Requirements-Driven Iterative Project Planning

Yves Wautelet[1], Manuel Kolp[2], and Stephan Poelmans[1]

[1]Hogeschool-Universiteit Brussel, Brussels, Belgium
[2]Université Catholique de Louvain, Louvain, Belgium
{yves.wautelet,stephan.poelmans}@hubrussel.be,
manuel.kolp@uclouvain.be

Abstract. Organizational modeling with the i* framework has widely been used for model-driven software development adopting a transformational approach, notably within the Tropos process. Its high-level representation elements allow to partition the software problem into adequate and manageable elements (*actors*, *goals*, *tasks*, *resources* and *dependencies*) leading to an agent-oriented design, and eventually an implementation with agent technologies (*JACK*, *Jadex*, *Chimera Agent*, ...). This paper proposes to use the i* framework for iterative software planning; each of the goals from the i* strategic dependency model are evaluated on the basis of the (high-level) *threats* they face and the expected *quality factors*. This allows to determine a priority among the model goals and "feed" an iterative template to plan the whole project realization. This framework is thus meant to be applied during the first iteration of the project for model-driven software project management. The development of a production management system in the steel industry is used as an example.

1 Introduction

Due to benefits and advantages such as efficient software project management, continuous organizational modeling and requirements acquisition, early implementation, continuous testing and modularity, iterative development is more and more adopted by software engineering professionals especially through methodologies such as the *Unified Process* inspired ones (*RUP, OpenUP, EUP, AUP, ...* [8–10, 12]). The idea governing the iterative approach is to decompose a software development process into a series of manageable entities avoiding to face as a whole, every aspect of the project. Moreover, variable and non-measurable effort is spent on each of the engineering disciplines during every iteration for fast prototyping and early testing. Obviously, the most risky issues i.e., the most difficult to be expressed by the users, understood by the analysts or those addressing the most sensitive issues (such as security, flexibility, adaptability,...) receive highest priority. Consequently, these "critical" questions are dealt with first so that the development team receives feedback from users and from the whole system environment early on. This improves the probability of adequately meeting user requirements and getting the right adoption of the system into its environment.

Most software methodologies based on the agent paradigm use a pure waterfall software development life cycle (SDLC) or advise their users to repeat stages "iteratively" during the project in an *ad-hoc* way. One main reason resides in the fact that no

M.J. Escalona, J. Cordeiro, and B. Shishkov (Eds.): ICSOFT 2011, CCIS 303, pp. 121–135, 2013.

theoretical framework to support this way of proceeding has ever been defined and published. We believe that iterative development requires a formal or semi-formal managerial framework to support dynamic requirements and risk-driven development planning so that we require an adequate way of breaking the software problem into independent manageable entities and then to prioritize them to plan their development and evaluate their achievement.

Tropos [4] is an agent-oriented requirement-driven methodology that uses the i* (i-star) modeling framework [17, 18] during the analysis stage; i* defines advanced organizational modeling features and semantics in the form of agents, goals, tasks and resources. This allows to partition a software problem on the basis of the agent paradigm which is the main reason why we have chosen to extend Tropos with an iterative template and use the i* goals as fundamental entities to decompose the problem into several aspects. Let us note that the research is actually generic enough to be adopted to extend other agent-oriented software methodologies. Iterative planning is here based on the i* strategic dependency diagram's goals. Moreover, when planning a project with an iterative SDLC, one needs a generic process template. For this matter, we define, in this paper, an "UP-compliant" reference framework in line with existing theory on iterative SDLCs. The contributions of this paper include this iterative template and a planning method illustrated with a running example based on the development of a production management system in the steel industry.

The paper is structured as follows: Section 2 overviews a proposal for an iterative template for Tropos developments. Section 3 presents a *model-driven architecture* method for iterative planning in the context of I-Tropos developments. This method deals with threats and quality factors as fundamental criterias for goal prioritization. Section 4 points to related work and finally Section 5 gives the reader a conclusion.

2 Iterative Template

The first proposal of this paper is to adopt the traditional Tropos models and stages to define a project management template used to drive the software process. We also propose a common engineering terminology.

An "I-Tropos development" is an extension of the Tropos methodology, made of disciplines iteratively repeated while the relative effort spent on each one is variable from one iteration to another. The phase and discipline notions are often presented as synonyms in the software engineering literature. Indeed, Tropos is described in [4] as composed of five phases (*Early Requirements*, *Late Requirements*, *Architectural Design*, *Detailed Design* and *Implementation*). However, the Software Process Engineering Metamodel [1] defines a discipline as *a particular specialization of Package that partitions the Activities within a process according to a common "theme"*, while a phase is defined as *a specialization of WorkDefinition such that its precondition defines the phase entry criteria and its goal (often called a "milestone") defines the phase exit criteria*. The Unified Process [12] defines disciplines as *a collection of activities that are all related to a major "area of concern"* while the phases *here are not the traditional sequence of requirements analysis, design, coding, integration, and test. They are completely orthogonal to the traditional phases. Each phase is concluded by a major*

milestone. In order to be compliant with the most generic terminology, traditional Tropos phases will be called disciplines in our software process description since "they partition activities under a common theme". In the same way, phases will be considered as groups of iterations which are workflows with a minor milestone.

In I-Tropos, the Organizational Modeling and Requirements Engineering disciplines respectively correspond to Tropos' Early and Late Requirements disciplines. The Architectural and Detailed Design disciplines correspond to the same stages of the traditional Tropos process. I-Tropos includes core disciplines, i.e., *Organizational Modeling*, *Requirements Engineering*, *Architectural Design*, *Detailed Design*, *Implementation*, *Test* and *Deployment* but also support disciplines to handle the project development called *Risk Management*, *Time Management*, *Quality Management* and *Software Process Management*. There is little need for support activities in a process using a waterfall SDLC since the core disciplines are sequentially achieved once for all. However, for an iterative process, the need for support disciplines to manage the whole software project is from primary importance to precisely understand which project aspect to work on (and through which activity) at a specific time and with the best resources. I-Tropos process' disciplines are described extensively in [15].

Using an iterative SDLC implies repeating process' disciplines many times during the software project. Each iteration belongs to one of the phases usually four to six depending on the process itself, four in our case. We relate to the phases defined in the UP-based methodologies (RUP, OpenUP, EUP, ...) but are redefined here to better match with the Tropos specificities. These phases are achieved sequentially and have different goals assessed at milestones through knowledge and achievement oriented metrics. Phases are informally described into the next section. Figure 1 offers a two dimensional view of the I-Tropos process depicting the disciplines on Y-axis and the four different phases they belong to on X-axis.

2.1 Core Disciplines

The I-Tropos process has been fully described using the Software Engineering Process Metamodel in [15] that details each process' Discipline, Activity, Role, WorkDefinition and WorkProduct, so that it can be used as a reference, template, pattern or guide for managing the system project. A lightened overview is given below. As already pointed out, the first four disciplines are inspired by the Tropos original stage.

- the *Organizational Modeling* discipline aims to understand the problem by studying the existing organizational setting;
- the *Requirements Engineering* discipline extends models created previously by including the system to-be, modeled as one or more actors;
- the *Architectural Design* discipline aims to build the system's architecture specification, by organizing the dependencies between the various sub-actors identified so far, in order to meet functional and non-functional requirements of the system;
- the *Detailed Design* discipline aims at defining the behavior of each architectural component in further detail;
- the *Implementation* discipline aims to produce an executable release of the application on the basis of the detailed design specification;

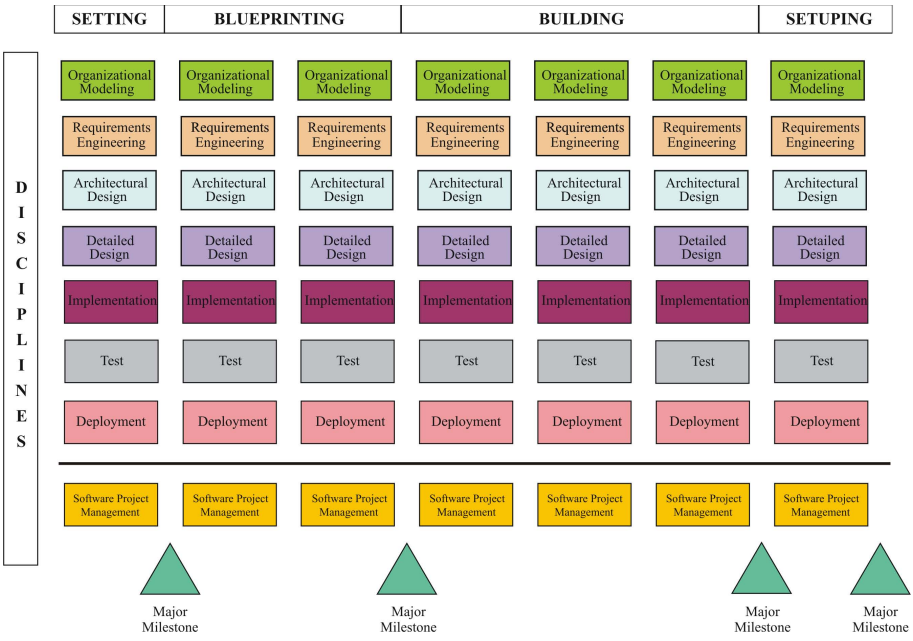

Fig. 1. I-Tropos: Iterative Perspective

- the *Test* discipline aims on evaluating the quality of the executable release;
- the *Deployment* discipline aims to test the software in its final operational environment.

2.2 Support Disciplines

These support disciplines provide features to support the software development on a particular project i.e., tools to manage threats, quality factors, time, effort, resources allocation but also the software process itself. All those features can be regrouped onto the term software project management [11].

- *Risk Management* is the process of identifying, analyzing, assessing risk as well as developing strategies to manage it. Strategies include transferring risk to another party, avoiding risk, reducing its negative effects or accepting some or all of the consequences of a particular one. Technical answers are available to manage risky issues. Choosing the right mean to deal with particular risk is a matter of compromise between level of security and cost. This trade-off requires an accurate identification of the threats as well as their adequate evaluation;
- *Quality Management* is the process of ensuring that quality expected and contracted with clients is achieved throughout the project. Strategies include defining quality issues and the minimum quality level for those issues. Technical answers are available to reach quality benchmarks. Choosing the right mean to deal with quality issues is a matter of compromise between level of quality and cost. This trade-off

requires an accurate identification of the quality benchmarks as well as their adequate evaluation;
- *Time Management* is the process of monitoring and controlling the resources (time, human and material) spent on the activities and tasks of a project. This discipline is of primary importance since, on the basis of the risk and quality analyses, the global iterations time and human resources allocation are computed; they are revised during each iteration;
- *Software Process Management* is the use of process engineering concepts, techniques, and practices to explicitly monitor, control, and improve the systems engineering process. The objective of systems' engineering process management is to enable an organization to produce system/segment products according to plan while simultaneously improving its ability to produce better products. In this context, Software Process Management regroups the activities aimed to tailor the generic process onto a specific project as well as improving the software process.

2.3 Process Phases

I-Tropos phases are inspired by UP-based processes and their milestones are based on the metrics from [3]; each phase is made of one or more iterations. Disciplines are conducted through the phases sequentially. Each phase has its own goal:

- the *Setting* phase is designed to identify and specify most stakeholders requirements, have a first approach of the environment scope, identify and evaluate project's threats and identify and evaluate quality factors;
- the *Blueprinting* phase is designed to produce a consistent architecture for the system on the basis of the identified requirements, eliminate most risky features in priority and evaluate blueprints/prototypes to stakeholders;
- the *Building* phase is designed to build a working application and validate developments;
- the *Setuping* phase is designed to finalize production, train users and document the system.

3 Iterative Planning

This section describes a method for planning Iterative Tropos developments. The relevant disciplines are illustrated on a running example, the development of an enterprise information system in the steel industry.

3.1 Running Example: Coking Process

CARSID, a steel production company located in the Walloon region , is developing a production management software system for a coking plant. The aim is provide users, engineers and workers with tools for information management, process automation, resource and production planning, decision making, etc. Coking is the process of heating coal into ovens to transform it into coke and remove volatile matter from it. Metallurgical Coke is used as a fuel and reducing agent in the production of iron, steel,

ferro-alloys, elemental phosphorus, calcium carbide and numerous other production processes. It is also used to produce carbon electrodes and to agglomerate sinter and iron ore pellets. The production of coke is one of the steps of steel making but further details about other phases of the production process are not necessary to understand the case study.

3.2 Agents for Steel Making

First of all, one question must be answered: *How can an industrial domain such as the steel industry that seems to belong to the past be interested in agent technologies?* In other words *why agent-oriented modeling (and development) would be more indicated than traditional - possibly object - technologies?*

The steel industry is by essence an agent-oriented world. Indeed, factories as a coking plant or a blast furnace are made of hundreds of different types of agents: *software agents*, *machines*, *automates*, *humans*, *sensors*, *releases*, *effectors*, *controllers*, *pyrometers*, *mobile devices*, *conveying belts*, etc. These are agents in the sense that:

- they are autonomous and dedicated to specific tasks;
- they are situated in a physical environment;
- they can act upon their environment if this is necessary by warning users, proposing solutions or taking autonomous action.

The whole I-Tropos project profile is illustrated in Figure 2. Rectangles represent the relative effort spent on each of the disciplines during each of the phases. Typically, the reader can notice that the effort spent on analysis disciplines (organizational modeling and requirements engineering) is higher into the setting and blueprinting phases and marginal for the building and setuping ones. On the contrary, the design (architectural design and detail design) and implementation ones are marginal for the setting phase (at the early beginning of the project) and higher in the blueprinting and building phases. The project management disciplines (i.e., the support disciplines) are addressed on a continuous basis with a stronger focus early on in the project (during the setting phase). We mostly concentrates in this paper on disciplines and activities performed during the setting phase since the focus is to describe a method for iterative planning. More precisely concerning the engineering disciplines, we will only focus here on organizational modeling and requirements engineering since they are the ones required for model-driven planning. The support disciplines will be covered in detail to depict the process of goal prioritization and development planning. However, the support discipline *Software Process Management* is not covered here since it goes into too many low-level details than we can afford in this research paper.

3.3 Engineering Disciplines

The i* framework can be evaluated on a series of nine features following [14]: *refinement*, *modularity*, *repeatability*, *complexity management*, *expressiveness*, *traceability*, *reusability*, *scalability* and *domain applicability*. Those features are exhaustively assessed on the basis of a *not supported/not well supported/well supported* scale. Notably

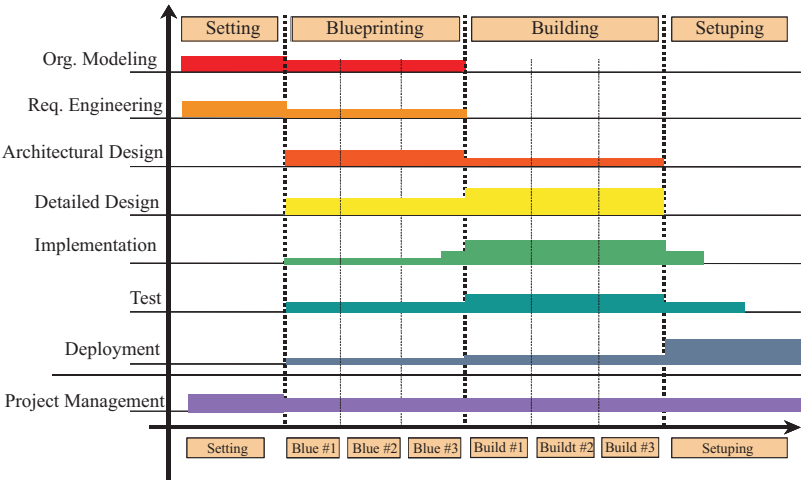

Fig. 2. I-Tropos profile for the Carsid Coking Plant project

they enlighten what is clearly needed to extend the i* framework with mechanisms to manage granularity and refinement. Indeed, [14] points out the lacks of mechanisms in i* for defining *granules* of information at different abstraction levels to structure, hierarchise or aggregate the semantics represented in the model. One of the flaws of i* is actually that all of the organizational modeling elements are represented on a unique abstraction level with poor hierarchy and composition/aggregation. Moreover, except for specifying abstract primitives as building blocks, analysts must be provided with guidelines to model a complete business setting through a set of organizational processes. These building entities could then be enriched into a set of more specific components that capture a certain organizational behavior.

These discussions are from primary concern in the perspective of finding scope elements, i.e., primary abstractions to drive the whole development process including support disciplines rather than just software engineering ones. Instead of defining a new model reaching those criteria, as proposed in [6, 14, 16], we rather prefer to provide guidelines to the software analysts in order to specify an i* strategic dependency model where each of the goals can be taken as input into the I-Tropos process. Those include:

- a goal must be defined at the highest abstraction level such as the organizational and strategic levels. Typically a scope element should describe a conceptual process, e.g., one business process so that a goal must encapsulate a high level service provided by the enterprise;
- lower level processes must be represented as tasks and a task must always be a refinement of a goal;
- goals are expressed independently, overlaps must be avoided and, if not possible, eventual redundancy among tasks of two different goals are addressed at a lower level.

By respecting those rules, all the goals of the i* strategic dependency diagram (SDD) are scope elements for breaking down the software project into manageable parts and

are taken as input in the support disciplines as shown in the next sections. Figure 3 (also reproduced at the end of the document) represents the SDD model built up following those rules on the CARSID case study. Due to the lack of space we cannot detail each of the diagram elements here, nevertheless the reader should pay attention to the facts that:

- *Human actors* are represented as circles, for example the *FADS Team* is the team in charge of handling the coal reception;
- *Equipment actors* are also represented as circles, for example the *Coke Car* is the automotive wagon that transports the red-hot coke to the *Quenching Tower*;
- *Resources* are represented as rectangles, for example *Coal* is the raw coal received at the coking plant;
- *Goals* represented as rounded rectangles, for example *Baking* is the process for which the *Oven Team* supervises the baking of the Coal Charge in the Oven;
- *Softgoals* represented as clouds, for example *Quality Coke* represent the willingness of Management to insure that the Coke produced in the coking plant is good as the steel quality depends on the Coke quality;
- *Tasks* are represented as hexagonal forms, for example *Bake* is a sub-process of the *Baking* goal only concerned with the physical transformation of the *Coal Charge* into *Coke*.

3.4 Risk Management

The Risk Management discipline uses the goals identified into the organizational modeling and requirements analysis disciplines as fundamental scope elements. Consequently the identified threats are evaluated on the basis of their impact on these elements. We define a threat as *an event that can negatively affect the proper resolution of a goal or that can be the result of the misuse of a goal execution both in terms of goal achievement and degradation of quality*. A threat is expressed as an aggregate risk with a quantification of the negative impact and a occurrence probability. A threat is later refined into a series of softgoals with respect to the transformation process.

Risk analysis was done in collaboration with stakeholders estimating and validating the possible threats impact. Risk quantification is done on a double basis. Firstly the general threat weight is estimated on the basis of the impact it can have on the project under development, the system-to-be or the concerned organization. Secondly, the involvement of each goal with respect to the risk evoked is evaluated following a *Low/Medium/High* scale. This has led to the identification of six categories of threats:

- *Requirements poorly understood*: the system does not run as expected. This threat is particularly faced by user-intensive software applications. This kind of risk has a weight of 3 since huge resources can be devoted to produce an inadequate system;
- *Facility damage*: it concerns the damages caused to production facilities. A representative example is when the pusher machine pushes while hot coke is stuck in the oven, resulting in damaging the pusher machine and/or the oven's walls. This kind of risk has a weight of 2 since facilities repairs are cost-intensive;

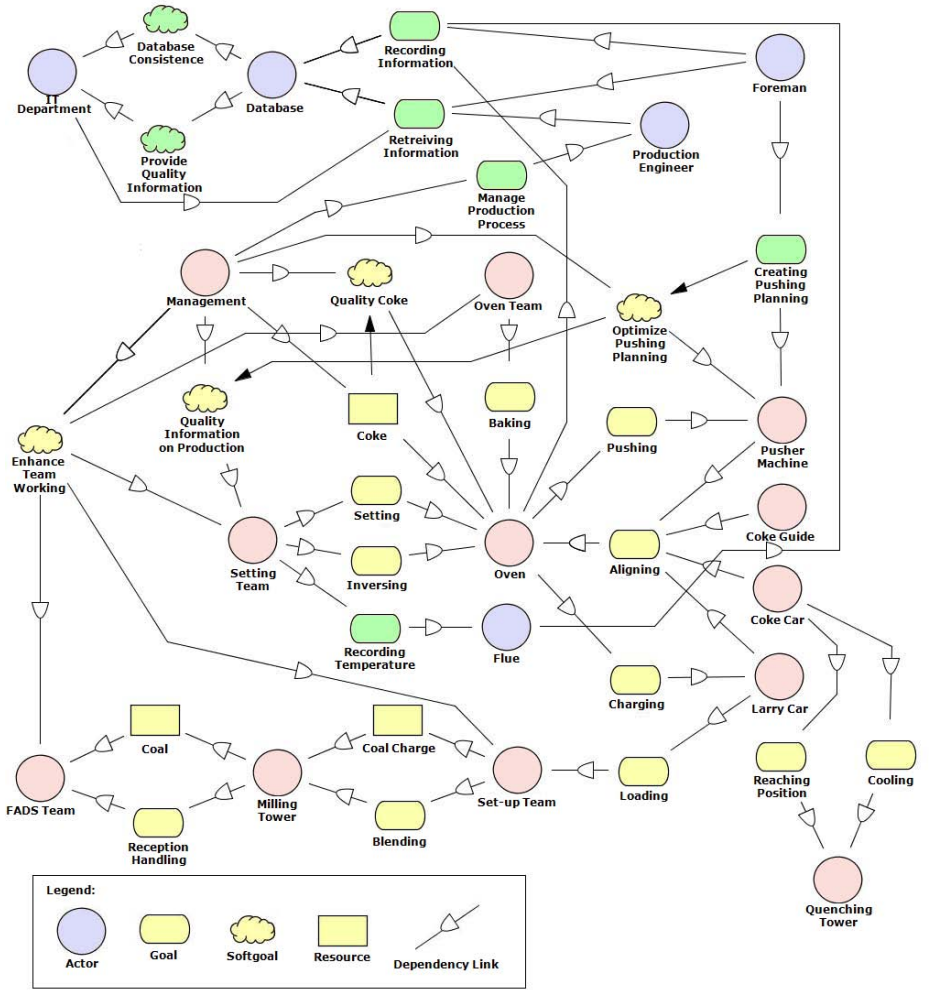

Fig. 3. Organizational Modeling

- *Mechanical error*: it concerns errors due to machinery dysfunction. For instance, a failure in the coke car engine, making impossible to cool down the coke. This risk has a weight of 1 since it will delay production;
- *Human injuries*: it concerns injury (or even death) of the staff and/or workers. A coking plant is a hazardous place; a replacement worker was recently killed in a coke plant in Ohio, and other numerous accident of this type took place in the past. This risk has a weight of 3 since it is very costly in terms of money and reputation;
- *Human error*: it concerns errors due to human intervention. This risk has a weight of 2 since it can potentially lead to other risks.
- *System failure*: the information system is not available. The system may be down and cannot be used for a certain amount of time. This risk has a weight of 1 because it will delay production.

	Threat weight		Aligning	Baking	Blending	Charging	Cooling	Creating Pushing Planning	Inversing	Loading	Manage Production Process	Pushing	Reaching Position	Reception Handling	Recording Information	Recording Temperature	Retrieving Information	Setting
Requirements poorly understood	3							L			H						H	
Facility damage	2			L		L		L		M		H		L				
Mechanical error	1		L	L	L	M	M	L	M			M	M	L				
Human injury	3					L			L	M		L	L	L				
Human error	2							M						M	H			H
System failure	1										M				L	L	L	
Data loss	2										M				M	M		
Goal Risk Exposure			1	3	1	7	2	10	5	10	18	13	5	10	13	5	10	8

Fig. 4. Project Goals Risk Exposure

– *Data loss*: some needed data is lost or never existed. It may happen if one of the worker forgets to encode some data. This risk has a weight of 2 because it will delay production and can potentially lead to other risks.

On the basis of the organizational model of Figure 3 and the risk analysis, the matrix in Figure 4 summarizes the impact of the identified threats onto the modeled goals. A specific goal is facing a particular threat is the marked and a measure is given. For example the goal *Aligning* faces the threat *Mechanical Error* even if the probability of occurence of the threat on the goal remains *Low*. The overall risk exposure of each goal is finally computed on the basis of the threats they faced, the intensity and the threat's weight.

3.5 Quality Management

The Quality Management discipline uses the goals identified into the organizational modeling and requirements analysis disciplines as fundamental scope elements. Consequently the identified *Quality Factors* are envisaged on the basis of their impact on them. Quality factors must firstly be distinguished from traditional softgoals in the sense defined in [5]. Since we address a higher level business view of the system to-be, we need an abstraction where we can specify the quality concerns of the system rather than its states as softgoals would characterize. In this sense, softgoals describe functions of a system while quality factors do not. We define a threat as *a constraint onto one or more goals in the form of a degree of excellence*. A quality factor is later refined into a serie of softgoals in the transformation process.

Quality analysis was done in collaboration with stakeholders estimating and validating the possible quality factors impact. That work has led to the identification of six categories of quality factors:

- *Reliability*: Software Reliability is the probability of failure-prone software operation for a specified period of time in a specified environment. This quality factor deals with the question: *What level of trust can be placed in what the system does?*
- *Efficiency*: Software Efficiency deals with execution time, the later can be influenced by source code optimization, the use of performing algorithmic techniques, faster sequential execution or code parallelization. This quality factor deals with the question: *How well are resources utilized?*
- *Usability*: Software Usability is the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions. This quality factor deals with the question: *How easy is it to use?*
- *Integrity*: Software Integrity is the ability of software to resist, tolerate and recover from events that threaten its dependability. This quality factor deals with the question: *How secure is it?*
- *Testability*: Software testability is a software characteristic that refers to the ease with which some formal or informal testing criteria can be satisfied. This quality factor deals with the question: *How easy is it to verify conformance to requirements?*
- *Flexibility*: Software flexibility is the ability of a software system to adapt to change. This quality factor deals with the question: *How easy is it to modify?*
- *Interoperability*: Software interoperability is defined as the ability for multiple software components written in different programming languages and distributed across multiple platforms to communicate and interact with one another. This quality factor deals with the question: *How easy is it to interface with another system?*

Based on the organizational model in Figure 3 and the quality analysis, the matrix in Figure 5 summarizes the impact of the identified quality factors onto the modeled goals. A goal facing a particular quality factor is marked and a measure is given, for example the goal *Aligning* faces the quality factor *Efficiency* even if the concern remains *Low*. The overall quality factor exposure of each goal is finally computed on the basis of the quality factors they faced, the intensity and the quality factor's weight.

3.6 Time Management

The previous two sections studied the threats and quality factors impact on the goals from the SDD. This section uses the global risk and quality exposures to determine a goal priority. Indeed, within the defined iterative life cycle we will plan the goals' realization. Goals with highest priority will firstly be designed, prototyped and tested during the *Blueprinting* phase and then implemented during the *Building* phase. This allows scrutinizing the trickiest issues first so that risks are addressed early on in the project when corrective actions are easier and cheaper to put into practice.

The relevant concepts and their computations are summarized in Figure 6. Particularly, we emphasize that:

- The *Overall Risk Exposure* is the sum of all threats involvement at any level on one goal;
- The *Total Risk Exposure* is the sum of the *Overall Risk Exposure* of all the project goals;

	Quality Factor Weight		Aligning	Baking	Blending	Charging	Cooling	Creating Pushing Planning	Inversing	Loading	Manage Production Process	Pushing	Reaching Position	Reception Handling	Recording Information	Recording Temperature	Retrieving Information	Setting
Reliability	3		M			L		M			M	H	M		H	M	H	L
Efficiency	2		L			L		H			H	M	L		L	L	M	M
Usability	1							M			M						M	
Integrity	3							L			M				H	M	M	
Testability	1			L	L		L	M	L	L	M		M	L			M	L
Flexibility	2		M			M	L	H		L	M	M	M					M
Interoperability	2		M	L		L	L	L		L	M		L		M	M		L
Goal Quality Exposure			16	3	1	11	5	31	1	5	32	20	16	1	30	18	26	14

Fig. 5. Project Goals Quality Issues

Goal	Overall Risk Exposure	Relative Risk Exposure	Overall Quality Exposure	Relative Quality Exposure	Priority Level	Goal Priority	Goal Complexity	Goal Complexity Weight	Precedence
1 Aligning	1	0,008264463	16	0,069565217	0,023589651	13	L	5	
2 Baking	3	0,024793388	3	0,013043478	0,021855911	14	L	5	
3 Blending	1	0,008264463	1	0,004347826	0,007285304	16	M	10	
4 Charging	7	0,05785124	11	0,047826087	0,055344951	9	M	10	
5 Cooling	2	0,016528926	5	0,02173913	0,017831477	15	M	10	
6 Creating Pushing Planing	10	0,082644628	31	0,134782609	0,095679123	4	H	15	G4, G10
7 Inversing	5	0,041322314	1	0,004347826	0,032078692	12	L	5	
8 Loading	10	0,082644628	5	0,02173913	0,067418254	6	M	10	
9 Manage Production Process	18	0,148760331	32	0,139130435	0,146352857	1	H	15	
10 Pushing	13	0,107438017	20	0,086956522	0,102317643	3	H	15	
11 Reaching Position	5	0,041322314	16	0,069565217	0,04838304	11	M	10	
12 Reception Handling	10	0,082644628	1	0,004347826	0,063070428	8	L	5	
13 Recording Information	13	0,107438017	30	0,130434783	0,113187208	2	L	5	
14 Recording Temperature	5	0,041322314	18	0,07826087	0,050556953	10	L	5	
15 Retrieving Information	10	0,082644628	26	0,113043478	0,090244341	5	M	10	G13
16 Setting	8	0,066115702	14	0,060869565	0,064804168	7	M	10	
Sum	121	1	230	1	1			145	

Fig. 6. Goal Prioritization

- On the basis of the *Overall Risk Exposure*, the *Relative Risk Exposure* is computed using the formula: $RelativeRiskExposure = \frac{OverallRiskExposure}{TotalRiskExposure}$;
- The *Overall Quality Exposure* is the sum of all the levels of involvement of all the quality factors on one goal;
- The *Total Quality Exposure* is the sum of the *Overall Quality Exposure* of all the project goals;
- Based on *Overall Quality Exposure*, the *Relative Quality Exposure* is computed with the formula: $RelativeQualityExposure = \frac{OverallQualityExposure}{TotalQualityExposure}$;

	Goal	Priority	Complexity	Elaboration 1	Elaboration 2	Elaboration 3	Construction 1	Construction 2	Construction 3
1	Manage Production Process	1	H	X			X		
2	Recording Information	2	L	X			X		
3	Pushing	3	H	X			X		
4	Charging	9	M	X			X		
5	Creating Pushing Planing	4	H		X			X	
6	Retrieving Information	5	M		X			X	
7	Loading	6	M		X			X	
8	Setting	7	M		X			X	
9	Reception Handling	8	L		X			X	
10	Recording Température	10	L			X			X
11	Reaching Position	11	M			X			X
12	Inversing	12	L			X			X
13	Aligning	13	L			X			X
14	Baking	14	L			X			X
15	Cooling	15	M			X			X
16	Blending	16	M			X			X
Total Effort Weight			145	15	16,66667	16,66667	30	33,33333	33,33333

Fig. 7. Iteration Plan

- The *Priority Level* is computed by "balancing" the *Relative Risk Exposure* and the *Relative Quality Exposure*. For this particular project we chose a repartition key of 75 percent for the risk component and 25 for the quality component. This repartition key can vary from one project to another and should be calibrated from data collected on a large number of projects as well as considering the working team experience;
- On the basis of the *Priority Level*, each *Goals Priority* is deduced.

On the basis of the *Goal Priority* list, the *Precedence* constraints and the estimated *Goal Complexity* we instanciate the iteration template defined in Section 2. The process is summarized in Figure 7. The Goal complexity estimates the amount of effort required to develop it. I-Tropos uses three categories of goal complexity, i.e., *Low/Medium/High*, owning respectively a weight of *5/10/15*. Transforming the overall weight to man-month requires calibration typically based on a regression model using statistics from large numbers of projects and remains an open issue. The proposed planning will be subject to modifications/reviews during the software project supported by users and environmental feedbacks.

4 Related Work

Numerous agent-oriented software development methodologies have been proposed in the past twenty years. We overview hereafter some of them with a particular focus on iterative SDLC and software project management. Tropos [4] offers the most advanced agent-based modeling features through the i* models, one of the reasons we use it as a basis for the process presented in this paper. Some papers related to this methodology propose to develop a software system in an iteratively way but no supporting theoretical framework has ever been proposed. Gaia [19], one of the most popular methodologies

due to its simple and clear process as weel as its neutrality with respect to implementation techniques or platforms has been extended with an iterative SDLC in [7]. The main drawback of their proposal is that they decompose functionality on the basis of design models (called "parts") and not analysis ones. Moreover, the priority given to those functional parts is done in an *ad hoc* manner since no framework is given to evaluate the criticality of these "functional parts". There is also no real template provided for cutting out the project into phases so that each iteration can be considered as a "sub-waterfall project" with no rapid prototyping for testing as implied by the cut out between blueprinting and building phases in our framework. ADELFE [2] claims to follow the RUP but no guideline or planning method is actually given for that purpose. The process is depicted in [2] in a waterfall manner. Finally, MASSIVE [13] uses an *Iterative View Engineering* approach itself based on *Iterative Enhancement* and *Round-trip Engineering*. From the authors' point of view, the method can be said to be incremental rather than iterative. As a matter of fact, it is founded on producing hybrid models to be tested and enhanced later on into the project on the basis of users' feedback. However, the software project is not broken down into manageable elements that are prioritized and worked on during multiple iterations so that no software project management framework is required to manage the SDLC.

5 Conclusions

Since the emergence of spiral development, software engineering professionals have understood the benefits of iterative approaches. However, poor guidelines and too simple project management frameworks have failed to provide practionners with clear methods to deal with such SDLCs in a structured manner. Moreover, *Model-Driven Architecture* (*MDA*) and *Model-Driven Development* (*MDD*) are extensively used in the software transformation process, i.e., the process of tracing high level analysis elements into design and implementation ones. This principle is applied here but analysis models are used to feed the process at a managerial level for *goal-driven project management*. Contributions thus include a structured (model-driven) method for iterative lifecycle management.

The process needs however to be refined and better calibrated on the basis of experience gained from multiple projects. It has recently been applied on the development of a collaborative supply chain management platform from which an in-depth analysis of the empirical results will be presented soon. A CASE-Tool called *DesCARTES Architect* supporting all of the methodology's models and I-Tropos life cycle management has also been developed to better operationalize the method. The present research is clearly driven by the willingness to bring agent-based development to large enterprise software developments in order to propose an integrated process with clear guidelines supported by practical tools. Finally and as just mentioned, the method was used here in the context of agent-oriented development but the transformation process from i* models can lead to an object implementation. Similarly, the goal planning method can be very easily and flexibly adapted to (semantically poorer) UML use-cases and serve RUP practionners in their everyday iterative development planning.

References

1. Anonymous. Software & systems process engineering meta-model specification. version 2.0. Technical report, Object Management Group (2008)
2. Bernon, C., Gleizes, M.P., Picard, G., Glize, P.: The adelfe methodology for an intranet system design. In: AOIS@CAiSE (2002)
3. Boehm, B.: Software Project Management. Addison-Wesley (1998)
4. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the tropos project. Inf. Syst. 27(6), 365–389 (2002)
5. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-functional requirements in software engineering. Kluwer Academic Publishing (2000)
6. Estrada, H., Rebollar, A.M., Pastor, O., Mylopoulos, J.: An Empirical Evaluation of the *i** Framework in a Model-Based Software Generation Environment. In: Martinez, F.H., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 513–527. Springer, Heidelberg (2006)
7. Gonzalez-Palacios, J., Luck, M.: Extending Gaia with Agent Design and Iterative Development. In: Luck, M., Padgham, L. (eds.) AOSE 2007. LNCS, vol. 4951, pp. 16–30. Springer, Heidelberg (2008)
8. IBM. The rational unified process. Rational Software Corporation, Version 2003.06.00.65 (2003)
9. Jacobson, I., Booch, G., Rumbaugh, J.: The unified software development process. Addision-Wesley (1999)
10. Jacobson, I., Bylund, S.: The road to the unified software development process. Cambridge University Press (2000)
11. Jalote, P.: Software Project Management in Practice. Addison Wesley (2002)
12. Kruchten, P.: The Rational Unified Process: An Introduction, 3rd edn. Addison-Wesley (2003)
13. Lind, J.: The MASSIVE Method. LNCS (LNAI), vol. 1994. Springer, Heidelberg (2001)
14. Pastor, O., Estrada, H., Martínez, A.: The strengths and weaknesses of the i* framework: an experimental evaluation. In: Giorgini, P., Maiden, N., Mylopoulos, J., Yu, E. (eds.) Social Modeling for Requirements Engineering. MIT Press (2011)
15. Wautelet, Y.: A goal-driven project management framework for multi-agent software development. PhD thesis, Université catholique de Louvain, Belgium (2008)
16. Wautelet, Y., Achbany, Y., Kolp, M.: A service-oriented framework for mas modeling. In: Cordeiro, J., Filipe, J. (eds.) ICEIS (3-1), pp. 120–128 (2008)
17. Yu, E.: Modeling strategic relationships for process reengineering. PhD thesis, University of Toronto, Department of Computer Science, Canada (1995)
18. Yu, E.: Social Modeling for Requirements Engineering. MIT Press (2011)
19. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The gaia methodology. ACM Trans. Softw. Eng. Methodol. 12(3), 317–370 (2003)