Incremental development of a repository-based framework supporting organizational inquiry and learning

Martine Devos, IS Manager (mdevos@argo.be)
Michel Tilman, System Architect (mtilman@argo.be)

Abstract

In this paper we report on our experience with the development of a framework for administration and a supporting method for incremental development and learning.

Argo is a semi-government organization managing several hundred public schools. We coach and counsel ARGO and develop applications, which share a common business model, and require database, electronic document, workflow and Internet functionality.

The framework helps us develop applications iterative, interactive and incrementally, with strong emphasis upon the process of problem appreciation. Thus we build increasingly complete specifications of applications that can be executed at once. The resulting system is easy to adapt to changes in the business. Argo can develop new applications through modeling and configuration, rather than through coding.

The framework is based on a repository in two ways. First, it consists of a set of tools for managing a repository of documents, data and processes, including their history and status. These tools let users select applications, enter and view data, query the repository, access the thesaurus and manage electronic documents, workflow processes, and task assignments. More importantly, the repository drives the framework behavior. In the repository we capture knowledge of the business model, none of which is hard coded. The tools consult the repository at runtime. End-users can change the behavior of the system by using high-level tools to change the business model. Thus we separate descriptions of an organization's business logic from the application functionality.

Since we can afford users to change their mind, inspired by learning experience with the technology at hand and the broader range of possibilities it offers, we use methods and techniques from soft system thinking to encourage our users to reflect on their applications and to envision better ways of working, even after initial delivery. To incorporate the feedback, we adapt both framework and the applications.

1. The project

Argo is a semi-government organization that manages the public schools (non-denominational) within the Flemish community in Belgium. It consists of a central administration and a few hundred semi-autonomous local sites (boards and schools).

The project started early 1994. At the start Argo was going through an external audit, which made it likely that there would be major changes of procedures, organization structure, accountability rules and delegations of responsibility. Thus, future requests for changes were a given at the start of the project.

We decided at the onset of the project that we needed a flexible approach. So we built a framework for adaptive design and support configuration of applications in this environment with intensive use of creativity techniques in workshops.

We started with three pilot applications to provide input for initial framework requirements and design:

1. A documentation center that provides flexible search and retrieval of data and documents, such as legislative texts and parliamentary decrees concerning education. Many documents have a complex structure of interrelationships like appendixes, references to subsequent changes and juridical decrees based on multiple laws.
2. A system to support the central board's decision procedures from initial draft version to the final text and then to monitor the decision's implementation by the administration or the local boards and schools. These decisions include motions that have been passed or tabled, contracts with teachers and

changes to curricula and budget reports. The path through the administration is not clear-cut and it changes depending on parties involved or on the subject of the decision. The procedures are not always limited to a strict hierarchical structure; some involve advice by temporary workgroups or external committees.

3. An application to import, index, store and route large volumes of documents sent in by the local boards, such as attendance records, decisions, meeting notes and large addenda.

So Argo needed a framework to build these and similar applications, sufficiently flexible to cope with changing and with emerging needs and opportunities, both functional and technological, imposed or desired. It wanted an environment that not only would survive the expected reorganization and decentralization but that would stimulate users to envision better ways of working once they are aware of what possibilities the, formerly unknown, technology has to offer.

Halfway through the project the pending re-organization was carried out. Although the nature and extent of it were largely unknown at the onset, we were able to adapt the existing applications by re-configuration without additional coding. In fact we are now in charge of a change program. We use our framework and our method for incremental development to prepare the administration for bigger local autonomy and to help them shift their focus from control to support.

Since then, several new end-user applications have been delivered. We are now in the process of extending the framework; most notably to give schools and local boards access to the central data and information through the Internet. We are converting large existing relational database applications and we added web-functionality and an Internet discussion system.

These applications together with new technological possibilities are in turn major sources of requirements for our evolving framework. Within this process we further refined and re-designed many framework components.

We not only build end user applications with data, document, workflow and Internet functionality. We use the same framework to develop the tools that we use to build and manage these applications. In this way users can use the interface, familiar from their business objects, to adapt applications.

End-users at Argo are gradually taking over management and configuration of the applications from the development team. Key-users teach and write parts of the documentation. User groups organize workshops to discuss how to put the technology to better use. This process was initially driven by small teams of highly motivated pioneer-users, and has gradually started to embrace the end-user community at large. Given the high degree of computer-illiteracy within Argo at the start of the project, we feel this to be a significant achievement on behalf of the end-users.

2. Initial Project Requirements

2.1. Functional

The technology had to support applications containing a mixture of database (e.g. school database, patrimonies, personal, budget, inventory), electronic documents (e.g. educational documentation, research papers, articles, laws and decrees), workflow management (e.g. task assignments and follow up, decision routing, process management) and had to give local sites access to the central repository of data and documents.

Database

Users have to be able to enter, modify, remove, query, list, browse, report, print, import and export data. They need tools to set up appropriate views on data, and to store queries for later re-use.

Electronic document management

Users need to create, index, search, edit, annotate, print, export and process documents. These can be either electronic (such as files or E-mail) or scanned paper documents. Structured information, thesaurus keywords and full-text-indexing are used for indexing and searching. Optical character recognition enables users to extract text from scanned documents. Users need tools to manage document versions. They must be able to choose the most appropriate alternative representations of the same document

depending on environment or job at hand e.g. a document created and adapted with a word processor is frozen once it has passed the decision process and can be presented in PDF format to schools.

Workflow management

Users have to be able to plan and handle incoming tasks, to keep track of outgoing task assignments and to manage workflow processes, which can be more or less well defined or completely ad-hoc. Argo wants workflow functionality to suggest opportunities to its users, guiding rather than restricting them.

Remote access

Argo needs remote access to its applications, data, documents and processes in the central repository in several ways

- using an off-line version of the system through import / export of documents and data for employees occasionally taking work home for specific tasks like preparing a proposal for the central board;
- using the system on-line by means of a direct dial-up connection, e.g. modem or ISDN for board members and employees working remote;
- using E-mail and Web browsers to access the repository through the Internet for schools and teachers.

2.2. Integrated environment

The applications must be integrated in one environment. All applications must use a common business model. This model provides the cohesion expected in a large administration, providing formal procedures and policies (e.g. hierarchy, responsibilities, indexing vocabulary of documentation, and global validation- and authorization rules), but giving ample scope and freedom to adapt the technology to the most appropriate local or personal working practices. For some legal activities all departments have to abide by strict rules requiring a prescribed number of signatures, while less formal processes are in use for more everyday tasks.

2.3. Configuration and administration tools

There must be tools to define the object model, constraints and behavior, to configure and manage applications, views and stored queries, to define and manage workflow processes, to set up access control, to manage the database, and to configure caching and backup of documents.

At the end of the project, Argo's users need to be able to use these tools themselves. Hence the deliveries do not only include end-user applications, but also the necessary tools to empower users to model and configure applications.

2.4. Support for change

To cater for changing needs, the system has to forego hard coding as much as possible: the technology has to support iterative and incremental development through the use of prototypes. Even more, it must act as a catalyst for a learning process when introducing this new technology, thus effectively complementing a Business Process Re-engineering project at Argo.

3. Our framework approach

3.1. Rationale

Relatively unburdened by existing legacy systems, we opted at the start of the project for a mix of well-known database technology (Oracle) and advanced object-oriented technology (VisualWorks\Smalltalk). The Smalltalk environment is deemed necessary to handle intricacies and complexities of modeling enterprise-wide business automation in a generic and open-ended way. The generic requirements of the Argo-project are typical for many large administrations. Most important was the need for flexibility, as the organization was in a state of major flux at the start of the project.

Our approach aims to combine the high–level-modeling power of Case-tools with the open-ended nature of object oriented frameworks. Our approach is based on the following observations:

- Users tend to describe their requirements in terms of how they currently work. It is difficult for them to picture how new technology can or will influence their future working practices. Hence the need to

try out, to learn and to correct the software being delivered. This requires extensive prototyping and incremental delivery.

- Whereas the business model is particular to each organization at a particular point in time, end-user or configuration functionality, such as data-entry, querying, reporting and document management, is more generic.
- Extensions or enhancements to the functionality can often be made into re-usable assets, independent of the actual business model, and vice versa. In fact we used the framework to build some applications for the Belgian police, thus testing its re-usability.

3.2. Architecture

The core of the architecture is a three level repository that is consulted dynamically by two sets of tools.

Business repository and end-user tools

Closest to the user are the business repository and the end-user tools.

The framework uses a "first" repository to capture knowledge of a particular business model -- in this particular case about schools, teachers, buildings, curricula, inventory, ... -- including organization structure, roles, data, documents and processes.

And we provide the user with tools to access this repository.

This set of tools enables him to select an application, to enter and view data, to query the repository, to display, print and export the results of a query, to access a thesaurus, to manage electronic documents; workflow processes and task assignments.

This delivers the functionality demanded by the end user applications for database, document management and workflow.

Configuration tools and meta-information

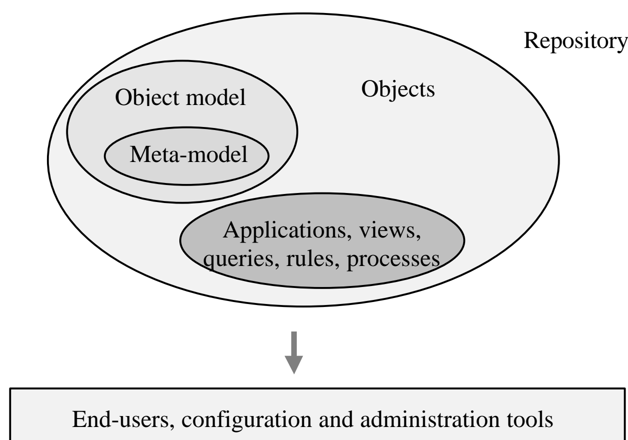
We want the end user to be able to tailor, configure and manage these applications. Therefore we provide a second set of high level tools. They define private and shared views on applications, to store queries for later re-use, edit the object model, to set up processes, define authorization and business rules, and to add or remove end-user functionality and even applications.

We don't generate the applications. Instead we use a second repository, storing information necessary to specify end user applications. This (meta-)repository contains meta-information: information about the structure of applications and business model, user views (layouts),...

Hence we effectively separate descriptions of an organization's business model from the application functionality.

Both sets of tools consult this meta-repository at run time. Hence changes made to the meta-information e.g. object structure, applications definitions, are immediately available to clients.

The tools are fully operational and they provide out of the box both end user functionality and the means to configure it. They only need the meta-information to get the system up and running.



Integration and uniform interface

To make these tools easily accessible to end-user we give them the same interface and functionality as the regular end user applications. Ideal would be to have one single set of tools, thus enhancing reuse and ease of maintenance.

Support of a bootstrapping process became an important secondary design goal: administration and configuration tools are gradually expressed in terms of the framework itself. We replace most hard-wired administration and configuration tools with configured applications. This leads to a small but powerful kernel of generic and orthogonal tools.

The meta model used to describe the actual object in turn is expressed in terms of itself and is stored in the repository too. This is a third level repository, a meta-meta-repository. This started as an experiment to push the limit of flexibility, but proved to be useful. This enables us to extend the semantics of an object model e.g. the concept of cascaded deletes is no longer hard wired in our meta model and the repository interface but can now be expressed in terms of the system itself.

In practice the three repositories are one and the same, managed by the same tools.

4. Using the framework

Starting from the model in the repository at a certain moment, we build applications by going through a set of steps. These steps can be performed in any order, even interactively. This enables the development process to unfold incrementally. Hence, we can try out and correct ideas dynamically, together with our users.

Starting with a model, which initially contains only meta-model and system objects, developing an end-user application implies a number of actions.

Extend or refine the object model.

If new structures or modifications are required, we update the model with the necessary object types, association types and global constraints. If needed, we add behavior. The object model is shared by all applications and so provides cohesion. We store this object model in the repository to make it dynamically available for the tools. This way, we can set up arbitrary business models without re-coding tools or applications. With the object model editor we indicate supertype and subtypes of a new object type and define the properties of a type, which can be either attribute or association. Object behavior and user-defined constraints may be added now or later on, e.g. to support cascaded deletes or to ensure correct initialization. Object behavior and constraints are common to all applications.

Setting up the application environment

This step starts with defining a view on the shared business model: the particular objects and properties to be accessed, created and queried. Once this basic environment has been stored in the repository and access privileges have been set, the user gets a default application that is available for immediate use. The user can log on, choose environment, enter data, query the repository, list or print results. He can keep track of task assignments, manage electronic documents and access the thesaurus. Functionality can be added or removed. The environment can be further refined for individual or shared use through views, queries and sub-environments.

Having defined the application environment, the user has access to a fully functional default application by logging on in the main application window.

The main window presents the user with the query editor and an overview list. It also provides access to other functionality, such as additional query and list windows, forms, document management, the thesaurus browser, workflow processes, in- / outbaskets, preference settings and on-line.

Users create, view, edit, print, export and browse objects through form tools. Each form is generated on the fly, according to a default strategy, subject to the authorization rules. The form generator selects appropriate editors for the individual properties and the object type at hand. In this figure we opened a form on the two elements selected in the overview list.

Layout editors enable users to define new forms and overview lists for private and shared use. The form layout editor selects the editor for each property. For instance, association editors may be replaced with embedded list views and forms, enabling re-use of existing layouts. In the list layout editor, hierarchical relationships can be used to define threaded list views. Furthermore, association chains can be followed when defining the list of properties to be displayed, allowing us to set up de-normalized views. Layouts can be the default layout for a particular object type within a given application, in which case they will be selected automatically in overview lists and forms for the corresponding object type.

Since layouts contain some complex information that needs not be modeled explicitly (such as property information and editor type), the major part of their definitions are stored in some encoded form. Only the elements to query and manage layouts are modeled explicitly, such as name, default flag, object type, user (if private) and application (if any). This suffices to configure a simple application for listing, reporting and easily sharing layouts between users and applications. The actual layout definitions are handled by means of the layout editors.

Extend or refine authorization rules

Access control is based on a set of authorization rules. These are not tied to any specific model and can be used to set up both very fine- and course-grained access. Authorization rules, just as action rules, can be context- and contents-sensitive. Some rules apply to everyone; other rules apply to groups of people whose members are determined at run-time. This way, authorization rules support cohesion, while giving considerable flexibility in supporting different team cultures.

Authorization rules contain four parts. The agent (= whom we are granting or denying access) and object (= what objects the rule is about) conditions are defined using the query editor component.

The aspect part specifies which aspects (e.g. properties) the rule applies to. The rights part denotes the privileges (e.g. read and export).

Define action rules

These rules capture business-semantics, set defaults, perform extra validation, impose constraints and add functionality. Action rules can cover an entire organization, particular functions or can be limited to a specific application environment. Hence action rules give extensive support to autonomous teams while preserving overall consistency. Action rules come in two flavors: high-level template rules for defining workflow processes and script rules for handling more specific cases.

Define workflow process templates

These specify default scenarios that the user typically follows in the context of a business process. Users can (usually) deviate from these scenarios and snap back into the pre-defined flow later on. Processes can be configured incrementally, e.g. between departments first, and within each department later on. Depending on team culture, these individual subprocesses are more or less strictly defined. In addition, we add automated tasks and dedicated private or shared "in- / outbaskets" to manage incoming and outgoing work.

5. Framework components

In more technical papers [Dev][Til]-- full text available http://www.argo.be/aiv/docs -- we present an overview of the framework components. We broadly classify the framework components in the following groups:

- the repository component: the object model, system objects and end-user objects
- front-end components: end-user and configuration tools, such as forms, document viewers and the object model editor
- back-end components: the persistency component, document-storage, background process managers, the E-mail gateway and the OCR-engine
- central components: the central core component, the application manager, the document session manager, the authorization rule-base, and the background process manager board, the action rule manager and managers for various other system objects.

6. Framework evolution

We started the framework initially building three applications. At this moment we built and evolve many more. We tuned, adapted and extented our framework and business model, based on what we learned from the needs and shortcomings of these applications and from the feedback from both configurators and end-users.

Since the business object, object model, script and applications specifications are all stored in the same repository we can use it as a sort of central knowledge base. Some working procedures and processes can be recognized as recurring patterns and are formally modeled using status and relations in the repository. Other informal business practices are less clearly recurring at the start and often come to surface, appearing in multiple custom scripts. When we identify these patterns, we try to make them more explicit either in the repository or in the tools. In several places, we needed to list objects in a hierarchical way. Thus we extended the overview lists with an option to select hierarchical display.

When developing applications with the framework we identify:

- **Working practices specific to the organization**. We create re-usable assets in the repository, re-factor the object model, or we develop more specific end-user and configuration applications using the existing tools.
- **Practices which transcend applications**. We change the framework functionality, and re-factor the framework [Foo95, Rob96].
- **Functionality specific to a particular application**. Depending on its nature, we change the functionality of the repository, the framework or both. For instance, some applications require a simple procedure to automatically generate standard reply letters based on repository information. In this case, we store extra behavior in the repository. If a new external tool is required, we usually add a component to the framework.
- **Technological needs and opportunities**. We use subframeworks for components that are strongly coupled to technology, such as the persistency layer and the document storage. These subframeworks can easily be replaced.
- **Additional functionality**, to support new types of applications, e.g. Internet applications.

This way, the framework explicitly supports evolution and re-use at different levels [Til96].

7. Building softer software

A context

Due to external and political factors, Argo was to undergo some rather drastic reorganizations, as well as major changes regarding their main objectives, in the course of the project. The actual outcome was largely unknown at the onset of the project. As such, the system to be implemented had to provide ample scope for modeling different kinds of organization and workflow process structures. Additionally, in order to allow for more flexible adaptation whenever the models needed to be re-designed, the concrete implementations had to forego hard-coded assumptions of a particular model as much as possible.

This is not unique to Argo. In many organizations the need to change has become a constant. Our software needs to support change, not obstruct it.

The need for softer software

"The system requires it" [Tal95] or "The system does not allow it", have become accepted (and often unavoidable) justifications of human behavior.

What we fear is that current methods do not allow us to build soft enough software. Present methods and design paradigms seem to inhibit adaptability. We become experts at what we can specify in advance, working with the unstated belief that there exists an optimal solution.

Once technology is adopted by an organization, it often becomes a constraining structure that in part shapes the action space of the user. When software is designed in accordance with the formal reality of organizations, the outcome is generally a system that must be subverted to restore flexibility to end users or that leads to an organization that hobbles along with a dysfunctional system. [Sal94]

Working in face-to-face projects where the end-user is known, where the user has a face that is happy or upset, we are remembered every day that our task is to build software that delights the user, that there is more to software than firmness and function.

We build software too much like we build hardware, as if it was difficult to change, as if it has to be difficult to change. We have to build "softer" software. It is no use trying to write full requirements up front. The user does not know what is possible and will ask for the pre-tech-paper solutions that he perveives to be possible. If even we do not know fully what we are building before it is actually built, our user has not a clue of what is possible before he can feel it, touch it[Blu96]. So we need another approach to building software. It is impossible to have full requirements up front or to freeze the context and environment. Requirements are written in a context. Our system transforms that context. New problems arise in the new system and the new context.

This issue is not solved through improved methods for identifying "the" user requirements; instead they call for a more complex process of generating fundamentally new operating alternatives.

It is very unsurprising that many information systems developments are delivered into an environment that has changed drastically from the one in which they were conceived. We often perceive a problem as uniquely technical. If we analyze things in enough detail before construction we will prevent costly errors later on in the trajectory. With wicked, ill-described problems, this does not work. Users cannot, with any amount of effort and wisdom accurately describe the operational requirements for a substantial software system in advance, without testing in an operational environment, and iteration on the specification. The systems built today are too complex for the mind to foresee all the ramifications purely as an exercise of the analytical imaginations [Cro96]

Dissemination of practices and applications

Soon after the start of the project, we found out that technical problems were not the major risk. The real risk is that substantial amounts of time and money are spend on trying to solve technicalities which are not the real problem. Our real problem is our user's lack of understanding of new possibilities and fear of the unknown. People hide their fear of change and use rational arguments to oppose it, rather than discussing their feelings and emotions.

Our major challenge is not to understand and analyse but to get the analysis accepted by people when it deeply offends their view of the world. Often people will not say "I am loosing prestige", they will say that the system is not efficient[Dhe98].

People will listen with much greater attention to their own people whom they trust. Users are our best marketing people.

Learning from chaos and self-organization

We started to work with metaphors of chaos [Pri84], autopoiesis [Mat72] and self-organization.

After 3 years of both strong opposition from some managers and broad support, we strongly agree with Beer [Bee90] that large scaled and expensive changes like cultural-, quality- or human resource programs have little, sometimes-aversive effect. Strategic management cycles focus too much on the possibility to build and to control change, and paradoxically big one-time- decisions lead to new rigidity; e.g. the outcome of many BPR-projects. Total organizational change can be successful when it starts in "small, isolated, peripheral operations". A new structure must first establish itself in a limited region and then invade the whole space.

In a large bureaucracy like Argo new ways of working, new practices, tend to be stabilized away in competition with the issuing of instructions and rules and the "this is how we do it" mentality. In smaller units creativity can bloom, and weak signals get a chance to be enforced spontaneously and to lead to renewal[Bro92]. Moreover, when a failure, they don’t disturb the whole organization.

We work intensively with small kernels of users and we are convinced that we reached a level of change acceptance that would not have been possible with a top-down introduction of the new systems.

Our framework allows us to have multiple kernels working in parallel, each starting close to their own proven way of working. Users are now active in propagation and dissemination of their little systems.

They train their colleagues, invite them to think about better practices to work together on new ways of using and tuning the system to their needs.

Divergence from formal descriptions

We need to be conscious of the fact that people fear that IT will change their jobs, will influence power relations in the organization. When interviewing people we need to be conscious that there may be a difference between espoused theory and theory in use [Arg92]. We will get the official story, the one that keeps the boss happy.

We know this, but still when we build software we tend to minimize or even ignore this and pretend that if we only we follow a method rigorously, we can stay in charge, in charge of our team and of our users. There is an important risk of writing shelfware if we neglect this.

There is little space for the informal realities of the organization to coexist with the formal structures of the organization if the software system is implemented as designed.

Traditional user requirements analysis tends to be dominated by those with most power who will generally define the objective requirements as those that maintain the status quo and specifically their own position. Managers often view themselves as the key beneficiaries of software and high emphasis may be set on reporting and on needs that they can readily understand. However the primary objective of many software applications is to facilitate the delivery of a service rather than to report on what has been accomplished.

Organizations often have formally described policies and procedures but operate according to an informal structure that provides more autonomy to workers and flexibility in procedures.

Software may lead to a tighter coupling of the organization, narrowing the gap between the formal rules and procedures of an organization and the actual reality of what people do to get the job done, effectively and flexibly. This tighter coupling can lead to significant leaps in quality and efficiency or it can lead to a crisis in organizations where efforts at improving work organization are just rhetoric [Sal94].

This can have profound consequences for the work lives of those using technology. We experienced during an analysis workshop that the one department that served customers in time was not working by the rules. Strict implementation of the official procedures would have made work-around impossible and would have had a very negative impact on effectiveness and efficiency.

The divergence between the formal description and the reality of the workplace operations is important and should not be neglected. Formal procedures, policies and rules are not likely to represent the way the organization actually functions or how tasks are actually done.

Ineffective design is not just a technical problem of inadequate methods for specifying or assessing user requirements. It resides just as well in the unstated belief that there is an optimal solution to user needs. Consensus among users within the organization is assumed, the locus of systems failure limited to developer-user interaction. [Ros]

User involvement is certainly an important step toward designing user-centered software, but thinking that the problem can just be handed to the user is an insufficient action.

Users often have limited technical knowledge about computer systems and users lack the broader view of the organization that allows them to evaluate the overall requirements of service or production.

User defined designs tend to be built around existing job requirements for the work "as is", rather than around work processes that are re-conceptualized to make use of advances in software design and hardware technology. Often we computerize practices even if they were not efficient and we replicate functions and features from older technologies, thus building the limitations of the older systems into the new ones.

Opportunity scouting – working together to find better ways to do things

We started the overall project with a series of vision workshops with management, stakeholders, users and their clients. Together with the users we created an image of the future in which systems will be used. Now, rather than using individual interviews that strengthen differences between departments, at the start of each new sub-project we organize a kick-off and a series of working groups with users, configurators, developers and a facilitator. In these workshops we try, using metaphors and creative techniques, to reach

beyond description of the “desired” system in terms of the current view on working procedures and a solution-after-next concept. [Bra94] [Cou96] [Das96] [DeB95] [Gra96] [Mor86]

Questioning and then changing governing beliefs help us supply the possibility of fundamental change. We use things like Technology fiction, Free association, Imaginizing (what if a certain measure would be completely achieved), Worst-case, Heroes, Future workshops, Story telling, Drawing and Random stimulus. Recently we started using the Solution-After-Next principle [Nad94] in our workshops. The solution-after-next principle [Nad94] encourages users to envision the ideal solution – one that might not be implementable now, but one toward which we can strive.

In Future Scenario writing (some atypical) managers and users ask themselves the question what they would do and need if such and such happen, thus building memories of the future [Bra94] [deG97]. This turned out to be very important since the organization evolved from a highly centralized system to high local autonomy. This led to the requirements for access to the central system through the Internet and the need to configure applications over the Internet.

Incremental software development -- a build-and-evolve paradigm

As soft system thinkers [Che98]we are still concerned with problem identification and solution, but we place increasing emphasis upon the process of problem appreciation. This process of appreciation can best be facilitated by providing a framework for organizational inquiry and learning [Sto94].

This is related to Alexander’s piecemeal growth and removal of misfits [Ale64]. The essential feature of an Alexanderian systems-fit approach is design as an ongoing process, having procedural rationality and involving active participation. [Ale75][Ale79]

People like to believe that systems -- products -- that match their specification as-is are a perfect support for their job.

But nearly all successful systems, including business, go through a process of learning better rules. An organization needs the freedom to experiment and to learn.

Architecture supported incremental development

The framework approach gives us the means needed to work as we described.

We adapted and complemented known and proven methods of Object Oriented Analysis and Design with lateral thinking techniques.

We placed more emphasis on approaches that try to capture working processes, and increase user involvement, e.g. through scenarios, use cases and role playing (CRC). We are convinced that CRC, scenario's and use case are very useful in change programs. In system envisioning workshops [Dev92] we help our users to find out what the impact of technology can be through future perfect thinking [Hoh97] and opportunity scouting.

Our approach involves active participation of users in development, not only in analysis and testing of new releases, but in the actual writing of end-user documentation and in the training of their peers. This is related to the nature of our framework and the way our end-user applications are build and tuned.

In our evolutionary, incremental approach, we start with a basic design that is easy to modify and adapt, port or change. Our users start using the system, and learn from it. Our framework allows rapid cycles where the user can experiment and actually work with his environment.

We specify the system's desired behavior through a collection of scenario's, we create and validate an architecture that exploits common patterns found in these scenarios and evolve the architecture, making mid-course corrections if necessary to adapt to new requirements as they are uncovered. [Boo96]

Corrections and evolution of the framework are based on feedback from the many applications we build and from interaction with both configurators and users. Since we rebuild our tools with the framework we add an additional test for the architecture.

For the end-user applications we go beyond correction and traditional evolution. We encourage the user to think about his ways of working and to change them to validate new opportunities and chances. We actively promote change in the applications. In our first version we stick as close as we can to what the user feels comfortable with, often his old pre-technology way of doing things with the flavor of new

possibilities. Often this is different for employees from different departments. Once acquainted with the possibilities and perceived constraints of the new system, the users from the different departments work together on new processes, shared applications, layouts, queries and reports, bringing together multiple contexts and scenario's. Our configurators and developers get feedback for future versions and incorporate common practices and patterns in the framework or in workflow templates.

As an industry we have invested a lot of effort into firmness, begrudgingly acknowledging commodity and puzzle at delight. [Cop98] With our architecture we provide structure to guarantee firmness and leave ample space for commodity and enough variability to delight the individual user.

We are convinced that this variability is critical to the success of a project.

8. The team

At the start of the project we gathered a team of skilled software developers, experienced in Smalltalk and (one) in C++ programming. Most members on the team came from research environments or small companies working on EDM and workflow projects. Few of them had any experience in structured approaches to software development and software engineering.

We did not adopt a particular method at the onset, but experimented with known techniques, adapted them to our needs and documented them as we gained confidence.

We found inspiration in the Framework and Application Prototyping teams as described by Goldberg and Rubin [Gol95]. We have a special Framework team, the core-team, which is highly interacting with, what we call the Configuration team that uses the framework to build end-user applications. These teams work together (and recently we decided to have two members working on both teams part of their time) in working out scenarios, both real and imaginary (some atypical for use and potential use of the framework).

Our highly skilled self-selecting team learned a lot from Coplien's Generative Development Process Pattern Language [Cop95]. Programming in pairs, we change partners -- and location -- often, thus enlarging skills and overall knowledge of the framework. We have been working along these lines for more than a year and at the end of the former big project- (and budget-) phase and start of the new; we organized a two days workshop with the team, evaluating does and don’ts with the pattern language. During that workshop additional patterns were selected from related languages. Recently we started using Alistair Cockburn’s Project Risk Reduction Patterns [Coc98]

We have a dedicated tester and document writer. They are working very intensively with end-users for alfa-and-beta-testing and writing of online-help and end-user-documentation. Documentation writing and the testing process start at the kick-off meeting of each project, both tester and writer being part of the initial requirements-development-workshop. At that meeting they start developing test-scenario's with the configurators and start writing tutorial examples.

We use creative workshops among the development team, as well as with users. Because we tend to produce and work within a paradigm often restricting our solutions to those we know how to build. We do know that we embody our technology with our habits of thought and we want to raise these habits to full consciousness and take responsibility for them.

Trust is a core value in the team, absolutely essential to the success of the project.
Trust. If we don't believe in what we are doing, we will not succeed. It took someone to believe he could fly to actually take us in the air. We need trust in our method, trust among the team, trust from the sponsor and from the user.

In a trusting environment it is allowed to learn, it is allowed to "throw one away". Throwing one away is a means to learn and it can save us a lot of stress and workarounds later on.

A welcoming environment and early success are important. Because early success reduces the energy needed in the future. Once you have convinced your environment that it is possible to fly, it is much easier to get funding to build our airplane.

Our team uses maximum bandwidth communication for information sharing and through daily Scrum meetings [http://www.controlchaos.com/index.html] -- each person getting one minute to report on activities, plans and changes or support needed. We expect the unpredictable, so we allow flexible

responses, assuming that our product is never complete and that working solutions must be completed, the framework and applications evolving with the user.

We use a managed, time-boxed, iterative, incremental cycle. This constant measuring and timely control ensure us that the product is the best that we can deliver, given the technology at hand in our environment and the timeframe. Our culture is centered on results. We build high visibility in our process, mobilizing pioneer user teams for every major release, handing the new versions of our framework and applications to them for testing and actual use.

We try to make the project a constant challenge for the group and the individual developers. There is space to experiment and to even "to throw one away".

A minor observation: we use the paper we produce and we produce the paper (and only the paper) we intend to use [Fow97]. We know from experience that few design documents are read after they have been written and before they are out of date. We generate UML schema's and guarantee documentation-as-built.

The team at large: user involvement

It is important that we are conscious that we embody our thoughts with our past and with our expectations of what is possible. And that the user, the customer does the same. We, technicians, need to show the customer the possibilities and the limits if needed. Analysis is mostly concerned with the "Ends", the goals, while developers are concerned about "Means", possibilities. Analysis and development have to meet and interact, not only at the start of the project, but during the whole trajectory. IT changes the environment, but the environment changes IT too. We as developers have to be aware of it. There is not "the" context, to be learned about by good analysis; there is a context we change continually.

On our team, a very special role is given to end-users. We use qualitative measures rather than quantitative. We want users to present the system to new colleagues as the thing to use, user delight expressed through users training their peers, organizing opportunity scouting workshops, writing online help and manuals. This process was initially driven by small teams of highly motivated pioneer-users working very close to developers, and has gradually started to embrace the end-user community at large.

9. Conclusion

At Argo we made a resolute choice for an architecture-driven approach, focussing on creating a framework that satisfies known "hard" requirements, and able to adapt -- resilient enough to adapt -- to those requirements that are not yet known or understood.

We build softer software, supported by softer methods. Soft system thinking supporting a repository based framework.

We use a managed, time-boxed, iterative, incremental cycle.

We apply as much process as we really need and stimulate the use of creativity and lateral thinking to broaden our users' scope of expectations.

Flexibility and adaptability are considered more important than a full set of analysis and design drawings. Models are made on a white board or with post-its on brown paper during workshops. Users do not know what to expect, they formulate requirements in terms of past experience, so we need to create an environment and provide them tools to envision and experience the "deliverables", allow them to give feedback, incorporate the feedback in short cycles.

The main goal of the framework design was a small kernel of generic components acting dynamically upon a central knowledge base. Hard coding was to be avoided as much as possible. Initially, we focused on achieving this goal for end-user applications. We developed hard-wired administration and configuration tools to help us bootstrap the system. As the framework evolved, and the tools became more flexible, we re-used components of end-user tools in some of our administration tools. In a third phase, we started to replace some of the hard-wired tools with applications configured in the system.

The framework configuration functionality will be further enhanced, in order to further increase the expressiveness of the tools, and to allow end-users to adapt the tools to their own needs even better, thus obviating the need for developers to a larger degree. Developers concentrate on the hard part, building

only what the user cannot build. This approach allows us to build for re-use, to build the framework with the initial budget and the initial number of people.

Thus some unplanned additional components are being developed, most notably to make the repository Internet-aware. The main component will act both as a client of the repository and as a generic Internet application server. This will empower Argo to develop Internet applications through modeling and configuration too.

Although we initially encountered some performance problems, these were not essentially related to the approach as such, but to the use of a persistency component, which was not really suited for our purposes. We must be aware, however, that the flexibility of the system allows end-users to build e.g. equally good and bad queries. Proper training is of primary importance, as is the need to initially hide some of the more complex elements of the tools from novice users.

After four years, we are convinced that the approach is a valid one. End-user applications can be developed iterative, interactive and incrementally. We do not code (apart from some custom scripting) and we do not generate end-user applications. Neither do we use or need throwaway prototypes. Instead, we build increasingly complete specifications of end-user applications. These specifications are available for immediate execution. In this way, we help close the gap between specification, development and use of the applications. This way we can afford to stimulate the users to think and rethink their processes and redesign them.

We tried to effectively use the objects: a means to attack complexity, building a system resilient to change able to launch incomplete systems that allow feedback and intensive user-involvement throughout the lifecycle, from defining business processes to designing and developing solutions, testing and regularly refining them, to make sure that the applications truly address critical business needs and users help implement "their" application. [Gol95][Tay97]

10. References

[Ale64] Christopher Alexander; Notes on the synthesis of form, Harvard University Press 1964

[Ale75] Christopher Alexander, M. Silverstein, S. Angel, S. Ishikawa, D. Abrams; The Oregon experiment; Oxford University Press, New York 1975

[Ale79] Christopher Alexander; The timeless way of Building, Oxford University press 1979

[Arg92] Chris Argyris; On organizational learning, Scriptum 1992

[Bee80] Stafford Beer; Preface to Autopoiesis and cognition [Mat72]

[Blu96] Bruce I. Blum; Beyond programming – to a new era of design, Oxford university press, New York 1996

[Boo96] Grady Booch; Object Solutions, Managing the object oriented project; Adison Wesley, CA 1996

[Bra94] Stewart Brand; How buildings learn; Viking, London 1994

[Bro92] Gerrit Broekstra; Chaossytemen als metafoor voor zelfvernieuwing van organisaties; van Dijkum en de Tombe ed.; Aramith Uitgevers 1992

[Che98] Peter Checkland and Sue Holwell, Information, Systems and Information systems; John Wiley & Sons 1998

[Cob98] Alistair Cockburn; Surviving Object Oriented Projects; AddisonWesley, 1998

[Cop95] James O. Coplien, A generative development process pattern language, PLOP ed. Coplien-Schmidt, Addison Wesley 1995

[Cop98] James O. Coplien; Software development as Science, art and Engineering in the Patterns Handbook, Linda Rising ed.; Sigs Books, New York 1998

[Cro96] Malcolm Crowe, Richard Beeby, John Gammack; Constructing systems and information; Mc Graw Hill 1996

[Cou96] J.Daniel Couger; Creativity and innovation in Information Systems Organizations; Boyd & Fraser Publishing Corporation, Colorado 1196

[Das96] Subrate Dasgupta; Technology and creativity; Oxford university press, New York 1996

[deB92] Edward de Bono; Serious Creativity; HarperCollins, London 1992

[deG97] Arie de Geus; The Living Company; Nicholas Braley London, 1997

[Dev94] Een visie voor Argo, proces rapport

[Dev96] Martine Devos and Michel Tilman, Design and Implementation of a Business Modeling Framework using Smalltalk, Object Technology'96, 1996

[Dhe98] Olivier D'Herbemont & Bruno César; La stratégie du projet latéral; Dunod 1996

[Foo95] Brian Foote and William F. Opdyke, Lifecycle and Refactoring Patterns that Support Evolution and Reuse, Pattern Languages of Program Design, Addison-Wesley, 1995

[Foo96] Brian Foote and Joseph Yoder, Evolution, Architecture, and Metamorphosis, Pattern Languages of Program Design, Addison-Wesley, 1996

[Fow97] Martin Fowler; UML distilled, applying the standard object modeling language; Addison Wesley 1997

[Flo93] Fernando Flores, Michael Graves, Brad Hartfield and Terry Winograd, Computer Systems and the Design of Organizational Interaction, Readings in Groupware and Computer-Supported Cooperative Work, Morgan Kaufmann, 1993

[Gab96] Richard 1. Gabriel; Abstraction Descant; The bead game, rugs and beauty; Patterns of Software; Oxford university press 1996

[Gam95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns, Elements of Reusable Object-Oriented Software, Addison Wesley, 1995

[Gol95] Goldberg A. & Rubin K.S., Succeeding with objects, Addison Wesley 1995

[Gra96] D. Grant and C. Oswick ed.; Metaphor and organizations; Sage Publications 1996

[Häg89] S. Hägglund, Iterative Design and Adaptive Maintenance of Knowledge-Based Office Systems, Proceedings of the IFIP WG 8.4 Working Conference on Office Information Systems: The Design Process, 1988, North-Holland

[Hoh97] Luke Hohman, Journey of the Software Professional, Prentice Hall, 1997

[Kar90] B. Karbe, N. Ramsperger and P. Weiss, Support for Cooperative Work by Electronic Circulation Folders, Proceedings of the ACM OIS'90 Conference, 1990

[Mat72] H. R.Maturana en F. J.Varela: Autopoiesis and cognition; D.Reidel Publishing Company, Dordrecht 1972

[Mor86] Garett Morgan, Images of organization; Sage publications 1986

[Nad94] G. Nadler and S. Hibino; Breakthrough thinking – the seven principles of creative problem solving; Prima publishing 1994

[Pri84] I. Prigogine and Stengers, Order out of chaos, Bantam New York, 1984

[Rob96] Don Roberts and Ralph Johnson, Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks http://st-www.cs.uiuc.edu /users/droberts/evolve.html

[Sal94] Harold Salzman & Stephen R.Rosenthal; Software by design; Oxford university press, New York, 1994

[Sto94] Frank Stowell and Duane West, Client-Led Design, A systemic approach to Information System Definition, McGraw-Hill, 1994

[Tal95] Stephen Talbott; The future does not compute, O'Reilly and associates, 1995

[Tay97] David A. Taylor; Object technology, a manager's guide (2nd edition); AddisonWesley 1997

[Xer96] Xerox Parc Aspect-Oriented Programming Project, A position paper on Aspect-Oriented Programming, position paper for the ACM Workshop on Strategic Directions in Computing Research, Working Group Object-Oriented Programming, MIT, June 14-15 1996.