribute "Age Limit" just to illustrate that some specific Attributes will be added

Do the same with each Insurance Entity.

Note that some Entities like "Contract" require relation with other Entities like: Subscriber, Beneficiary, Insured Item (Good or Person). Add these Entities each time they are required.

(each Attribute should have a Reusable "Type" as defined above).

Functions:

The Product designer defines all **categories of Functions** and classifies them, such as:

- "Check eligibility to Subscribe": based on Information belonging to Item, Subscriber, Beneficiary; it can be done at the Offer or at the Coverage level
- "Compute Price"
- "Evaluate benefit": it is defined by Product Designer at "Offered Benefit" level, but in some cases it can be Customized by the Distributor at "Subscribed Benefit" level.

For each Function, check if the Information is available: if not, add it to the right Insurance Entity.

Process Patterns

Identify main Insurance Process Patterns and classify them into Transformation Processes which modify the Model and Operation Processes which Operate the Model. For example:

Transformation Process Patterns

- Build a new Offer
- Modify an Offer
- Change the Price

Operation Process Patterns

- Quote

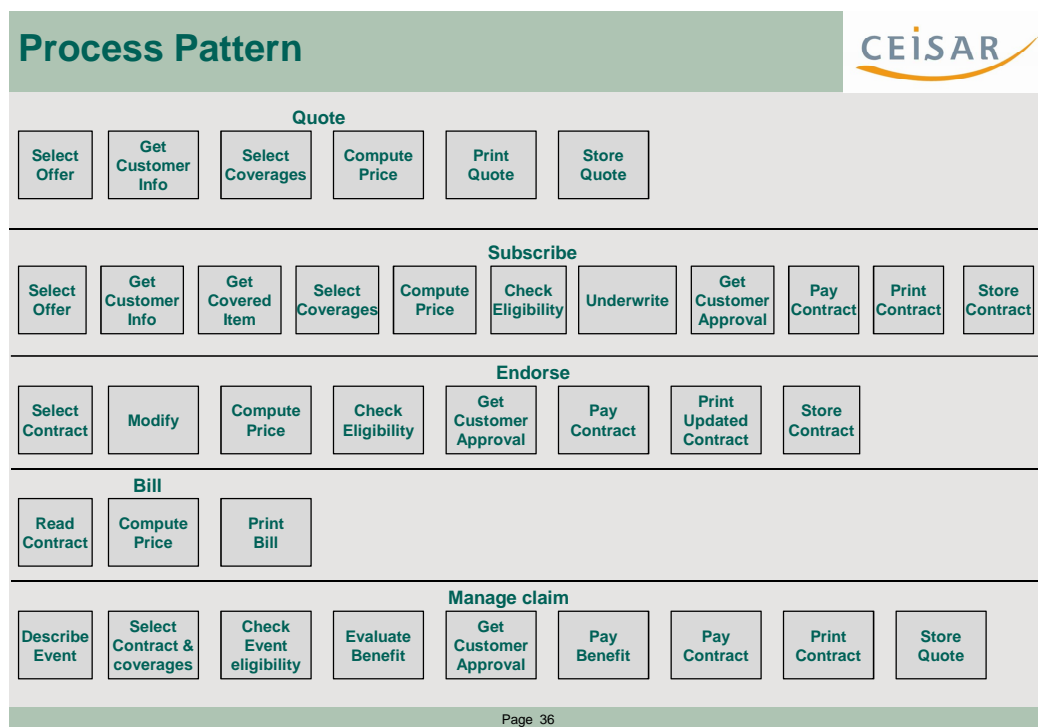
- Subscribe a Contract
- Modify a Contract (“Endorsement”)
- Bill
- Manage a Claim

Each Process Pattern chains Activities in the same order, except that the contents of each Activity is different.

For example, as presented in the following slide, each Subscription Process chains Functions like:

- Select Offer
- Get Customer Information
- Get Covered items
- Select Coverages
- Compute Price
- Check Eligibility
- Underwrite (which means: ask experts to check eligibility if risk is high)
- Get customer Approval
- Pay
- Print
- Store

To simplify the presentation, it is presented as a sequential chain, while there can be some more complex navigation between Activities.



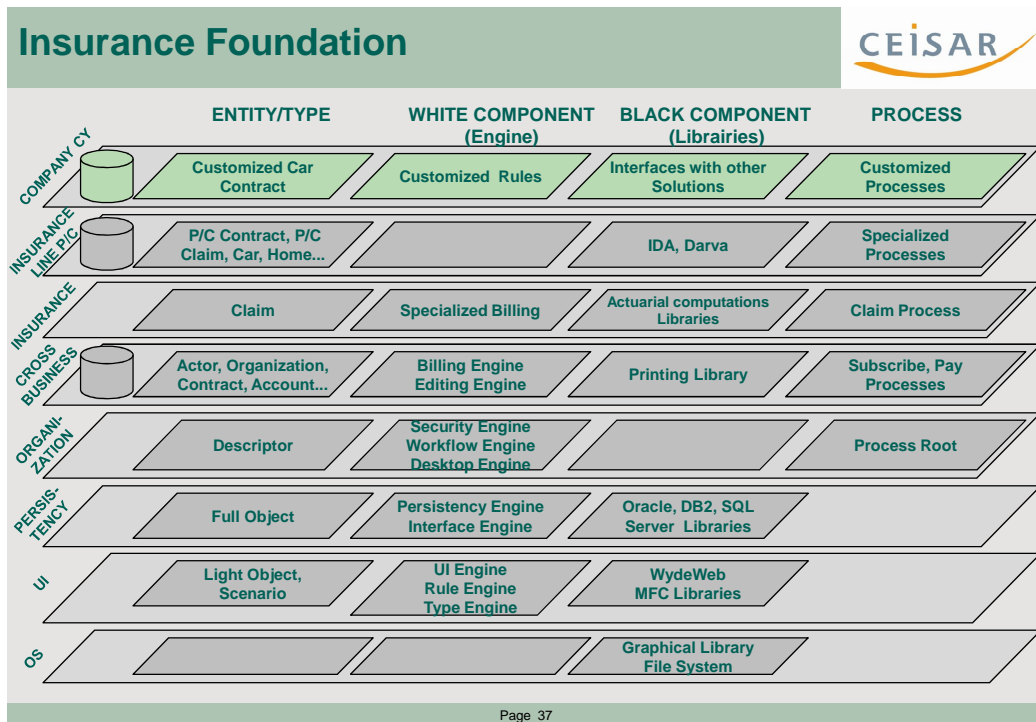
11.2.2 What is different

- **Products** are different: different coverages, different rules requiring some additional information
- **Processes** are different: Process Patterns are the same but assignment of Activities to Actors are not the same: it depends on the Organization
- **Context** is different: Actors (Users or Customers) use different languages, different currencies, different security functions, different user interface...
- **Existing Solutions** to which the new Solution must interface are different

Once similarities and differences are well identified at business level, it becomes possible to Build a Foundation to support similarities on the one side and to allow customization on the other.

11.3 Build the Insurance Foundation

It is a Business Approach more than an IT approach.



7 layers of Foundation have been progressively Built from Technical layers to Business layers. We present them starting from technical layers.

- **OS** layer is delivered by the OS Provider
- **UI** layer delivers the UI Engine: manages Reusable GUI elements, offers inheritance and Composition mechanisms
- **Persistency** layer delivers the persistency engine based on Data base libraries: it manages Relations, versioning, maps object/table and Offers Functions to Read, Write, Delete Business Objects...
- **Organization** layer delivers the Security Engine and the Workflow Engine; it also includes Functions to manage Entities (called "Descriptors" in this slide).
- **Cross-Business** layer delivers Business Models Reusable for all economic activities: Entities like: "Human Actor", "Computer Actor", Contract, Account: it includes Business Entities, Access Functions to Entities, Processes to Manage these Entities such as "Subscribe Contract", or "Pay a Bill", "or "Print".
- **Insurance** layer includes everything which is specific to Insurance Business and not already Provided by the Cross-Business layer: like Claim management, Billing Specialized to Insurance business
- **Insurance Line** layer: different Product Lines exist in Insurance like "Property and Casualty" Insurance (insure goods such as Car or Home), Life Insurance, Health Insurance, Group Insurance... In the slide, we just represented one line of Business: P/C. This level defines Entities, Functions and Processes for each of these levels;

The last green line represents what must be done for a Company which Reuses this Foundation

11.4 How each Company Reuses the Insurance Package to Build its own Model

11.4.1 Add Information to Entities, if necessary

Check if more Information is required on existing Entities.

There should not be many. It generally occurs for side Entities like Actors or Insured items, more than for core Entities like Contract or Claim.

If **Dynamic Attributes** are allowed, the Business Modeler may use this mechanism directly. Via configuration it becomes possible:

- to add Attributes,
- to automatically increment the UI with these new Attributes
- to make these new Attributes available for Rule Engine usage

11.4.2 Select Implementations

Some Functions have one Interface but may propose different Implementations because requirements are not the same in different Companies. This is the case, for example for:

- Security Function
- Editing generation Function: define Model, send an XML flow
- Workflow mechanisms such as “by Actor” or “by team”
- Desktop Model
- Error management
- Batches logs
- Business intelligence target
- Types: selecting standard presentations for types like dates, name, addresses...

The Package Solution proposes different Implementations and allows each Company to select the ones it prefers.

11.4.3 Customize

If proposed implementations are not sufficient, the Company requires customization. The customization **does not change the global structure**: it is just another implementation of some layer parts.

-Customization can be achieved by **Building new Implementations**: Security, Editing, Business Intelligence...

-Customization can also be achieved by **Building Interfaces** with existing Solutions: Security, Printing Engine, Accounting feed, may all require **Interfaces** to external Solutions.

Enterprises, today, do not replace all Solution Models at once, they replace Solutions progressively. For example, an Enterprise may decide to replace its Claim Solution only. The new Claim Solution must get information on Contracts by interrogating the old Contract Solution: it is carried out through an Exchange Function (Black Function). If a large variety of Products is used, many Exchange Models are required: a lot of work for Interface team. Each new interface requires work on the new Solution side and on the old Solution side.

The semi-interface on the new Solution side can be automated by just mapping Attributes of the Offer Model with Attributes defined in the Interface.

If, one day, the Enterprise decides to replace its old Contract Solution Model by a new Contract Solution Model based on same Insurance Model, then integration of Claim Solution and Contract Solution becomes obvious: Interface work is no longer required.

-**Customize Processes** if you want to add Activities.

Process Patterns are built by splitting the different Activities: remember that Activities are defined as a subset of Process Rules which are always executed by a single Actor.

11.4.4 Model each Insurance Product

“Creating a new Offer” is a Transformation Process which helps the Product Designer to Model its new Product: identifier, name, coverages, eligibility rules, pricing rules, benefit evaluation rules... are stored in a static catalog and Reused by the Foundation.

Once the Product Model is ready, there is **nothing more to do**: deliver the new Product Model to IT Operations, and **all Processes become available to Operation Actors for this new Offer**. There is no specific Contract or Claim to Model for each new Product. There is nothing magic: it is just the elegance of the Insurance Architecture which executes the same processes and Reuses the same Entity Models with a different context of Rules. Rules are then considered as Information stored, and retrieved as any Information to be executed.

There is no software development required for new Products except specific static Rules: time to market becomes really different. Product designers follow the specific Process which guides them in Building the new Product, asking to define Rules at the right level each time it is required by the Model. This Transformation process is based on “Configuration” (parameters and Rule Engine): it can be executed by someone who has no Software Development experience, but understands the Business structure.

Classification of Rules by level (Offer, Coverage, Benefit) and category (Eligibility, Pricing, Evaluation...) guides the Designer

The tool should present **only useful Attributes** to the Rule Modeler and not thousands of Attributes.

The tool should also allow him to

- Build parameter **tables**,
- **Reuse** Rules,
- **check syntax**,
- **test** a rule immediately

11.4.5 Model parameters for assignment of Activities

Assignment of Activities to Roles can be automatically achieved by parameters managed by the Rule Engine: just playing with parameters allows you to offer different Organization scenarios.

Each Actor Role is defined by their Rights (for security) and Duties (for assigning Activities).

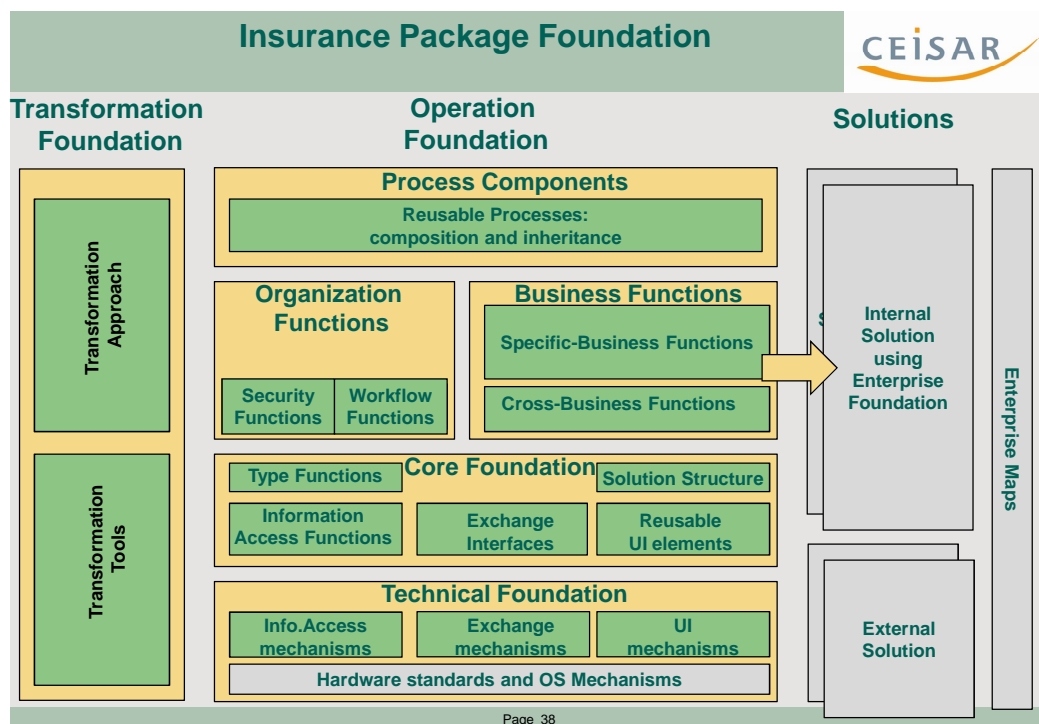
By matching Duties and context of the Activity, it becomes possible to dynamically assign Activity to the right Actor.

To summarize:

- The Company **customizes** the new Solution
 - adds Attributes if really necessary
 - selects preferred implementations
 - adds implementations if really necessary
 - Builds Interfaces to legacy Solutions
- Models hundreds of **Insurance Offers** just by using configuration (parameters and Rule engines)
- uses the **workflow** engine to Model Organization Scenarios

11.5 What is the final Structure?

11.5.1 Which elements of the Building Foundation are present?



In this Solution Model, almost all elements of a Building Foundation are Reused.

11.5.2 Transformation Tools

To succeed implementation of such an Insurance Architecture, a set of Transformation Tools has been used which are all integrated around a single Meta Model (about 1000 classes):

- UML design tool
- Software development tools
- Software debugging tool
- Integrated Rule Engine
- Process Building and Workflow engine to assign Activities to Actors
- Software Configuration management
- persistency Broker
- User Interface Building tools
- Portal Building tools
- Multi Enterprise mechanisms: language, currency...
- Tuning tool
- Solution Interface Building tools
- Migration tool
- Tools to retrieve Foundation elements
- stress testing tool
- Test Automation tools
- Software quality analysis
- workgroup tool

They use powerful Features such as:

Information Access mechanisms

- UML modeling tool
- access Function to Business Objects (via OQL) on different DBMS: DB2, Oracle, SQL Server
- versioning
- Functions to navigate through Relations are directly available by language
- replication Functions
- Dynamic Attributes

Information Access functions

- a Cross-Business Information Model has been designed: Actors,
- an Insurance Model has been defined
- Reusable Access Functions are delivered to all Business Entities

Exchange Mechanisms

- interface with any standard middleware: MQ, WebSphere
- Interface builder: for mapping, conversion

Exchange Interfaces

- adapters are provided with standard external Solutions such as: LDAP, standard printing systems...

UI mechanisms

- inheritance and composition of UI elements
- light client tool

Reusable UI elements

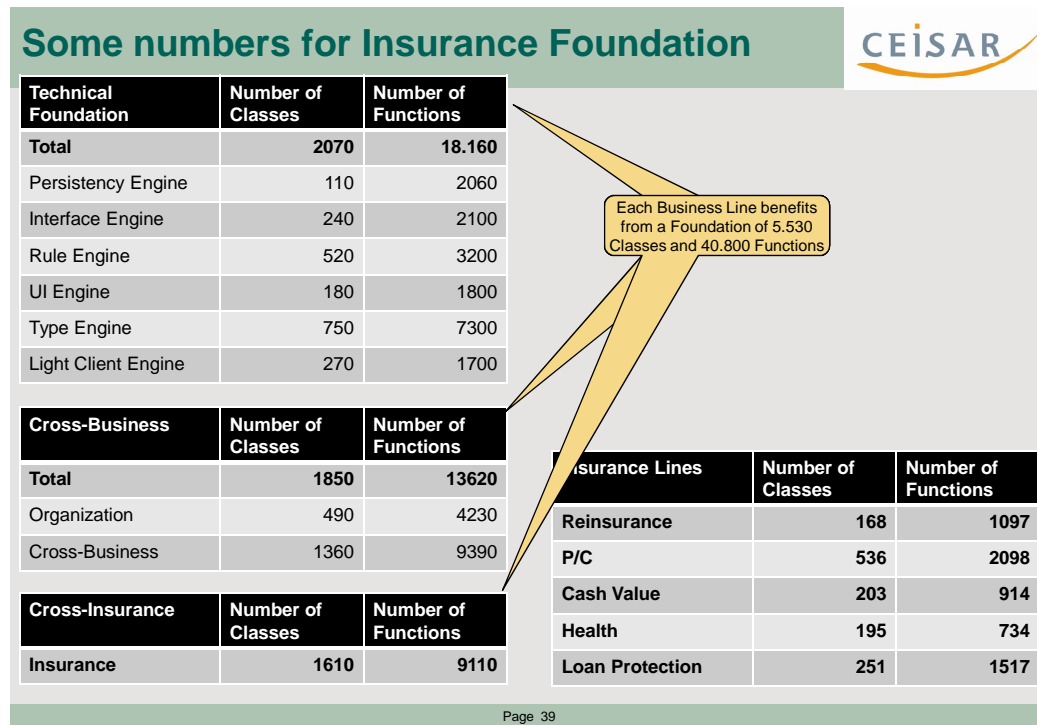
- Reusable Type UI
- Customizable Desktop
- all Reusable Classes offer their UI elements to sub-classes

Type Functions

- **various Reusable Types:** date, time, amount, name, address, text, table, hierarchy...
- **for each Type:** definition, internal format, external presentations, attached Functions

11.5.3 Size of the Solution

The next slide gives some numbers by layer: number of Classes and number of Functions. Building Foundation, including Business Lines, has 6,900 classes and 47,000 Functions which reuse each other.



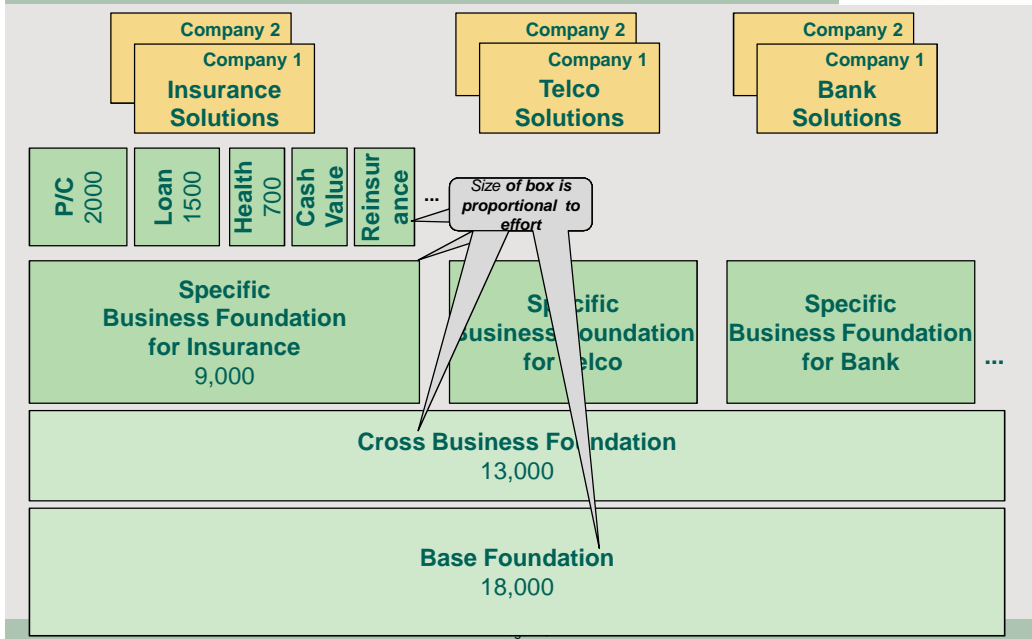
11.6 Value of Building Foundation

In this case, a Solution Package has been Built in such a way that adding new Business Lines is relatively simple: a 25-person team of experts have Built and maintained this architecture. This is a good architecture for a package provider which prefers to have one Solution Model for many Business Lines. One important thing these numbers tell us is that it costs from 1,000 to 2,000 Functions for a new Business Line, which is **5%** of the Building Foundation investment.

Installing the Solution Model for one Company has a cost: choose among implementations, build interfaces with Legacy Systems, model products are all necessary activities. But the total cost to deploy the Solution is much lower than with classical tools because most of the Customization is done through Configuration.

In return, it is necessary to Build or Buy a Building Foundation. Building such a Foundation is difficult and requires experts who have already Built such Foundation.

Large parts of Foundation are Reusable by different Businesses (nb of Functions)



In the end, Building Foundation represents the majority of the Model: Models for Business Lines like P/C (Property and Casualty) are small. It means a large Building Foundation team and a small Business Line teams.

Another interesting consequence is that lower layers can be Reused in business domains other than Insurance. Layers up to Cross-Business have been Reused for the Telecom industry and Manufacturing.

12 Exhibits

12.1 Exhibit “List of Questions on Foundation”

After discussion with its Sponsors, CEISAR classified their main questions into 4 categories:

- **What** is Foundation?
- How to **decide** Foundation?
- How to **build** and maintain good Foundation?
- How to efficiently **use** Foundation?

12.1.1 Questions on “What is Foundation?”

- Foundation definition and classification
- The Foundation perimeter will not be the same for an Industry like Chemicals or a Service activity like Finance. Is it possible to define this perimeter according to Enterprise Model strategy?
- Which frontier between Group and Company Foundations?
- What does Foundations become when using packages?

12.1.2 Questions on “How to decide Foundation?”

- What works today in various industries? Which elements of Foundation are mature and widely deployed across Companies? Which elements are not deployed so much?
- How to measure Foundation Value? Which criteria to measure Foundation efficiency? Is it Reuse, Flexibility, Productivity, Modularity?
- How to obtain budgets for "Foundation"?
- Who is the Foundation owner?

12.1.3 Questions on "How to build and maintain good Foundation?"

- How to manage Foundation risks? How to find relevant parts in Foundation?
- Foundation: a list of independent elements, or an integrated structure?
- How to ensure upwards compatibility?
- Parallel evolution of Foundation layers: ascending compatibility?
- Parallel evolution of Solutions and Foundation: ascending compatibility?
 - Scalability
 - Granularity
 - User interface inside or outside Foundation?
 - Which planning for Foundation
 - Do we buy or build Foundation?
 - Best of breed or **minimum providers**?
 - How to build multi-Enterprise foundations?
 - What does Foundation become when using Packages?

12.1.4 Questions on "How to use Foundation?"

- How can project teams efficiently use foundation: governance, change management?
- What is the adapted Approach?
- How to certify Foundation usage by Solutions?
- What is needed from Foundation Support?
- Reuse by package or reuse by Foundation?
- What is the impact of Foundation on Package selection criteria?

12.2 Exhibit "CEISAR Terminology"

CEISAR uses the same wording in all its white papers. We here summarize the main definitions. If you are interested in more detailed definitions, go to white paper on www.ceisar.org

The goal of an **Enterprise** is to deliver a Product (Goods or Services) to its Customer.

An Enterprise groups all Actors under a single responsibility.

Governmental agencies, Universities, research centers, associations also are considered as "Enterprises".

In a **Group** of several **Companies** (or **Business Lines**): the Group is an Enterprise and each Company is also an Enterprise.

To achieve their goal, **Actors** execute **Actions** with **Information**.

Actors can be **persons** (like employees, consultants, partners, customers...) or **computers**.

Manual Actions are executed by persons while **Automated Actions** are executed by computers.

When the real world becomes too complex, it must be modeled.

The **Enterprise Model** is a simplified global view of the real world which helps us to understand and react: it includes Entity Map, Process Map, Function Map, Solution Map.

The **Detailed Enterprise Model** describes

- Actor Model: Roles for Human-Actors, Configuration for Computer-Actors,
- Action Model: Procedures (for Persons) or **Software** (for Computers)
- Information Model

An Enterprise executes Operations and Transformation.

Operations means day to day Activities according to the current Enterprise Model.

Transformation means building, updating and deploying the future Model (the "Projects"):

- **Transformation Engineering** is building the Model
- **Transformation Management** is managing the project (planning, resources, exceptions...)

Enterprise Architecture describes how Actors execute Actions, with Information, in Operations and Transformation: the full scope.

A **Solution Model** is a consistent list of Action Models: CRM Solution, HR Solution, Pricing Solution. The Solution includes Process, Organization and Software: this is why we prefer to use the term "Solution", rather than "Application".

Commodity Solution: all requirements can be defined before building the Solution Model (it requires **Contractual Approach**). See CEISAR White Paper on Agility.

Evolving Solution: all requirements cannot be defined before building the Solution Model (requires **Cooperative Approach**)

Business Process: Chain of Actions delivering Value to the Process Client and triggered by a Business Event (sell, produce, manage...)

Organized Process: a scenario of a Business Process for a given organization triggered by an independent Event

Activity: set of Functions executed by same Actor at same time inside an Organized Process

Function: elementary Action with Interface; may call other Functions.

Business Function: required for Business purposes like compute Price, Print, checks

Organization Function: only useful for Organization purposes like authorize, find next actor, add to To-Do list

Reused Solution Model: Solution Model reused by different Enterprises (provided by a Package supplier or by the Group for its Companies)

Customized Solution Model: Reused Solution Model, but authorizes customization

- by **Configuration**: parameters, dynamic Attributes, Rule Engine, Workflow Engine
- by **Extension** (inheritance): allows splitting of maintained parts

Shared Solution: Solution Operated for several Enterprises; implies that these Enterprises Reuse the same Solution Model.

Foundation : Reusable Models

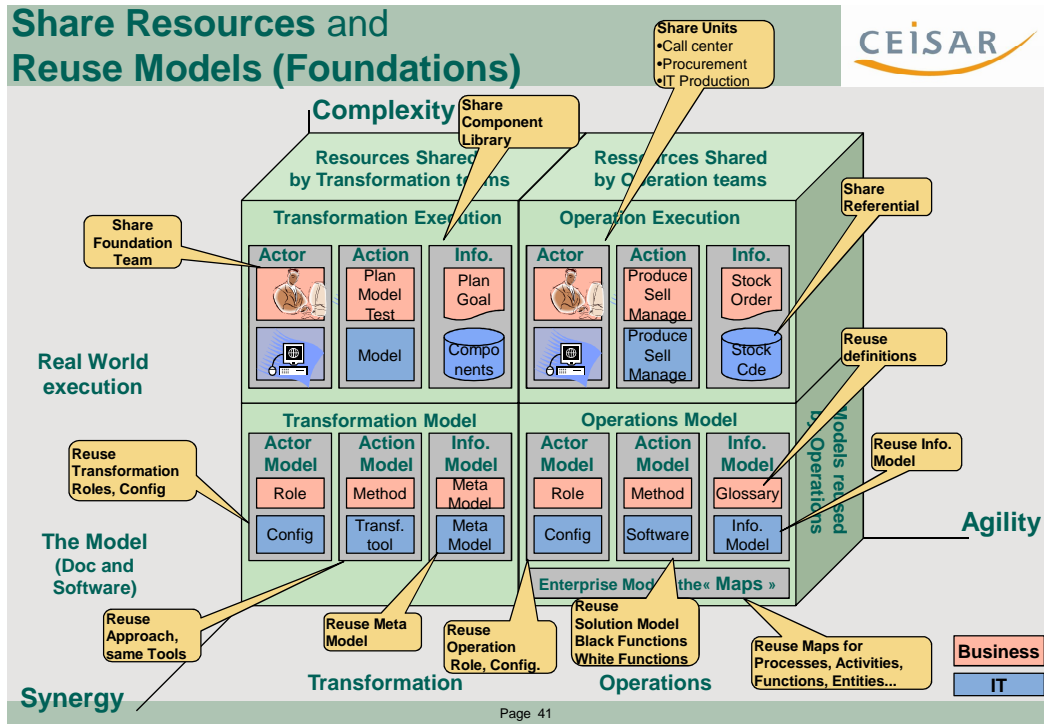
Operation Foundation includes

- Enterprise Model
- Reusable Operation Information: Glossary, Master Information Model
- Reusable Operation actions: Business process patterns, Reusable Functions (SOA or not)

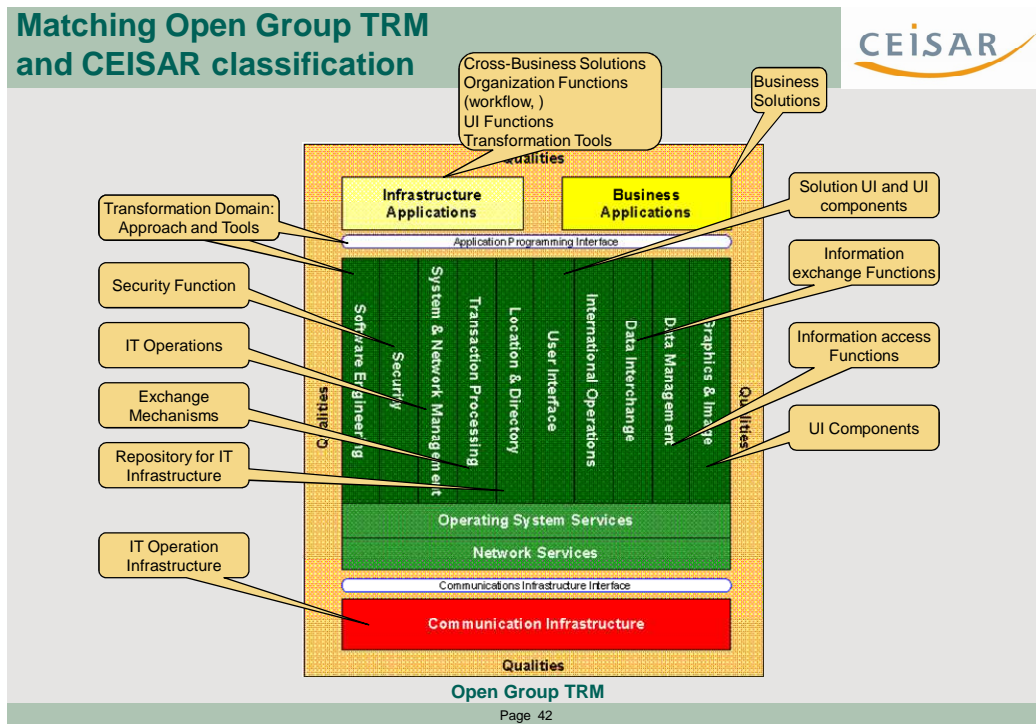
Transformation Foundation includes Approaches and Tools reused by Transformation teams.

12.3 Exhibit “Detailed CEISAR Cube”

A more detailed figure can be proposed by introducing Actors, Actions and Information, for Business and IT.



12.4 Exhibit “Togaf Technical Reference Model”



Togaf defines Application Platform Service Categories.

We have matched them with CEISAR Foundation Classification (in green, below).

- Data Interchange Services correspond to
 - **Types Functions** for data and typing services, text processing Functions, Document Processing Functions, Video and Audio Functions
- Data Management Services correspond to
 - **Information Mechanisms** for DBMS, OO DBMS, File mgt systems, Warehousing Functions
 - **UI Functions: for Screen generation Functions**
 - **Exchange Mechanisms** for Networking, concurrent access Functions
- Graphic and imaging Services correspond to
 - **UI Functions** for graphical object mgt services, drawing services, imaging Functions
- International Operation Services correspond to
 - **Transformation Tools** for Character sets, local language support services
- Location and Directory Services correspond to
 - **Organization Functions** for Directory Services, service location Services,
- Network Services correspond to
 - **Exchange Mechanisms** for Data Communication Services, remote process services
 - **Solutions** for Electronic Mail Solutions, Video Conferencing
 - **Information mechanisms** for Distributed Data services
 - **Business Functions** for mailing list Functions
- Operating System Services correspond to
 - **OS mechanisms**
- Software engineering Services correspond to
 - **Transformation Approach and Tools**
- Transaction Processing Services correspond to
 - **Exchange Mechanisms**
- UI Services correspond to
 - **UI Functions**

- Security Services correspond to
 - **Security Functions**
- System and network Management Services correspond to
 - **IT Operation Solutions**
- Object Request Broker Services and Common Object Services correspond to
 - **Transformation Approach and Tools**

Most items are the same.

Some **terms** are different:

- CEISAR uses “**Function**” rather than “Service” because a Function can be implemented or not with software.
- Togaf uses “**Service**” and sometimes “Function”

Classification is different:

- CEISAR makes a distinction between Operation Foundation and Transformation Foundation
- CEISAR makes a distinction between Solution (what is launched and used by the operation Actor) and Function (a sub-part of Solution)

Version 9 of Togaf defines the potential re-usable building blocks, but does not describe how to Build them.

Arismore (representing Togaf in France) has prepared a reference table to help communication between this White Paper and Togaf9.

Foundation White Paper	TOGAF 9
2.2 Foundation as a technique to increase Synergy	Part IV. Architecture Content Framework 33. Introduction to the Architecture Content Framework http://www.opengroup.org/architecture/togaf9-doc/arch/toc-pt4.html
3 Operation Foundation	35. Architectural Artifacts http://www.opengroup.org/architecture/togaf9-doc/arch/chap35.html 36. Architectural Deliverables http://www.opengroup.org/architecture/togaf9-doc/arch/chap36.html 37. Building Blocks http://www.opengroup.org/architecture/togaf9-doc/arch/chap37.html 43. Foundation Architecture: TRM http://www.opengroup.org/architecture/togaf9-doc/arch/chap43.html
4 Transformation Foundation	Part VII. Architecture Capability Framework http://www.opengroup.org/architecture/togaf9-doc/arch/toc-pt7.html 51. Architecture Maturity Models http://www.opengroup.org/architecture/togaf9-doc/arch/chap51.html
4.1 Reusable Roles for Transformation	35.4 Views and Viewpoints http://www.opengroup.org/architecture/togaf9-doc/arch/chap35.html#tag_36_04
4.4 Transformation Engineering Tools	42. Tools for Architecture Development http://www.opengroup.org/architecture/togaf9-doc/arch/chap42.html
3.1 Exchange and Building Foundation	Part V. Enterprise Continuum and Tools http://www.opengroup.org/architecture/togaf9-doc/arch/toc-pt5.html 38. Enterprise Continuum – Introduction http://www.opengroup.org/architecture/togaf9-doc/arch/chap38.html 39. Enterprise Continuum http://www.opengroup.org/architecture/togaf9-doc/arch/chap39.html 44. Integrated Information Infrastructure Reference Model (III-RM) http://www.opengroup.org/architecture/togaf9-doc/arch/chap44.html
6.4 How to convince top management?	24. Stakeholder Management http://www.opengroup.org/architecture/togaf9-doc/arch/chap24.html 30. Business Transformation Readiness Assessment 7. ADM Phase A http://www.opengroup.org/architecture/togaf9-

	doc/arch/chap30.html
6.5 Define a Global Plan	<p>5. Introduction to the ADM > Scoping the Architecture http://www.opengroup.org/architecture/togaf9-doc/arch/chap05.html#tag_06_05</p> <p>7. ADM Phase A http://www.opengroup.org/architecture/togaf9-doc/arch/chap07.html</p> <p>Part III. ADM Guidelines and Techniques http://www.opengroup.org/architecture/togaf9-doc/arch/toc-pt3.html</p> <p>18. ADM Guidelines and Techniques – Introduction http://www.opengroup.org/architecture/togaf9-doc/arch/chap18.html</p> <p>19. Applying Iteration to the ADM http://www.opengroup.org/architecture/togaf9-doc/arch/chap19.html</p> <p>20. Applying the ADM at different Enterprise Levels http://www.opengroup.org/architecture/togaf9-doc/arch/chap20.html</p>
7 What is a good Foundation?	<p>Part V. Enterprise Continuum and Tools http://www.opengroup.org/architecture/togaf9-doc/arch/toc-pt5.html</p> <p>38. Enterprise Continuum – Introduction http://www.opengroup.org/architecture/togaf9-doc/arch/chap38.html</p> <p>39. Enterprise Continuum http://www.opengroup.org/architecture/togaf9-doc/arch/chap39.html</p>
8.4 Quality and experience of Foundation architects 8.5 Foundation Customer is required	<p>52. Architecture Skills Framework http://www.opengroup.org/architecture/togaf9-doc/arch/chap52.html</p> <p>47. Architecture Board http://www.opengroup.org/architecture/togaf9-doc/arch/chap47.html</p> <p>48. Architecture Compliance http://www.opengroup.org/architecture/togaf9-doc/arch/chap48.html</p> <p>49. Architecture Contracts http://www.opengroup.org/architecture/togaf9-doc/arch/chap49.html</p> <p>50. Architecture Governance http://www.opengroup.org/architecture/togaf9-doc/arch/chap50.html</p>
8.7 Planning for a Foundation approach 8.8 Foundation life cycle	<p>Part II Architecture Development Method http://www.opengroup.org/architecture/togaf9-doc/arch/toc-pt2.html</p>
9.3 Which Governance?	<p>47. Architecture Board http://www.opengroup.org/architecture/togaf9-doc/arch/chap47.html</p> <p>48. Architecture Compliance http://www.opengroup.org/architecture/togaf9-doc/arch/chap48.html</p> <p>49. Architecture Contracts http://www.opengroup.org/architecture/togaf9-doc/arch/chap49.html</p> <p>50. Architecture Governance http://www.opengroup.org/architecture/togaf9-doc/arch/chap50.html</p>
10 What does Foundation become when using Packages?	<p>39. Enterprise Continuum http://www.opengroup.org/architecture/togaf9-doc/arch/chap39.html</p>
12.4 Exhibit "Togaf Technical Reference Model"	<p>43. Foundation Architecture: TRM http://www.opengroup.org/architecture/togaf9-doc/arch/chap43.html</p> <p>44. Integrated Information Infrastructure Reference Model (III-RM) http://www.opengroup.org/architecture/togaf9-doc/arch/chap44.html</p>
12.5 Exhibit "Sharing" is not Reuse 12.6 Exhibit "Complement to Operation Foundation"	<p>41. Architecture Repository http://www.opengroup.org/architecture/togaf9-doc/arch/chap41.html</p>
12.7 Exhibit "Sharing Operation Resources" 12.9 Exhibit "Sharing Transformation Resources"	<p>40. Architecture Partitioning http://www.opengroup.org/architecture/togaf9-doc/arch/chap40.html</p>

12.5 Exhibit "Sharing" is not Reusing

Sharing Resources			CEISAR
	Sharing Units	Sharing IT Infrastructure	Sharing Information
Operation	Share HR, Procurement, IT Operation... Units	Share IT Operation Infrastructure	Share Operation referentials like Customer file, Organization file...
Transformation	Share Foundation Unit	Share IT Transformation Infrastructure	Share Transformation referentials: metrics, component repository...

Page 43

Solution Classification

Mixing Reusing Solution Models and Sharing Resources allows classification of Solutions according to the following table.

It helps, for example, to identify what is different between:

- B3: "Cloud Computing" means "develop specific Solution" but "let the provider Operate the Solution in the Cloud"
- C2: "SAAS" means "Reuse a Pre-Built Solution Model" and "let the provider Operate it for different Enterprises".

Solution Classification



For a Standard Company which is not a Software Editor.

Operation Model	A-Operated only for me and by me	B-Operated only for me and by others	C-Operated for all and by others
1-External Solution Model with no adaptation	Ex: Excel		Amadeus, Swift, Darva, Google Apps
2-External Solution Model with adaptations	Ex: SAP	Ex: SAP Operated by an external company	SAAS like external Payroll Service
3-Specific Solution Model internally built	Develop a Specific Solution and Operate it	Externalized IT Operations, Cloud Computing	

12.6 Exhibit “Complement to Operation Foundation”

For some Operation Foundation elements, we here give more detailed information, useful when an Enterprise is in the process of Building Foundation.

12.6.1 Reuse Access mechanisms to Information

Each Information Access Function must be Built by Reusing powerful **Access Mechanisms** which lighten the task.

To give some examples:

- Ability to change DBMS (Oracle, DB2, SQL Server...) without rebuilding the Solution.
- To be as close as possible to Business analysis, offer capacity to access a **Business Object** rather than a Table: one implementation is to use OQL as a standard language to access Objects and store them in a standard relational database. It requires a mapping Function to be able to switch Business Objects to Tables.
- Every Business Object has an identifier: Customer id, Contract id, Account id... Using a universal identifier for any instance, allows reusing of same mechanisms for Relations between Objects. Using this universal identifier does not prevent us from also keeping business identifiers used by Actors.
- Manage **Versioning**: identify the successive images of the same Object; manage historic evolutions, list and present successive versions of the same Object, compare different versions of the same Object, offer versioned relations between Instances...
- **Navigation through relations**: offer mechanisms to navigate from one Object to another through Relations, like from “Customer” to its “Contracts”, from each “Contract” to “Delivered Services”...
- **Dynamic Attributes**: availability for non IT actor to define a new attribute, modify data model, modify UI, make the new Attribute available for Rule Engine
- **Business Transaction** mechanisms: provide a Business Transaction Mechanism which reuses the DBMS transaction mechanism, and allow application of Business Transaction:
 - for different DBMS,
 - on different Servers,
 - Transactions of Transactions
- **Replication Mechanism**: provide reusable mechanisms to replicate creations or modifications of Instances copied on different Servers; subscription, identification of modifications and what must be sent, applying modification to each data base

12.6.2 Reuse Access Functions to Information

Each industry is defining its own Information Model

- ACORD (<http://www.acord.org/home/home.aspx>), for US Insurance or EEG7 (<http://www.eeg7.org/>) for European Insurance
- CIM (Common Information Model) from DMTF (Distributed Management Task Force, Inc): for Electricity and Utility see <http://www.dmtf.org/standards/cim/>
- IEC 61968 for Electric Utilities
- OAGIS: the Open Applications Group Integration Specification (OAGIS) is an effort to provide a canonical business language for information integration. It uses XML as the common alphabet for defining business messages, and for identifying business processes (scenarios) that allow businesses and business applications to communicate. Not only is OAGIS the most complete set of XML business messages currently available, but it also accommodates the additional requirements of specific industries by partnering with various vertical industry groups (see <http://www.ibm.com/developerworks/xml/library/x-oagis/>)
- Telco SID for the Telco industry

Some **Providers** have defined and sell a Model for specific Industries.

For example IBM provide IFW for the Financial industry

<http://www-03.ibm.com/industries/financialservices/us/detail/component/I803938H46550Z52.html>

Most of these industries use **Cross-Business Information** like:

- **Actor**: Person, Legal Entity, Computer
 - Reuse the same Model for Persons, Legal Entities and Computers.

- Provide components to define their Roles: customer, partner, provider ...
- **Address:** provide a Component to manage Postal Address, email, telephone number, type or class, presentation, checks...
- **Third Party Account:** provide a Model for Bills, Payments, Accounts and Accounting Lines
- **Organization:** provide a reusable Model
 - for Organization Units (direction, department, division, branch...),
 - their hierarchy,
 - up to Position (the smallest Business Unit where a single Person may be assigned)
- **Actor Profile:** provide a reusable Model for Actor Profiles: to store Rights and Duties.
- **Operation IT description:** provide a single Model to describe Computers, network, software configuration, data localization
- **Location and Facilities:** a single Model to describe Locations and facilities (Building, Offices, factories...)
- **Product:** provide a single Product Model.
 - Goods Products in Industry are broken down into parts: this is “parts nomenclature”
 - Services Products in the Service Business are broken down into “component nomenclature” (benefits, check Rules, Pricing Rules...)
- **Contract:** provide a Contract Model which is reusable by all Solutions; can be a reusable Contract header which is reused by all Contract entities

12.6.3 Reuse Exchange Mechanisms between Solutions

Generally these exchanges are classified into 3 categories:

- synchronous question-answer: to read information owned by another Solution or execute a Function which does not modify Information (like “compute a price”)
- synchronous update: to update Information owned by another Solution
- Asynchronous feed: when a Solution delivers inputs to another Solution

Each Exchange must be Built by Reusing **Exchange Mechanisms** like: Routing, conversion (using XML, PDF, flat file, mapping...) which are provided by Middleware suppliers.

Reusing the same Middleware helps Reuse of Functions.

It must be open to external reusable Functions.

It must be available on all servers.

OSOA initiative (<http://www.osoa.org/display/Main/Home>)

The Open SOA Collaboration represents an informal group of industry leaders that share a common interest: defining a language-neutral programming model that meets the needs of enterprise developers who are developing software that exploits Service Oriented Architecture characteristics and benefits. The Collaboration is not a Standards Body; it is a set of vendors who wish to innovate rapidly in the development of this programming model and to deliver Specifications to the community for implementation. These specifications are made available to the community on a Royalty Free basis for the creation of compatible implementations. When mature, the intent is to hand these specifications over to a suitable Standards Body for future shepherding.

12.6.4 Reuse Exchange Functions (the “Adapters”)

Examples of Adapters:

- Adapter to feed General Ledger
- Adapter to feed third party accounting
- Adapter to feed Business intelligence
- Adapter to feed Log, audit trail
- Adapter to feed email, SMS Solution
- Adapter to feed Word, Excel, Acrobat
- Adapter to feed Imaging Solution
- Adapter to feed Archiving Solution
- Adapter to feed Printing Solution
- Adapter from call Centers to feed Back Office Solutions

- ...

Some **suppliers** provide “Adapters” to most well known Packages like JDEdwards, EnterpriseOne, SAP Software, Oracle E-Business Suite, Siebel Business Applications, PeopleSoft Enterprise.

12.6.5 Reuse User Interface Components

We here list some pre-built UI elements which can be Reused by composition or inheritance.

Standard look and feel

Documentation of UI standards which must be applied to all Solutions.

UI elements

Use inheritance and composition to Build new User Interfaces from reusable UI elements

Desktop

Present Entry Points to Activities in one desktop tree.

It must be possible to adapt Desktop content to current user profile

Navigation

Navigation standards implemented through components to go from one Business Object to another, to select choice, to cancel actions...

Type Presentations

Define standard presentations for reusable Types like date, amount/currency, address...

Tree, Folders

Present Trees and Folders.

Multi-Language

Allows reuse of the same Solution with a different language and different character sets.

Can be defined at deployment time or dynamically, according to Actor choice.

Search on Object (Picker)

All Solutions require to search for Objects. The "Picker" is a reusable Component for any search which defines:

- search criteria,
- instance list presentation (which columns)
- authorized Action on the selected instance

Alert and warning

Generate Alert: display alert and propose answers

Comments on what really works today

Look and feel standards exist in most Enterprises.

But UI Elements do not exist: User Interface Building is still a specific task with no Reuse.

12.6.6 Reuse Organization Functions

Examples of Organization Functions:

Task Basket

Provide reusable Functions to manage list of Tasks by Actor:

- generate a Task
- list past or future Tasks by Actor or Team
- launch the right Activity from each Task

Workflow

Provide reusable Functions to manage assignment of Activities inside a Process:

- define next Activity
- assign next Activity to the right Actor

Security

Provide a single sign-on: identification and authentication.

Provide a reusable authorization Function?

Provide a Function to warn the manager?

Calendar and scheduling Functions

Provide a Function like "Update a calendar".

Business Auditability

Mechanisms to retrieve business source information

Calendar and scheduling Services

Provide a reusable Function to update a Calendar.

Comments on what really works today:

- Single sign-on is in place or will be in the near future in most Enterprises.
- Security Functions: they exist in all Enterprise, but very few Enterprises have a unique Security Function
- Workflow Functions exist in some Solutions, but are never generalized as a unique Reusable Function for all Solutions

12.6.7 Reuse Business Functions

Cross-Business Functions are the same for all Business domains like:

- Bill: Provide a reusable Bill generation Function, manage Bill Setup
- Manage third party account: Generate Accounting Entry, Update Account
- Electronic Payment: provide a Function to execute payment
- Get Customer Summary: provide a Function to present a Customer summary
- Generate Contact: provide a reusable Function to generate a Contact (when, who, content, next step...)

Model Solutions which allow different Organizations thanks to Foundation

This Topic was developed in a former white paper.

The main idea is that we should not develop a new Solution each time we implement a new Organization, like launch a new distribution channel, or decentralize back office Activities.

We should rather **Reuse the same Solution Model** and

- externalize assignment of Activities to Actors
- isolate User Interface when it has to be adapted to the communication channel.

Reusing the same Solution Models for different Organizations will decrease the number of Solutions but it requires that the Solution Model follows rules described in former white papers:

- analyze **Business Process** before **Organized Process**: split **Business Functions** that are always necessary from **Organization Functions** which depend on current organization
- Build **Organization Functions** which are **Reused by all Solutions**: Security Function, Assignment Function, To-Do list Function...
- **Externalize assignment of Activities** to Actors through Workflow Engine using Profiles of Rights and Duties

Foundation is crucial to succeeding in such an approach: it isolates Business Functions and provides Organization Functions.

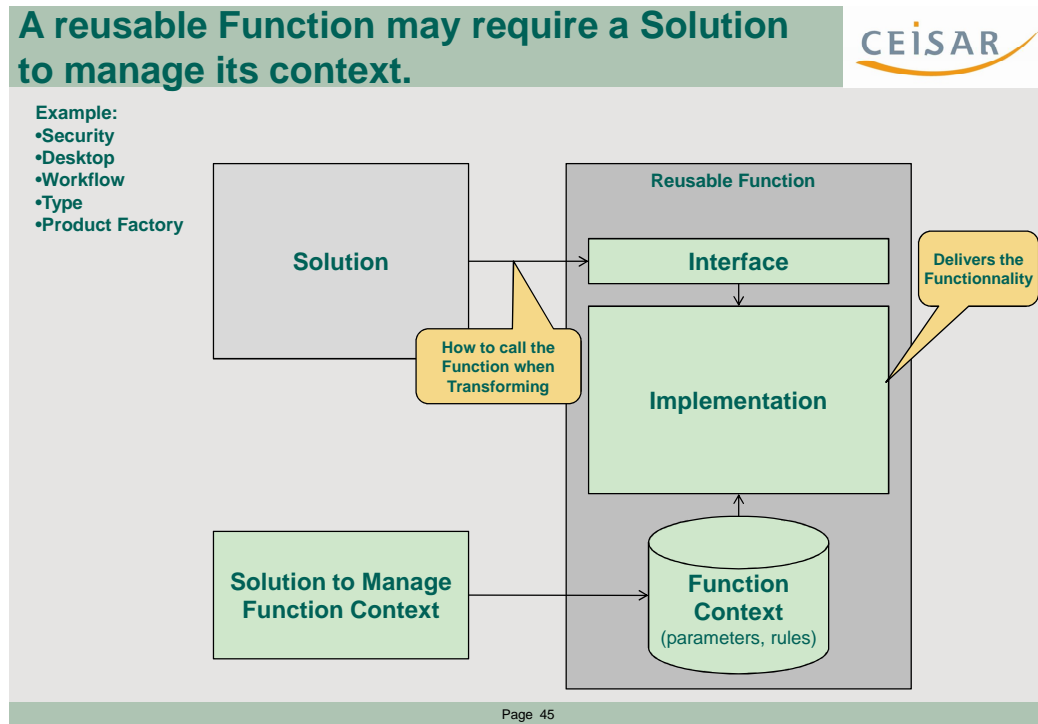
Some Providers document case studies:

- see IBM : <http://www-01.ibm.com/software/success/cssdb.nsf/topstoriesFM?OpenForm&Site=soa> or <http://www-01.ibm.com/software/success/cssdb.nsf/advancedsearchVW?SearchView&Query=%5BWebSiteProfileList>

12.6.8 Reuse Solution Models

First set of Reused Solution Models is related to **Reusable Functions**.

Building a Reusable Function is not always sufficient. If the Reusable Function requires its own specific Information to be executed, then it is also necessary to Build the Solution which manages this context information.



For Reusable Function	Offer a Reusable Solution
Authentication Function	Manage Users identifier and passwords
Authorization Function	Manage Profiles of Rights
Assign an Activity to an Actor	Manage Profiles of Duties based on Functional Domain, territory...
Text processing, spreadsheet processing Functions	Office automation Solutions like Word, Excel
Create an Image and Retrieve it	Imaging Solution
Send a Message, an SMS	Email Solution
Call an Actor on the phone	Phone Solution
Generate a Contact	Contact management Solution
Scan, store, attach and retrieve an image	Imaging Solution
Compute Price	Manage tables of pricing parameters and Rules
Compose a document	Document management Solution
Print a document	Printing Solution
Transform a Zip code into town	Manage mapping between Towns and Zip Codes
Find the currency of a country	Manage mapping between Currencies and Countries

Some Solution Models may also be reusable because requirements are the same for different companies:

- Collaborative Solutions
- Call Center Solutions
- Build and present a Customer Summary

- Product Factory: to allow Business Actors to directly assemble new Service Products without IT involvement
- HR Solution
- Procurement Solution
- Accounting Solution
- BI Solution

For each Function, must be defined:

- Function Interface: how to call it
- Function Implementation: how it is Built
- Function Context: Information related to the Function
- Solution to manage the Function Context (parameters and Rules).

12.7 Exhibit "Sharing Operation Resources"

Reusing the same Operation Models allows us to Share Operation resources.

12.7.1 Sharing Operation Units or Centralizing a Business Unit

As explained before, Reusing the same Model does not prevent each Company of a Group from being independent: **Model definition is centralized**, but Resources which operate this Model are decentralized. The key idea is that reusing the same Model brings consistency.

In a Group of several Companies, top management may think that **centralizing** HR, Procurement and Financial Control **Resources**... is more efficient for different reasons:

- Economies of scale
- Expertise is rare
- Top Management wants to directly manage certain domains

To Share a unique HR Unit which works for the Group:

- the same HR Model is imposed on all Companies
- this unique Model is Operated by a single centralized team

Organization Maps are provided to describe Organization of Units.

These Maps **are not Model Maps** like the ones described above: they just describe Human Actor Resources.

12.7.2 Sharing IT Infrastructure for Operations

The same principle may be applied for IT Infrastructure.

- **decentralized IT Operations**: the same IT Configuration Model is Reused (same OS, same hardware), but each Company Operates its **own** IT Infrastructure
- **centralized IT Operations**: the same IT Configuration Model is Reused, and all IT Operation Units are **Centralized** in a single Unit managed by the Group

When IT Operation Infrastructure is complex, Maps for IT Infrastructure are provided to present Servers, Work Stations, the network, and localization of data on servers.

These Maps **are not Model Maps** like the ones described above: they just describe Enterprise Resources and not Enterprise Model. They are not part of Foundation.

12.7.3 Sharing Operation Information

Information Repositories (or Master Data) can be Shared by the different Solutions.

Examples: Customer data base, Enterprise Organization, shared nomenclatures such as:

- Functional domains
- Territory
- Zip Codes and Town
- Reusable Profiles
- Accounting Nature
- Product Domains
- Customer Segments
- Countries, Currencies

Each time Information is Shared, it must have an owner: if several owners are necessary for a single data base, define how to identify this owner (Attribute in the Object or external rule).

Sharing can be at Group level or Company level.

To give an example, Air France decided to Share the Flight Repository which is used by many Solutions, but had difficulties in deciding who should be the owner of the Information.

A detailed classification has been defined by the CEISAR in an Excel spreadsheet reserved for Sponsors.

12.8 Exhibit "Complements to Transformation Foundation"

12.8.1 Transformation Engineering Tools

Transformation tools cover the full cycle: not only programming, but also specifications, Map modeling, design, tests, integration...

Many different tools can be reused in Transformation Processes such as:

- Enterprise Modeling: Solution Map, Entity Map, Process Map, Function Map
- Process Building and Workflow engine
- Software Design
- Software development
- Software debugging tool
- Integrated Rule Engine
- Workflow Engine to assign Activities to Actors
- User Interface Building tools
- Information Access Building tools
- Portal Building tools
- Multi Enterprise mechanisms: language, currency...
- Solution Interface Building tools
- Migration tools
- Tools to retrieve Foundation elements
- Multi team work
- Test Automation tools
- Software quality analysis
- Tuning Tools
- Reporting Configurator
- Business Intelligence Tools
- Prototyping tools
- Software Configuration Tools
- Solution Integration tools

12.8.2 Multi language

As Solution Models are increasingly used as Multi-Country Models, the Model must adapt to different languages.

We suggest the following rules:

- Build a dictionary of translated words
- Verify each individual Window: translation is automatically applied from the dictionary, but a human check is necessary: several translations may exist for same word, some translated words can be longer and require rearrangement of the windows
- Printing adaptations are done by the editing product
- For Enumerated Types like "single, married, divorced" offer a Type translation to translate once only
- Propose to rename class name and attribute names
- User messages: break them down into standard messages ("the information <attribute name> is required") and reuse class name or attribute name
- Inheritance of windows eases translation

12.9 Exhibit "Sharing Transformation Resources"

When Transformation Models are Reused by different Companies of a Group of different Teams in a Company, it becomes possible to Share Transformation Resources such as: Transformation teams, IT Infrastructure, repository for Operation Foundation, or Metrics for Transformation,

12.9.1 Sharing "Foundation teams"

A Solution Team prefers to contact only one Foundation Team. First recommendation is to **Merge all the teams which take care of "Common good" into one Foundation team.**

When this is done for each Company of a Group, it is possible to merge different Company Foundation Teams inside a **unique Group Foundation Team** if they are working on the same topics so that they do not duplicate work.

Foundation must be managed as a whole because

- its clients, the Solution Builders, prefer a **unique contact** for all Foundation problems rather than different teams such as Technical Architecture team, SOA Team, Methodology team, quality team, security team, Repository or Master data Information team.
- **Foundation Reuses Foundation:** integration of Components must be done once by the Foundation team and not by each of the Solution teams

It means that there exists a unique Foundation portfolio with a unique Governance Process to make decisions on Foundation Projects.

The Foundation team must be split into 2 activities:

- Foundation **Building:** they Build/Buy components and assemble them to offer a simple interface to Solution Builders
- Foundation **Support:** they train, coach, control Solution Builders

What works today?

Most Foundation Topics are today managed by different teams: methodology team, technical architecture team, SOA team, Repository team, quality team, security team...

But split Group level from Company level

Most of the visited Enterprises have defined Foundation teams (often called "Architecture Team").

In large Groups, **Foundation teams may** exist at different levels:

- **Group Level:** indispensable if the Group is centralized (like an Airline Group), but can also be useful for a decentralized Group which requires acceleration of its Transformation speed or reduction of its Transformation costs. Generally they split Group Solution Modeling and Group Foundation Modeling. They define IT technical standards, some basic Functions (as single sign-on, single security function), and Transformation approach
- **Company Level:** each company owns its own Foundation team which must Reuse the Group Foundation and add what is specific to the Company.
- **Business Unit level:** sometimes, in large companies, a single Business Unit defines Reusable Models for its own use. 3 levels for Foundation are difficult to manage and synchronize.

12.9.2 Sharing IT Infrastructure for Transformation

Transformation **IT infrastructure** may also be centralized for all Transformation teams: the set of Servers, work-stations, printers, networks which are used by Transformation teams can be Shared.

It may bring savings and provide easier Foundation updating.

It requires that the same IT Model is reused by all Transformation teams.

12.9.3 Sharing Transformation Information

Solution Builders must find elements of Foundation which help them to Build Solutions: a **repository** of all these **Foundation elements** is required.

For each element, it must include:

- what the delivered Functionality is
- how to use it.
- who the owner is

It does not include the implementation of each element.

If Engineering tools do not automatically Build this repository, it is the role of the Foundation Support team to organize it.

Transformation Actors can also share **Transformation metrics:**

- metrics for Engineering (Configuration evaluation: size of servers, network throughput)
- or metrics for Management to evaluate Project workload and Build Project Planning (workload estimates)

12.10 Exhibit "How to make Decisions"

Define Enterprise Goals



Enterprise Goals

Productivity

Good Service
•Reactivity
•Quality
•Comfort
•Global Service


Globally
deploy best
Products and
Processes

Better
decision
making

Agility:
enhanced
differentiation
via timing

Page 46

Which Enterprise Challenges to reach Goals?



Enterprise Challenge: How to?

Reduce
Operation
Complexity

Develop
Business
Synergy

Share and
aggregate
Information

Reduce
Transformation
Complexity

Move from
Commodity to
Competitive
Solutions

Enterprise Goals

Productivity

Good Service
•Reactivity
•Quality
•Comfort
•Global Service

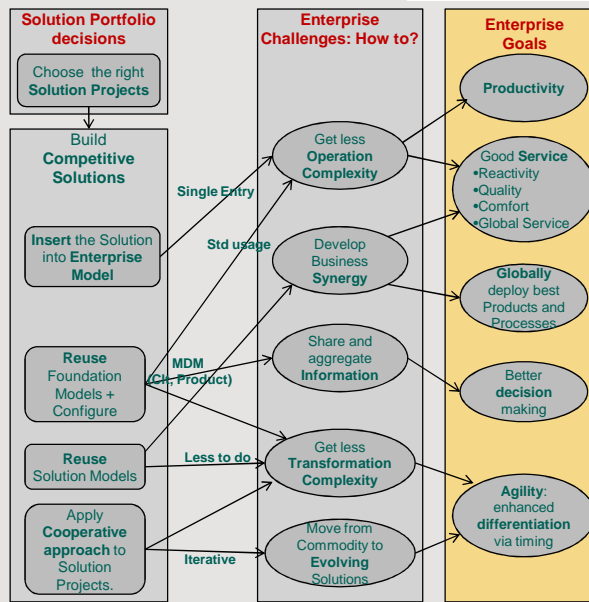
Globally
deploy best
Products and
Processes

Better
decision
making

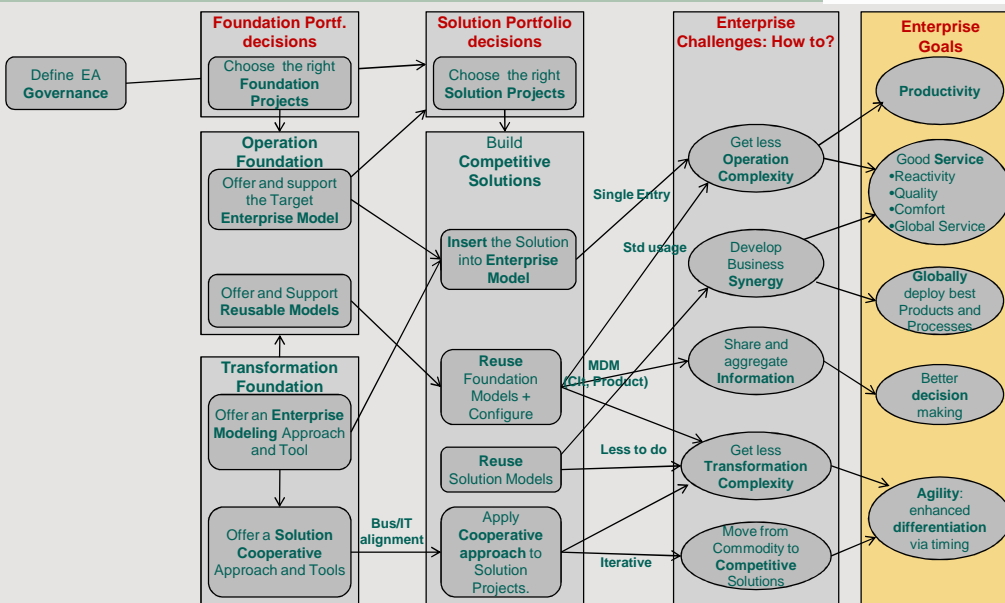
Agility:
enhanced
differentiation
via timing

Page 47

Building Solution Models to address Enterprise Challenges



To Build good Solutions, deliver good Foundations



Everything starts from clear executive decisions.

