GOAL DRIVEN ITERATIVE SOFTWARE PROJECT MANAGEMENT

Yves Wautelet\textsuperscript{1} and Manuel Kolp\textsuperscript{2}

\textsuperscript{1}Faculté Economie en Management, Hogeschool-Universiteit Brussel, Brussels, Belgium
\textsuperscript{2}Louvain School of Management, Dept of Operation and Information Systems, Université catholique de Louvain
Louvain-la-Neuve, Belgium

yves.wautelet@hubrussel.be, manuel.kolp@uclouvain.be

Keywords: Iterative development, Agent-oriented software engineering, i* modeling, Software project management.

Abstract: Iterative development has gained popularity in the software industry notably in the development of enterprise applications where requirements and needs are difficult to express for the users and business processes difficult to understand by analysts. Such a software development life cycle is nevertheless often used in an \textit{ad-hoc} manner. Even when templates such as the Unified Process are furnished, poor documentation is provided on how to breakdown the project into manageable units and to plan their development. This paper defines a template for agent-oriented iterative development as well as a software project management framework to plan the project iterations. The agent paradigm is used at analysis level with actors’ goals considered as piecing elements, an iterative template is proposed for planning purpose. High-level risk and quality issues focus on prioritizing the project goals so that each element’s “criticality” can be evaluated and a model-driven schedule of the overall software project can be set up. The software process is illustrated with the development of a production planning system for a steel industry.

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 (Kruchten, 2003). 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 each of the iterations 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 an adequate 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 \textit{ad-hoc} way. One main reason resides in the fact that no 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 (Castro et al., 2002) is an agent-oriented requirement-driven methodology that uses the i* (i-star) modeling framework (Yu, 1995; Yu, 2011) 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, the main reason 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 MDA (model-driven architecture) method for iterative planning in the context of I-Tropos developments. This method deal 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 (Castro et al., 2002) as composed of five phases (Early Requirements, Late Requirements, Architectural Design, Detailed Design and Implementation). However, the Software Process Engineering Metamodel (OMG, 2005) 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 (Kruchten, 2003) 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 (Wautelet, 2008).

Figure 1: I-Tropos: Iterative Perspective.

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 mile-
stones 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 (Wautelet, 2008) 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 of the 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;
- 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 (Jalote, 2002).

- 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 compromise 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 compromise 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 ((Boehm, 1998)); 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.

Figure 2: I-Tropos profile for the Carsid Coking Plant project.
3.3 Engineering Disciplines

The i* framework can be evaluated on a series of nine features following (Pastor et al., 2011): 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 they enlighten what is clearly needed to extend the i* framework with mechanisms to manage granularity and refinement. Indeed, (Pastor et al., 2011) 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 (Estrada et al., 2006; Pastor et al., 2011; Wautelet et al., 2008), 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 autonomous 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.

Figure 3: Organizational Modeling.
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;
- **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;
- **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.
- **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 occurrence 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 (Chung et al., 2000). 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 series 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;
- On the basis of the Overall Risk Exposure, the Relative Risk Exposure is computed using the formula: RelativeRiskExposure = \( \frac{\text{Overall Risk Exposure}}{\text{Total Risk Exposure}} \);
- 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;
- On the basis of the Overall Quality Exposure, the Relative Quality Exposure is computed us-
GOAL DRIVEN ITERATIVE SOFTWARE PROJECT MANAGEMENT

Figure 5: Project Goals Quality Issues.

Figure 6: Goal Prioritization.

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 instantiate 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 (Castro et al., 2002) 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 (Zambonelli et al., 2003), one of the most popular methodologies due to its simple and clear process as well as its neutrality with respect to implementation techniques or platforms has been extended with an iterative SDLC in (Gonzalez-Palacios and Luck, 2007). 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 (Bernon et al., 2002) claims to follow the RUP but no guideline or planning method is actually given for that purpose. The process is depicted in (Bernon et al., 2002) in a waterfall manner. Finally, MASSIVE (Lind, 2001) 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 learned the benefits of iterative approaches. However, poor guidelines and project management frameworks have provided practitioners with clear methods to deal with such SDLCs in a structured manner. Moreover, MDA (Model-Driven Architecture) and MDD (Model-Driven Development) is 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 the managerial level for goal-driven project management.

Finally, the process needs to gain experience, it has recently been applied (and has given encouraging results) on the development of a collaborative supply chain management platform with an in-depth analysis of the empirical results. The method can also be very easily and flexibly adapted to UML use-case driven development and serve RUP practitioners in their everyday iterative object-oriented development planning.
REFERENCES


