Goal Sketching: Towards Agile Requirements Engineering

Kenneth Boness
University of Reading,
Berks, RG6 6AY UK
k.d.boness@reading.ac.uk

Rachel Harrison
Stratton Edge Consulting Ltd,
GL7 2LS, UK
rachel.harrison@strattonedge.com

Abstract

This paper describes a technique that can be used as part of a simple and practical agile method for requirements engineering. The technique can be used together with Agile Programming to develop software in internet time. We illustrate the technique and introduce lazy refinement, responsibility composition and context sketching. Goal sketching has been used in a number of real-world development projects, one of which is described here.

1 Introduction

Our goal sketching technique starts with the creation of a goal graph which expresses the high level motivations behind the intention to develop the software. This is typically an incomplete sketch of what is understood about the overall intention. In general there is often a vague long-term vision coupled with some short-term clarity. A series of staged developments are planned using the system graph as a guide. This is similar to using Scrum sprints [1], or increments in an iterative and incremental development process [2]. Each stage is preceded by taking a portion of the system graph in its current state and refining it so that there are no remaining vague intentions. This is called the 'stage graph'. In the execution of any stage it is possible that the stage graph will be updated as a result of the usual agile practice of improving the quality of the work in hand. At the completion of each such stage its graph is used to update the system graph. Thus the true goal graph emerges by successive iterations and refactoring and so becomes the inventory, recording the associated rationale for posterity.

When preparing each stage the goals are refined only as far as necessary for the stage in hand (a technique called *lazy refinement)* using stories or more rigorous techniques (such as problem frames [3], temporal logic etc). We use *pair sketching*, in which the goal graphs are sketched by two people working together (often an analyst and a stakeholder) to ensure that the refinement argument is sound, in a manner akin to pair programming. Once an acceptable goal graph has been produced, according to the measures collected by technique 2, it is incorporated into the system goal graph. The system graph may need to be refactored for the next stage.

The goal graphs are exported to a database for subsequent analysis. From the database we can produce matrices to expose composition issues which may arise from cross-cutting concerns for analysts, designers, developers and testers. We formalise the responsibility re-composition of cross-cutting requirements as follows:

{(responsibility, CR, sub-system} =
{(responsibility, CR)} X {(CR, sub-system)}

where X represents a natural join and CR is a cross-cutting requirement. This formalism can be used to determine which responsibilities interact with which functional requirements in which sub-system.

4 Example

To illustrate the technique we will use an example involving the calculation of body mass index.

The customer, WeighCom, wishes to develop new walk-on scales that can be installed in public places and used by any passers-by to measure their weight, height and body mass index (BMI) and receive a business card sized printed record on the spot. Normal operation is for the user to step onto a pressure mat facing an instruction screen and stand under an acoustic ranger. The measurements are made once the user pays a fee of 1 Euro into a receptor.

WeighCom specifies that the solution must use certain components: pressure mat (PM); coin receptor (CR); acoustic ranger (AR) and integrated processor with alpha numerical visual display and user selection touch screen (IP). All of these are to be controlled through software using an API. These components support an existing assembly in which the whole is weather proof and vandal proof.

WeighCom currently installs personal weighing equipment in public places for coin operated use by the public. They have an excellent reputation, which is of paramount importance to them, for always providing a reliable service or repaying. They have a call centre which customers can call if their installations appear to be malfunctioning.

Figure 1. Problem Statement

Scrutiny of the problem statement suggests the following primary concerns:-

- Operation in public places.
- Normal operation (i.e. accepting payment through to printing a card)
- Use of prescribed components.
- WeighCom's reputation.

These provide the necessary information for an initial system goal sketch as shown in Fig. 3.

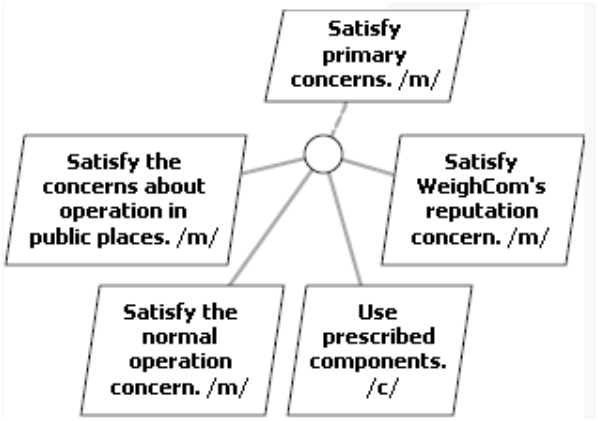

Figure 2. Primary Concerns

The goals in Fig.3 represent four child goals entailing one parent as is usual in KAOS but using semantic entailment instead of logical entailment. As an aid to goal sketching each goal is given a type from the selection shown in Table 1. It is usual to have motivation goals ("/m/") as root and sub root goals. These may be refined into further motivation goals and eventually into constraint and behavior goals. The motivation goals essentially are the drivers of the intention whereas the behavior goals represent the capability and condition requirements of system requirements according to the IEEE guidelines [4]. Similarly the constraint goals correspond to the constraint requirements of the guidelines. There is an important distinction between the motivation and behavior goals. The former can express temporal intentions in a project timeline whereas the behavior goals represent temporal intentions in terms of the system run-time. This means that the goal graph must progress in one direction from top to bottom; at the top are motivations and below them are behavior and constraint goals. In a well formed goal sketch only motivation goals appear as root goals and only behavior or constraint goals appear as the leaves.

Table 1. Tags for different types of goals

Tag	Goal type	Comment
m	motivation	A business case driver of objectives
c	constraint	A constraint in well-formed requirements.
b	behavior in the environment	One or more capabilities and conditions.
a	assumption	An unrefinable asserted goal needed to justify a refinement argument.

Each of the primary concern goals is refined as far as current knowledge will allow depending on the state of the development (see technique 1). Early on the understanding of the 'public places' concern may be vague (or be simply deferred as complex but unnecessary for early demonstrations of the product) and so is left as a 'to be determined' (TBD) goal; see Fig. 4. Meanwhile attention may be placed on completing the refinement of the normal operation concern. This is what we predicted in technique 1. Clearly there is the possibility that when subsequently the TBD is replaced with a full refinement there may be a need to refactor and rework the system goal graph and the supporting implementation of software. This is no different to normal agile practice except that we have the goal sketch as a constant indicator of the assumptions.

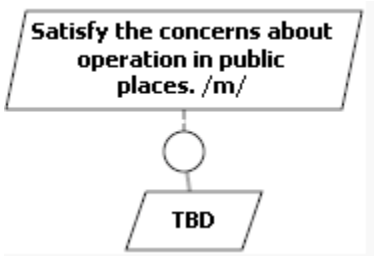

Figure 4. Deliberately leaving a refinement to later stages of development (lazy refinement)

Pursuing the normal operation goal (shown in Fig.3): refinement is continued until goals are reached that are concrete enough and deemed suitable to be allocated to one or more agents to take run-time responsibility. This is a departure from KAOS where responsibility is given to a single agent with full control over the real-world parameters. Here a management choice is made that the goal can be satisfied by the behavior of one or more agents in cooperation. This arises typically when an eXtreme type story is reached; for example:-

> **Transaction Initiation Story**: "When a customer pays €1 into the CR they may either confirm the payment or cancel the payment. If they cancel then the CR refunds the payment. If they confirm then the service is initiated."

Three agents must cooperate in the satisfaction of this goal: the user, a coin receptor (CR) and the software-to-be (S2B). The rigorous application of KAOS would require more refinement to the point of separately specified responsibilities. Alternatively the stakeholders may decide that this is sufficiently concrete and risk free for them to accept the current degree of refinement as adequate; left in this state it is called a *lazy* refinement.

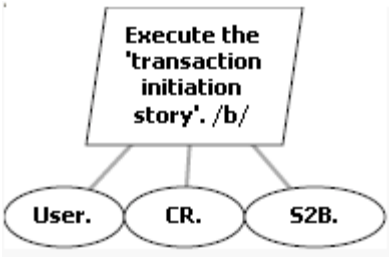

Figure 5a. Lazy Refinement

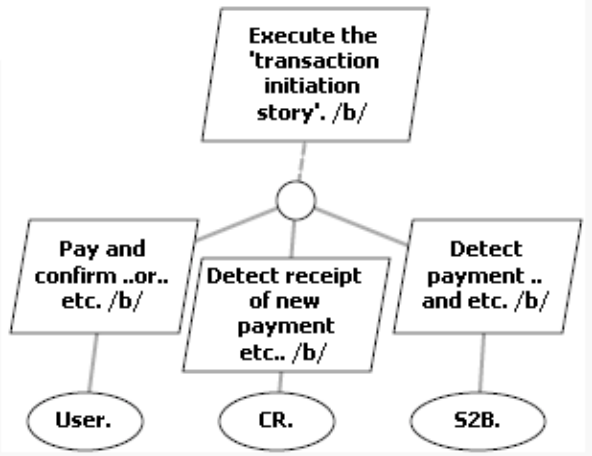

Figure 5b. Full Refinement

The two routes are shown in Fig.5 (a) and (b). As case (a) is a lazy refinement depending on multiple agents to collaborate in its satisfaction it is not a pure specification /s/ or expectation /x/ and therefore will be a behavior (/b/) goal. As case (b) is fully refined through additional sub goals each of the leaf goals is assigned to a single agent and the leaves are expectations and specifications; those shown in the figure are only suggestions.

Goal sketching favors lazy refinement wherever possible. In contrast KAOS would progress to case (b). Goal sketching also favors the use of natural language whereas KAOS employs a formal logic calculus; thus in goal sketching goal refinement arguments are semantically entailed rather than logically entailed as they are in KAOS.

In the process of creating clear refinement arguments goal sketching favors a strict policy of separation of concerns. This implies decomposition and thus necessitates a late re-composition [3] as cross-cutting concerns (e.g. collaboration between responsible agents to indicate necessary collaboration between capabilities and the imposition of constraints and conditions [3]). In our experience this approach minimizes the number of goals with multiple parents and thus reduces visual tangling in the goal graph. The price for this benefit is that the composition concerns are not explicit. However a lightweight solution is to annotate the assigned responsibilities using a system of *composition tags* (described below).

In contrast KAOS uses object and operation models to accommodate composition concerns. This can be rigorous but tends to be heavyweight.

Each oval is a responsibility.
The **agent** name is shown before the full stop.
The presence of **composition** concerns is indicated by including semantic tags as text inside chevrons (e.g. <MYTAG>). If the semantic tag includes the symbol "@" this means that the associated goal must be composed with all other responsibilities bearing the same <MYTAG>. There may be multiple tags representing the accumulations of multiple compositions

Figure 6. Responsibility annotation

Fig.6 shows three versions of the responsibility assignments. Each is shown as an oval with the name of an assigned agent followed by a full stop. The architectural precision of the agent depends upon the underlying domain analysis being used; e.g. an object in a UML model or a sub-domain of a Jackson context diagram [3]. An optional system of semantic tagging is allowed after the full stop. Each tag is written in the form "<MYTAG>" or <@MYTAG>. Any responsibility with a given tag (say <MYTAG>) is a target for composition with a similar named tag including the "@". Thus a responsibility marked <@MYTAG> composes with all responsibilities tagged with <MYTAG>. Composition means that the goal associated with the '@' symbol is added to or changes the goal associated with the other responsibilities. This feature allows strict separation of cross-cutting concerns and subsequent re-composition. The semantic tags are created and managed by the analyst either manually or with tool support.

The rigour of the goal sketch can be greatly enhanced if it is drawn with a companion *context sketch* based on the method of Jackson's context diagrams [3]. If this is done the context diagram provides the vocabulary (entities and phenomena) that may be included in the goal sketch. In practice we find the goal sketch and the context sketch co-evolve. When a context diagram is available the responsibility annotations can be enriched with the sub-domain identifier being taken from the diagram and the composition tags reflecting shared phenomena [3] defined in the context diagram. This method of co-evolving a context sketch with the goal sketch and using composition notation has been used in the following industrial projects with the aim of assisting managers, designers and testers.

5 Industrial Projects

Our method is on trial in a number of industrial applications. These include (1) a product supported by venture capital, (2) the specification, procurement and acceptance of a management information system (MIS) to support a food processing company and (3) a tool to support professional services in healthcare. We start with some general observations and then look at the MIS application in more detail.

The staffing profiles for the projects involve managers, executives, developers and testers. These people all have very different perspectives and analytical abilities. In all cases the managers and executives were not involved with detailed requirements analysis, whereas the developers and testers were. In the venture capital sponsored project the developers (circa 15 engineers) were inclined to use agile methods and the testers (circa 3) were trying to cope with poor requirements whilst also dealing with developing in internet time. In the MIS the developers (circa 2 engineers) are engaged as suppliers configuring and adapting their own COTS product and testing is mostly conducted by the customer (one person with ad-hoc support from departments) as acceptance testing. In the bespoke tool development the developers and testers work in an interdisciplinary way (circa 3 people).

From the beginning it was clear that our industrial colleagues were not familiar with goal based requirements methods. We began by creating preliminary, incomplete, sketches of the system goal graphs, paying attention to the motivation goals. The analyst (one of the authors) worked with key staff members (usually project or product managers) to create goal graphs with emphasis on the motivation goals. At this stage a common form of analysis was adopted guided by the marketing principles of "pain" and "gain" analysis [5] and the things to be maintained by the application of the new software. This approach appealed to the staff co-opted to help write the motivation goals and also the executive staff who were asked to review the sketches. The sketches were grounded in domain sketches based on Jackson context diagrams [3] and the combined goal and domain sketches were used to promote a shared understanding of the project. In each case the cost of reaching this point was a few staff days.

The creation of behavior goal refinements by the analyst and partner was more difficult than the motivation goal refinements for two reasons:-

1. The tendency when faced with semantic entailment to create activity sequences (project plans) rather than proper behavioral goal refinements. The key to avoiding this is to remember that behavior goals are satisfied by the run-time behavior of the system to be whilst motivation goals are satisfied by the coming into being of the system to be.
2. The tendency to over-elaborate when there is not enough information. The keys to avoiding this are to remember the agile discipline of focusing on the immediate stage, inserting TBD goals and using lazy refinement as much as possible.

Whilst each of these problems can be mitigated it is clear that the analyst's training of is of paramount importance. This issue is one that we have carried forward into our future work in the belief that more attention must be paid to process and tool support (such as incorporating UML state and activity diagrams to guide the behavior goal refinements). However whilst the creation of behavior goal sketches was difficult they provided good quality specifications for the developers and testers. The practice of using composition tags also maintained focus on creating behavior goals from essential use cases [6], [7]; with the consequential advantage of distilling the essential requirements from the design details. The essential behaviors are made into real behaviors by the use of composition of design constraints (such as user interface particulars). This distilling of essential requirements has already been found to be particularly advantageous in the preparation of acceptance testing, especially in the tool development.

In the case of the MIS the initial system goal graph was developed and presented to the managing director for approval with resulting productive discussion about the scope of the MIS. The project is very large and it is expected that commissioning will take over a year. Analyzing the whole behavior up front is considered futile because it is expected that there will be a degree of concurrent process re-engineering. Therefore the plan is to work in a series of stages each preceded with a goal sketch that is produced on a just in time basis. The goal sketch for each stage provides the acceptance test criteria and the accumulated stage goal graphs refactored into the evolved system graph will provide the legacy record so that future extensions can be considered in the light of the rationale for the commissioned system. This exercise will be reported separately. Here the point is that goal sketching is being applied to the requirements analysis and the preparation of acceptance tests.

6 Related Work

Work has been done on how some of the best practices of requirements engineering could enrich agile approaches [8]. The practices described include customer interaction, requirements analysis, non-functional requirements and managing change. The paper suggests that ways of adapting requirements management practices for agile processes are needed. However note that [8] simply describes how to include requirements engineering methods in an agile development process, rather than describing a method for requirements engineering that is agile. Similarly Nawrocki et al propose a way in which documented requirements could be introduced into XP through the use of automated tools, the Web and on-line documentation [9].

Orr suggests that it is possible to combine requirements and agile development by using up-to-date hardware and sophisticated graphical software [10]. Prototypes are suggested as a way to improve the process of defining requirements. However this work emerged from practice rather than from a theoretical technique such as goal-oriented requirements engineering.

Ambler describes an agile approach to modeling requirements, utilizing approaches such as the planning game of Extreme Programming and the Scrum methodology [11]. Similarly Leffingwell and Widrig discuss an agile requirements technique that is based on use-case specifications [12]. They also provide guidelines for selecting which requirements method (extreme, agile, or robust) is right for a particular project. However, again these approaches do not have a formal method such as goal-oriented requirements as a basis.

7 Further Work

The work reported here concerns the basics of the goal sketching technique. We are undertaking the following investigations to advance the work:-

1. Application to more industrial projects to confirm the applicability and practicality of the method for use in Agile projects.
2. Development of tools to accelerate the speed of sketch drafting and refactoring. In this area we are currently exploring the use of UML diagrams such as activity diagrams as these are well suited to the problem of determining behavioral goal refinements.
3. Development of metrics and supporting tools to exploit the structure of goal graphs in conjunction

with expert judgments to quantify the adequacy and feasibility of the intention expressed in a goal graph. It is anticipated that this will contribute significantly to the better planning of project stages and the improved sharing of expectations amongst the key stakeholders.

4. Tools to export goals sketches into KAOS for cases that justify upgrading from a goal sketch to a rigorous KAOS analysis.

8 Conclusions

In this paper we have presented a goal sketching technique that is intuitive and easy-to-read for project managers, sponsors and developers as well as for users. The importance of this is to empower the key decision makers when negotiating project decisions. The technique can be used together with Agile programming to develop software in internet time. We have given an example of the technique and we are currently testing its feasibility by application to a number of industrial systems development projects. Whilst more validation is needed we can report that agility has been observed, with an ability to adapt to evolving requirements and to cope with unresolved requirements. In a number of cases we have been able to construct initial goal graphs to show the motivation goals within a couple of days and by relying on lazy refinement we were able to add the detail for early sprints in another couple of days; in all cases fully involving the non-engineering stakeholders. The goal graphs became key artifacts for planning and negotiating subsequent sprints and so the key assumptions from the earliest sprints remained evident.

This research is the first account of the goal sketching technique. The technique can be used with any form of development method to provide a practical and complete method for internet time software development.

The technique has been empirically tested on a number of industrial projects. These trials support the claim that we can successfully develop evolvable systems using this technique. Our experience suggests that the technique is sympathetic with the real world needs of industrial software development. We have also partially automated this technique by implementing a lightweight tool called KAOS Lite for goal sketching and automated data collection.

9 Acknowledgements

The authors would like to acknowledge their industrial collaborators. In particular: Nick Gradwell, Product Manager of ClearPace Ltd; Ian Lycett KTP Associate at Image Farm Ltd; and Sean O'Mahoney and Richard Olearczyk of Surelines Audit Services Ltd.

8 References

[1] Rising, L., Janoff N., "The Scrum Software Development Process for Small Teams," *IEEE Software* July/August 2000.

[2] Boehm, B., A Spiral Model of Software Development and Enhancement, *Computer*, May 1988, pp. 61-72.

[3] Jackson, M., Problem Frames: Analysing and Structuring Software Development Problems, Addison Wesley, 2000.

[4] IEEE, "IEEE Guide for Developing System Requirements Specifications" IEEE Std-1223 (1998).

[5] Deep, S., Sussman, L., Sandler Institute, "Close the Deal: 120 Checklists for Sales Success"

[6] Constantine, L., "The Case for Essential Use Cases," Object Magazine, May 1997. SIGS Publications.

[7] Larman, Craig 1998. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design. Prentice Hall. Upper Saddle River, NJ.

[8] A Eberlein and J Cesar Sampaio do Prado Leite, Agile Requirements Definition: A View from Requirements Engineering, International Workshop on Time-Constrained Requirements Engineering TCRE'02, Essen, Germany, Sep, 2002.

[9] Nawrocki, J. R., Jasiñski, M., Walter, B., and Wojciechowski, A. 2002. Extreme Programming Modified: Embrace Requirements Engineering Practices. In *Proceedings of the 10th Anniversary IEEE Joint international Conference on Requirements Engineering* (September 09 - 13, 2002). RE. IEEE Computer Society, Washington, DC, 303-310.

[10] Orr, K., Agile Requirements: Opportunity or Oxymoron? *IEEE Software, 21, 3* (2004), 71-73.

[11] S. W. Ambler, Agile Modelling: Effective Practices for EXtreme Programming and the Unified Process, John Wiley & Sons, 2002

[12] Leffingwell, D. and Widrig, D. 2003 *Managing Software Requirements: a Use Case Approach*. 2. Pearson Education.