Teaching Goal Models in Agile Requirements Engineering

Antonio Lopez-Lorca
School of Computing and
Information Systems
The University of Melbourne
antonio.lopez@unimelb.edu.au

Rachel Burrows
School of Computing and
Information Systems
The University of Melbourne
rachel.burrows@unimelb.edu.au

Leon Sterling
School of Computing and
Information Systems
The University of Melbourne
Emeritus Professor for Design Innovation
Swinburne University of Technology
leonss@unimelb.edu.au

***Abstract*—Software engineering courses continually strive to maintain an excellent teaching curriculum that provides students with the agile skills as per industry needs. A particular challenge of teaching requirements engineering is capturing and communicating software requirements without killing team agility with excessive documentation. In many projects, requirements can be ambiguous and inconsistent. It is important to find a middle ground between completely by-passing requirements documentation and writing a complete Software Requirements Specification. In this paper, we report our experiences, presenting a guideline for students and educators who wish to adopt goal modelling, a lightweight approach to requirements elicitation and modelling, for agile requirements engineering. This is an efficient technique that also represents a good boundary object to support discussions between the developers and non-technical clients. Finally, we outline discussion points regarding where goal models could fit into other agile practices.**

I. Introduction

Agile methods are now extensively adopted in industry. Students leaving university need to be fluent in agile methods to be *industry ready*. Consequently, agile curriculum developers strive to effectively distill and incorporate a wealth of agile methods and practices into the offerings of their software project courses.

There has been successful progress in transitioning educational courses that address these needs [5], [8], [14], [20]; however, these same initiatives also highlight a number of challenges. The focus of this paper is on one particular challenge: teaching students to effectively capture and communicate their software requirements in an agile environment. This is often a difficult task in any situation be it academic or industrial, as requirements can often be ambiguous, subjective and difficult to interpret. With the industrial adoption of agile, any new requirements elicitation technique will only be adopted if it is compatible with the principles outlined in the Agile Manifesto. That is, there is a clear need for lightweight models that support communication between stakeholders and all those who are involved in the development project.

The learning curve to adopt effective requirements elicitation techniques in the classroom is steep. Most students do not have prior industrial experience and require guidance in communicating with their client and maintaining a project level awareness with their team throughout the project.

In light of these challenges, we incorporated goal models into our existing software engineering subjects. Goal models were originally an agent-oriented methodology construct [15]. They present a hierarchical structure of the goals of the software system at a high-level of abstraction. The models capture roles of all stakeholders involved in the system, the functional goals of the system, the quality goals of the system, and emotional goals, which represent how people want to feel when interacting with the system. Over the past ten years, the elicitation methods for the models have been streamlined from those described in Sterling and Taveter's book [15], for example [9], [10].

Our goal models differ from other goal-oriented requirements engineering techniques such as *i** and KAOS [19], [18] in that they are intended to be boundary objects to support discussion with non-technical clients. Additionally, our goal models include the emotional needs of stakeholders as emotional goals. Due to the intended use of goal models we do not concern ourselves with technical notations such as the AND/OR decomposition of goals. This would be important in automated approaches, where requirements can be automatically generated from the models [18]. Similarly to us, *i** also supports modelling emotional concerns as *soft-goals* [19]. However, recent research [10] shows that stakeholders tend to understand better a problem described using a goal model than the equivalent one expressed in *i** notation.

Our students use goal models in their capstone projects, where they interact with external clients to tackle real-world projects. We have capitalised on the high-level of abstraction of goal models to use them as a shared artifact for communication between students, their supervisors, and their non-technical clients. It is timely to document and reflect on the value that goal models bring to software engineering teaching especially since our approach has now been refined and taught to approximately 300 students – across three masters subjects and over two and a half years. The contribution of this paper is therefore twofold: (i) we reflect on the value that goal models bring to our agile software development subjects and (ii) we provide a guideline for educators and students who wish to incorporate goal modelling into their software engineering projects.

The rest of the paper is organised as follows. We examine published work in section II. In subsection III-A we present our approach to elicit the elements of the goal model. In subsection III-B we detail the process to build goal models. In section IV we discuss how we integrate goal models with agile processes in our teaching. In section V we conclude the paper and we sketch our future work.

II. Teaching Trends in Requirements Engineering for Agile

Traditionally there has been a misalignment between how requirements engineering evolves in industry practices and how it is taught [6]. There is a lack of recent research on the topic, as much reported work is over 10 years old, and based on a traditional waterfall approach to requirements engineering.

With the widespread adoption of agile practice by industry, the need to feed these practices into education is imperative. In fact, many teaching approaches now emphasise agile learning outcomes requiring students to gather and analyse 'Just Enough' requirements 'Just in Time'. With regard to requirements elicitation, there are many methods that are well aligned with agile principles. For instance, Use Cases, Goal-based approaches and Scenario-based approaches have been adopted in industry due to their capabilities for representing requirements in a lightweight and communicable form [20], [13]. The Agile Manifesto[1] emphasises *working software over comprehensive documentation*. Students often take this as an excuse to completely ignore documentation. Goal models represent a good compromise as they provide a valuable shared understanding between client and developers at very low cost.

Increasingly, effective teaching strategies for agile techniques are being created and adopted. Some agile teaching experiences elaborate on the benefits of problem-based learning to develop team communication skills [5]. It is also common practice to place students into teams to work together to gather requirements for a particular problem and implement a software solutions. To teach initial requirements elicitation, many have detailed role playing approaches [17], [12], [21] to learn and practice oral communication skills; namely, interviewing, facilitation and negotiation skills. Although most courses are taught using co-located teams, experiences have been reported of applications of requirements engineering in a global software development project [4].

Students tend to be uncomfortable dealing with ambiguity as they struggle to formulate a problem based on a real-world 'mess'. In [1], the authors teach students to deal with unknown (incompleteness) and unknowable (unpredictably changing) requirements. Other experiential papers also introduce realistic and 'messy' elements of requirements engineering practice, such as ambiguity, uncertainty, confusion, fear, collaboration and corporate politics [11], [2]. While these two efforts are not in the context of an agile software development project, they work towards bridging the gap between well-defined and correctly specified university assignments and ambiguous and inconsistent real-world projects.

One challenge that remains in both industry and educational arenas is to bridge the gap between the separate communities of practice that exist. The creation and use of requirements artefacts needs to support knowledge sharing between all members of the project team—including a variety of external stakeholders. In [14], the authors reiterate these challenges and issues that agile requirement engineering practitioners face in industry in 2017. One of the six identified key challenges is, again, maintaining a project-wide level of awareness during the implementation of complex requirements. The authors suggest defining a product vision and using visualisation techniques to communicate with the client while maintaining focus.

III. Goal Models in Requirements Engineering

In this section we introduce two techniques that we teach in our lectures for requirements elicitation and modelling. The first one is *do, be, feel* lists and the second one is goal modelling. While we use both techniques together, they are flexible enough to be treated as two separate activities.

We will outline the steps involved in the technique and illustrate each step in action with an example from an internal project. The main objective of the project was to re-imagine the faculty space in a faculty of design at a Melbourne university. The faculty wanted to promote a positive culture and initiated an inclusive process whereby all faculty stakeholders could contribute their ideas. The process that we ran represented a lightweight method to understand the problem domain and capture the core roles and goals. While the design of faculty space is not a typical software engineering problem domain, we feel that it illustrates well our techniques. We believe that goal models are adaptable to any problem domain and as such are not only limited to representing goals of a software application. Furthermore the chosen problem domain is easily understandable by most, and it comprises a variety of roles and goals to demonstrate each step effectively.

A. *Elicitation of goals*

The purpose of the requirements elicitation activity is to uncover the roles and goals of all software project stakeholders. To do this, we teach students to run a workshop to elicit *do, be, feel* lists. The activity is a lightweight, interactive and adaptable way of capturing diverse ideas from a group of people. Typically, the workshops can be conducted in 30 minutes, but their duration varies. In our experience, they can be as quick as 15 minutes, or occasionally they can be stretched over multiple sessions to ensure the inclusion of all stakeholders.

1) Preparation and Setup: In preparation for the workshop, it has to be decided who will be the workshop facilitator, who also typically acts as scribe. For workshops with a large number of attendants, it might be necessary having a second person recording the ideas to ensure elements are captured quickly as they are being called out. Prior research and experience in the problem domain, including any relevant terminology,

[1] https://www.agilealliance.org/agile101/the-agile-manifesto/

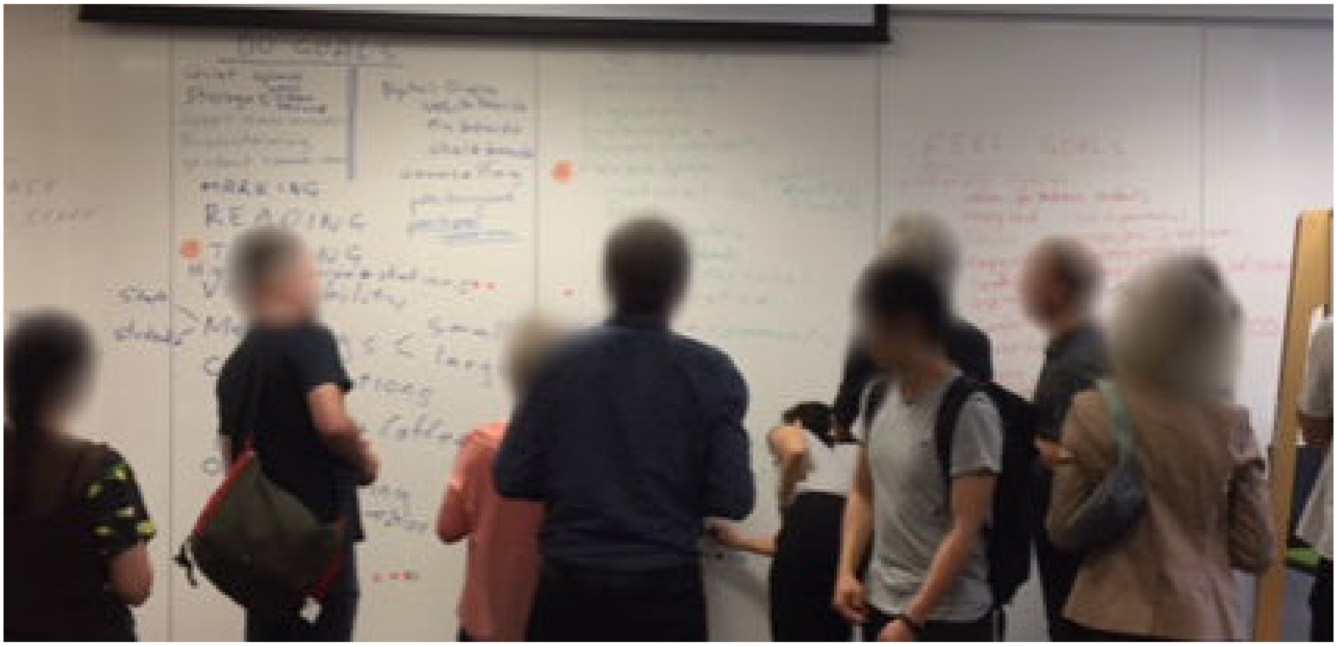

Fig. 1. A do–be–feel workshop in progress

are important to ensure that the facilitator is well prepared to understand the workshop discussions. Arrangements need to be made to recruit participants who are able to advocate for a variety of roles and viewpoints of the problem. It helps to have support from upper management to get staff at all levels engaged in the activity. The activity requires access to a whiteboard and preferably four differently coloured whiteboard markers.

In our example project, the main objective was to improve the faculty space. The workshop was scheduled to last 30 minutes and participants were faculty members spanning all organisational levels and roles. We had the support of the Dean of the faculty who introduced the activity and helped to attract participants to the workshop.

2) Introducing the Activity: At the start of the workshop, the facilitator welcomes participants and explains the purpose of the activity. The facilitator then describes and motivates the three categories (*do, be, feel*), and the fourth stakeholder category (*role*) that have been written as headings on the whiteboard. *Do goals* correspond to functional requirements, *Be goals* correspond to their quality requirements, *Feel goals* correspond to the emotional needs of the stakeholders, and *Roles* correspond to any stakeholder involved in the system. The activity differs from a usual software engineering requirements elicitation process where emotional considerations are not typically considered. We have previously presented a case for the inclusion of emotional considerations in software engineering [10].

3) Populating the Lists: The facilitator guides the participants to contribute ideas for each category. The scribe captures these goals and writes them under the associated category in the assigned colour. This activity follows a standard brainstorming approach in that ideas should not be filtered out. This activity ends when participants cannot think of new ideas. In practice, there is no strict order and the conversation can flow organically between categories.

In our example project, the facilitator asked participants (i) what they wanted to do in the space, (ii) how they wanted the space to be, (iii) how they wanted to feel in the space, and (iv) who would use the space.

4) Adding Priorities to List Elements: An optional final activity is to prioritise the ideas using any lean technique, for instance dot voting[2]. Each participant is assigned a finite number of dots that they can assign to any of the goals based on their importance. Markers are given to participants to mark their assigned dots next to the idea on the whiteboard. This is a quick method to capture a rough idea of goals that are important while all participants are familiar with all group-generated ideas. The number of dots assigned is a judgment call but typically we allow for 5-7 dots per participant.

5) Closing the Activity and Capturing the Results: The facilitator thanks everyone for their time and contributions and captures the results on the whiteboard. They explain that once the results have been processed, the participants will be invited back to review the result and will be given the opportunity to give feedback and provide any further clarifications.

For our example project, Figure 1 shows the workshop in progress with four whiteboard areas being populated with goals and roles. The whiteboard was recorded at the end of the workshop.

[2] http://dotmocracy.org/dot-voting/

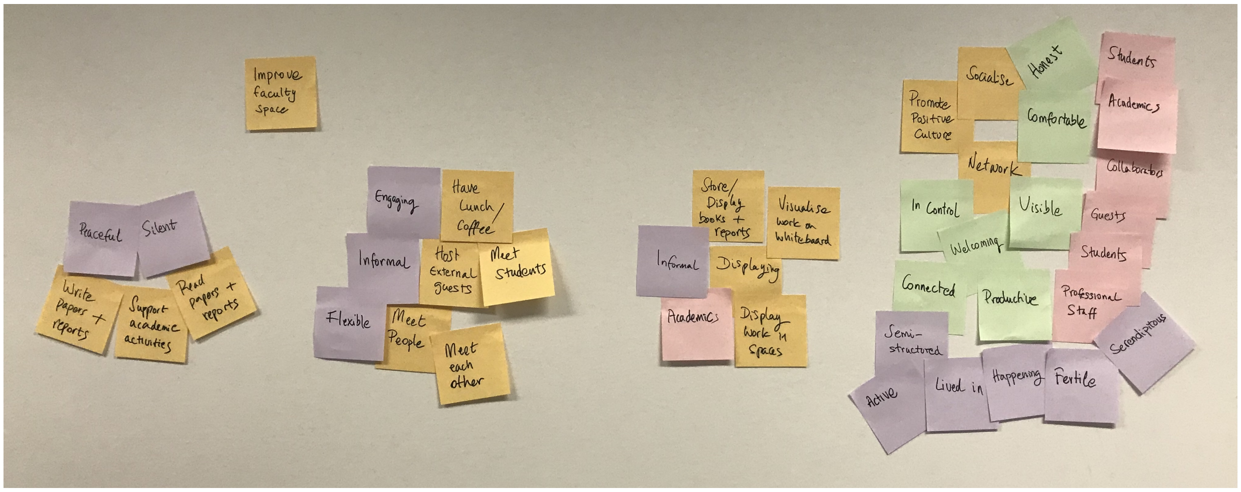

Fig. 2. Clustering of elements for the faculty meeting space project

B. Goal Model Construction

This sections details how results from the goal elicitation activity are used as input to construct a goal model. Building goal models is a low-cost activity that typically can be completed under a day. We elaborate on our approach for dealing with ambiguous requirements. We find that additional questions that are revealed from building the model provoke discussion and solidify understanding. The steps detail clustering and hierarchy definition activities. The output of this activity is a model that provides a high-level overview of the system goals that can be used as a lightweight communication tool with stakeholder organisations and between members of the development team. We continue to use the design of the faculty space as our running example.

1) Reviewing elements in the lists: The outcome of the workshop activity was a set of four lists categorised into *do* goals, *be* goals, *feel* goals and roles. These are sets of elements without clear relationships within or between categories. The first step is to discuss each element together as a team to ensure that there is common understanding. It is important that the facilitator of the workshop is present for this activity as they may be able to elaborate on confusing aspects based on discussions held during the workshop. Any questions, ambiguities, differing interpretations or assumptions should be noted for later clarification with the client. Any duplicated elements, elements that should be broken into sub-elements, or redundant elements should also be noted for discussion with the client. At this stage, rewording elements may happen to maintain consistency. Following conventions when naming goals improves clarity. We suggest wording functional goals as active constructions starting with a verb; quality goals as attributes of the system, adjectives or adjectivised sentences; and emotional goals worded to complete the sentence *I want to feel...*

2) Clustering lists contents: Next the team performs a clustering activity to group related elements together. A good technique to follow to create the clusters are affinity diagrams[3]. Following this approach, the team would write the elements from each list on sticky notes of 4 different colours, each representing one of the list categories. The team would then group these sticky notes into clusters of related elements. This activity makes it easy to visualise the clusters that are heavier in emotional, quality or functionality aspects. Or, what clusters involve multiple stakeholders. Team members are working separately, but on the same physical space. Their actions will reaffirm (or contradict) individual conceptualisations, and will contribute to a shared understanding of the problem across all team members. It may occur that multiple instances of the same goal or role appear in different clusters. In the case of repeated roles, qualities or emotions, this indicates participation of the role in more than one functionality of the system, or that the particular consideration is more important for certain functionality, respectively. Repeated functional goals are harder to justify and will trigger further questions for the client. Once the clustering is completed, the team chooses a label for each cluster. Ideally, a representative functional goal that already exists within each cluster could be nominated for this purpose. Otherwise, a new functional goal should be created that encompasses all elements in the cluster.

In our example project, the list elements were reviewed by the requirements analysis team and four clusters were formed (see Figure 2). Each cluster represented elements associated with: *promote positive culture*, *displaying*, *meet people* and *support academic activities*. Some goals were divided into two; the goal for *have lunch / coffee and socialise* was divided into *have lunch / coffee* and *socialise*. Elements within the *support academic activities* cluster provoked questions amongst the team as it was unclear whether these activities were intended to be individual or group-based. As the answer to this distinction affected the clustering it was noted for later clarification with the client.

[3] https://www.interaction-design.org/literature/article/affinity-diagrams-learn-how-to-cluster-and-bundle-ideas-and-facts

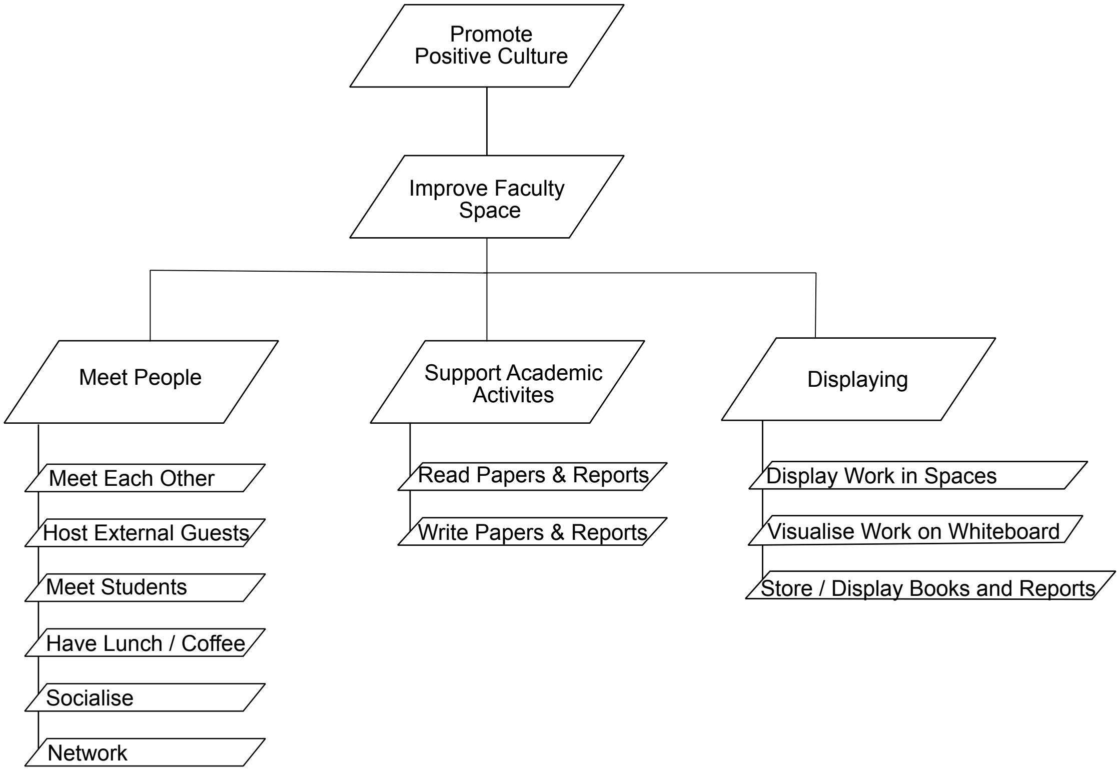

Fig. 3. Hierarchical structure of faculty meeting space goal model

3) Establishing the hierarchy: To establish the structure of the goal model the team constructs a hierarchical view of the functional goals. Subsequently, the goals in the functional hierarchy are associated with their corresponding roles, quality and emotional goals.

For each cluster, every functional goal within it is examined to determine the cluster's hierarchical structure of functionalities. Clusters that have only one functional goal will result in a single-element hierarchy. For multiple element hierarchies, the top-level functional goal (or root) will be the functional goal that we selected (or created) in the previous stage to represent the cluster. An effective way of establishing the functional goal hierarchy (both at cluster level and overall system level) is the application of *How/Why Laddering*[4]. In essence, given a hierarchy of functional goals, the sub-goals detail *how* the parent goal is achieved, the parent goal explains *why* its sub-goals are necessary. For instance, looking at a fragment of our running example shown in Figure 3, one mid-level goal is to *Promote Positive Culture*. If we ask the question *how do we promote positive culture?*, the answer could be by *meeting people*. Similarly, if we ask the question *why do we want to meet people?*, one reason is because we want to *promote positive culture*.

At this stage, there should be a set of disconnected sub-trees. The next step is to join them together into one hierarchical structure. A suitable functional goal should be chosen (or created) to be the root functional goal. It will become the parent of the entire model. If each hierarchy cluster corresponds to a unique aspect of the system (i.e. there is no hierarchy relation between them) then all clusters may be situated under the root functional goal. At this stage, further structural amendments may be required. For instance, goals that crosscut many other goals may be more suited as their parent goal, and consequently could be raised up in the hierarchy to reflect this. In other instances, some functional goals may need to be duplicated across multiple sub-trees, and possibly renamed to reflect this distinction (clarification with the client will ensue). This is an iterative process that is complete once a clear and meaningful structure is obtained.

For our example project, we first considered the *Meet People* cluster and the functional goals within it. The functional goals that belonged to this cluster (e.g. meet students, meet each other) were all considered to be directly sub-functional relative to the *Meet People* goal and consequently depicted as such in the figure. A similar process and result was performed to identify the hierarchy for the remaining three clusters.

We then decided that high-level goals corresponding to the three clusters were distinct enough to be positioned as three separate functional goals underneath the root goal. These three goals were *Meet People*, *Support Academic Activities*,

[4] https://dschool-old.stanford.edu/groups/k12/wiki/afdc3/HowWhy_Laddering.html

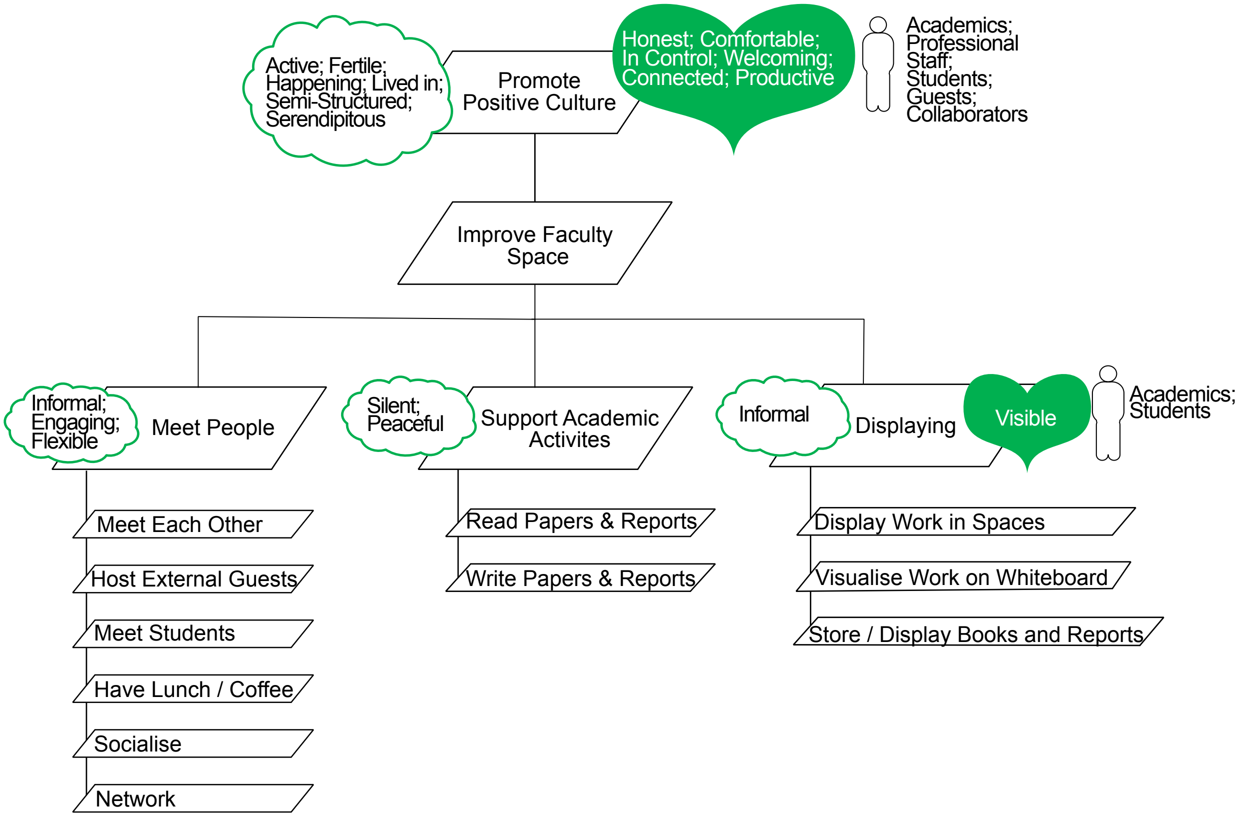
Active; Fertile; Happening; Lived in; Semi-Structured; Serendipitous
Promote Positive Culture
Honest; Comfortable; In Control; Welcoming; Connected; Productive
Academics; Professional Staff; Students; Guests; Collaborators
Improve Faculty Space
Informal; Engaging; Flexible
Meet People
Silent; Peaceful
Support Academic Activites
Informal
Displaying
Visible
Academics; Students
Meet Each Other
Host External Guests
Meet Students
Have Lunch / Coffee
Socialise
Network
Read Papers & Reports
Write Papers & Reports
Display Work in Spaces
Visualise Work on Whiteboard
Store / Display Books and Reports

Fig. 4. Final goal model for the faculty space re-imagination project

and *Displaying*. The fourth goal, *Promote Positive Culture* was different as it crosscuts the other three goals. In fact, promoting a positive culture was a reason why the three other high-level goals were being supported in the meeting space. Consequently, this was positioned as the new root goal and above the previous root *Improve Faculty Space*. The resulting hierarchy is shown in Figure 3.

4) Adding in the roles, quality and emotional goals: In this step we add the remaining elements (roles, quality and emotional goals) to the newly created functional goal hierarchy. We find that the best approach is to firstly focus on the roles. This is important as the emotional goals depend not only of the functional goals but are also associated to specific roles.

For each role and functional goal in the cluster hierarchy, it has to be examined whether the role is responsible for or involved in the achievement of the functionality. If this is the case, the role should be associated with the functional goal. The roles are added to the diagram, following the notation style. Associations can be shown with visual proximity, as additional connectors such as lines or arrows add unnecessary clutter and make the model cognitively taxing for readers. If a role is associated to a functional goal, then it is also associated to its corresponding sub-functional goals. A key consideration here is the level of granularity of the role. Keeping the pairing at a high-level can indicate a broad relationship to many aspects of the system. Differently, pinpointing a role to a low-level goal can bring a contextual focus to part of the solution. In any case, placing a role should be a decision and should intend to reflect a meaningful piece of information about the system.

Once this is done, we can focus on the remaining elements, either the quality goals or the emotional goals. Which one goes first is arbitrary and a similar process is repeated. Assuming we consider quality goals next, we examine, for each quality goal and functional goal in the hierarchy whether the quality goal is important while achieving the functional goal. If the answer is yes, the quality goal should be associated with the functional goal. It is likely that a quality goal will apply to more than one functional goal, if this is the case, it is correct to duplicate it. If a quality goal has an impact on all the functional goals at the same level (in practice, roots of separate sub-trees), that quality goal should possibly be placed in the parent functional goal of those instead. For instance, following with our running example depicted in Figure 4, all the sub-goals of supporting academic activities, i.e. reading and writing papers, should be done in silence. Therefore, the quality goals silent and peaceful should qualify the parent functional goal, *Support Academic Activities*.

To add the emotional goals, we proceed similarly as for the quality goals, but with one distinction. For the emotional goals, it is relevant to distinguish the role who wants to feel like that (emotional goal) in relation to the functional goal. In this sense, emotional goals are related to both a functional goal and a role. This is obvious when we consider that there may be more than one role involved in a functional goal, and their emotional needs may be very different. For each emotional goal, role and functional goal in a cluster hierarchy, we consider whether the role wants to feel this emotion for this functional goal. A positive answer means that the emotional goal is placed between the functional goal and the role involved. Similarly as for quality goals, it should be considered whether emotional goals should be applied to subgoals or if it is warranted that they are moved to a parent level. Also, similarly to the quality goals, it is likely that emotional goals will need to be duplicated in various branches. For instance, in the example shown in Figure 4, we can see that everyone using the new space wants to feel *productive* and *comfortable*, among others.

For the sake of simplicity of notation, we suggest grouping as many quality and emotional goals inside a shape as practical. It is key to avoid overloading the diagram with notation, so non-technical stakeholders can understand it.

In our project example, we had a number of roles that would be associated with every functional goal. These roles, such as the role of an academic, were therefore placed next to the root goal. Similar decisions were made for the quality and emotional goals. Figure 4 shows the complete goal model for our running example.

5) Review and clarify with the client: Once the team has arrived at a first goal model draft, and no more internal feedback or changes are due to occur, it is time to present it to the client for review. A representative of the team should present the team's understanding of the system by verbally walking through the goal model, and explaining the notation where appropriate. This is a chance to gain feedback on any aspect of the solution, including the clarifications that have been collected by the team during the model building stage. One strategy is to add these clarifications explicitly as annotations in the goal model so they will not be forgotten during the discussion. The discussion should be structured in a top-down manner and cover each element in the model. Feedback from the client could indicate a missing element (incompleteness), or an inconsistency. This is all part of the process and should be noted and amended in the model afterwards. Any alterations improve the team's representation and understanding of the problem. At the end of the session, there should be a feeling that both clients and team are on the same page. The team gain feedback from the client that will be incorporated in the next version of the goal model. The process will iterate until no more changes are required by the client.

In for our example project, we reviewed the goal model with the client and asked for some clarifications. Some alterations were necessary, including a rewording of one functional goal and also in response to personal layout preferences for improved readability.

IV. Goal Models within Agile Development

In this section we elaborate on our teaching context and we look at uses of goal models throughout an agile software development lifecycle.

The combination of techniques that we detail in this paper, the elicitation workshop and goal model, has been developed over the last 7 years for research purposes and we have taught them since 2016 to students of Master's of Software Engineering at the University of Melbourne. Goal modelling has now been taught in 3 subjects with enrolment numbers for 2016, 2017 and 2018 approximately 300.

The Requirements Engineering subject has a large project component (60%) where students work in teams of 4-5 to produce a requirements specification for a real-world client. In the following semester, students enrol in the semester-long project, where they implement the requirements to finalise the projects initiated in the first semester. In the following year, students participate in teams of 10 in the year-long project, where they go through the complete lifecycle of a more complex project for a real world client, from requirements elicitation to deployment. The three subjects run in an agile fashion.

The decision to coordinate each project with an industrial client comes with additional overhead and risk on part of the teaching team [2]. We believe the benefits far outweigh the overheads as this setting exposes students to added complexity, as they learn to manage expectations and interactions with real clients, thus creating an invaluable learning experience with real impact.

The initial adoption of goal models in these subjects was instigated while transitioning from a waterfall to an agile approach. This transition moved away from traditional artefacts such as a Software Requirements Specification (SRS), which can easily have tens of pages and can be overwhelming for the client. Certainly, goal models fit better within the agile spirit[5]. They are easier to keep updated than SRS's and clients can give quicker feedback about them. From a teaching perspective, they represent an effective use of time for everyone involved.

Uses of goal models throughout the agile software development that are promising, and require further exploration, include:

1) Goal Models to Justify the Project: There is often an education gap, between the starting point of projects for students and the starting point of projects in industry. Typically, a student starting a project will be given precise instructions containing all they need to know to complete the task. This is not the case in an industry setting, where a team may need to present a case to management arguing why the proposed project should go ahead. To mirror this in an educational setting, our students produce a goal model as part of the business brief to motivate and scope the project [9].

2) Goal Models as Shared Conceptualisation: The initial goal model is validated with the client, who clarifies any

[5] https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

aspects and gives relevant feedback. After the feedback has been incorporated and the client is satisfied with the goal model, the model gets signed off by the client. It shows an agreement of what the problem is between the client and the development team.

3) Goal Models as Conversation Starters: Goal models provide a means of representing ambiguous, subjective and difficult-to-articulate requirements. Presenting this hierarchical model to the team and stakeholders acts as a conversation starter to navigate important goals, how they are decomposed into sub-goals, and address any ambiguity. Maintaining a conversation to cover the high-level goals also avoids over-emphasis on implementation-level details too early, which we find is a common pitfall in initial client meetings.

4) Goal Models to Support Prioritisation: The goal model also provides a visual representation of the individual capabilities of the sought system. It supports the discussion of priorities of goals in the system (for instance by allowing stakeholders to assign resources to the goals). This will be key for the development team to plan their sprints. Once goals have been prioritised, they can be color coded to indicate the priority or be decorated with priority values.

5) Goal Models Leading to User Stories: Part of the inception period in agile is the generation of user stories. Having the goal model gives us a direct mechanism for derivating user stories (in the spirit of the work presented in [16]). A possible template of user stories adapted from [3] is:

- As a *[role]*, I want to *[do something]*, so that *[I get some benefit]*

This maps naturally with the goal model. Roles fit in the [role] slot, functional goals and quality goals fit in the [do something] and emotional goals or maybe a super-goal fits in the [I get some benefit]. An example from our running case study could be:

- As an *academic*, I want to *meet other academics in an informal and engaging environment*, so that *I help promoting a positive culture in the department*

These user stories will be very high level, as the goal model is intended to capture the motivation of the project. As the development team move into working in a particular user story, they will have to decompose it into finer grain ones that will become part of the sprint backlog.

6) Goal Models as Project Management Tool. : Another way of using goal models is to track or report progress by colour coding the goals. It can be maintained internally by the development team to have a visual reference of the progress of the project. One example of this usage is illustrated in [7]. Goal models were displayed in the lab, where 40 students worked on the implementation of a game to contribute to raise awareness of Asperger syndrome. The goal model was maintained over two semesters, by two cohorts of 40 students each.

The development team can also use goal model to visually show progress to the client at the end of each sprint, in the review meeting. As the client is already familiar with the goal model, colours of the goals gives a very quick idea of the progress. Our students use colour coding during their retrospective meetings at the end of each sprint to reflect on their project progression.

V. Conclusions and Future Work

In this paper we present our experience teaching students a combination of requirements elicitation and modelling. We deem our approach particularly suitable for agile developments, as over 3 years, 300 students working on agile software engineering projects with real-world clients have applied it. Goal models are easily understandable by non-technical stakeholders, becoming an excellent boundary object to support discussions between client and development team. Moreover, they have other lean uses in the software development lifecycle, ranging from project management to generation of user stories.

Our next step will be to develop a tool to support the complete process. There are a range of tools that support the creation of any diagram, in particular goal models. However, they tend to be offer too many options which are unnecessary for goal modelling. They also lack support of the combination of techniques. In 2017, two teams of students developed two proofs of concept for a goal model editor. These prototypes were very good first steps towards understanding the requirements of a tool to support the complete elicitation and modelling process. In 2018, another larger team of students is working on an extended version of this tool based on the lessons learned from the 2017 experience.

We welcome discussion on uses of goal models throughout the agile development lifecycle and the capabilities of a future tool to support this.

Acknowledgements

The work was partially supported by ARC Discovery project DP160104083

References

[1] R. J. Barnes, D. C. Gause, and E. C. Way. Teaching the unknown and the unknowable in requirements engineering education. In *Requirements Engineering Education and Training, 2008. REET'08.*, pages 30–37. IEEE, 2008.

[2] D. Callele and D. Makaroff. Teaching requirements engineering to an unsuspecting audience. In *ACM SIGCSE Bulletin*, volume 38, pages 433–437. ACM, 2006.

[3] M. Cohn. *User stories applied: For agile software development.* Addison-Wesley Professional, 2004.

[4] D. Damian, B. Al-Ani, D. Cubranic, and L. Robles. Teaching requirements engineering in global software development: a report on a three-university collaboration. In *Proc. 1st International Workshop on Requirements Engineering Education and Training (REET 2005)*, 2005.

[5] K. Lundqvist, C. Anslow, M. Homer, K. Bubendorfer, and D. Carnegie. An agile conversion masters degree programme in software development. In *In Proceedings of the ACM Special Interest Group on Computer Science Education (SIGCSE), Baltimore, Maryland, USA*, pages 846–851. ACM, 2018.

[6] L. Macaulay and J. Mylopoulos. Requirements engineering: an educational dilemma. *Automated Software Engineering*, 2(4):343–351, 1995.

[7] J. Marshall. Agent-based modelling of emotional goals in digital media design projects. In *Innovative Methods, User-Friendly Tools, Coding, and Design Approaches in People-Oriented Programming*. Hershey, PA: Information Science Publishing, 2018 – To Appear.

[8] A. Martin, C. Anslow, and D. Johnson. *Teaching Agile Methods to Software Engineering Professionals: 10 Years, 1000 Release Plans*, pages 151–166. Springer International Publishing, Cham, 2017.

[9] T. Miller, B. Lu, L. Sterling, G. Beydoun, and K. Taveter. Requirements elicitation and specification using the agent paradigm: the case study of an aircraft turnaround simulator. *IEEE Transactions on Software Engineering*, 40(10):1007–1024, 2014.

[10] T. Miller, S. Pedell, A. A. Lopez-Lorca, A. Mendoza, L. Sterling, and A. Keirnan. Emotion-led modelling for people-oriented requirements engineering: Case study of emergency systems. *Journal of Systems and Software*, 105:54–71, 2015.

[11] G. Regev, D. C. Gause, and A. Wegmann. Experiential learning approach for requirements engineering education. *Requirements engineering*, 14(4):269, 2009.

[12] D. Rosca. An active/collaborative approach in teaching requirements engineering. In *Frontiers in Education Conference, 2000. FIE 2000. 30th Annual*, volume 1, pages T2C–9. IEEE, 2000.

[13] E. Schön, J. Thomaschewski, and M. J. Escalona. Agile requirements engineering: Systematic literature review. *Computer Standards & Interfaces*, 49:79–91, 2017.

[14] E.-M. Schön, D. Winter, M. J. Escalona, and J. Thomaschewski. Key challenges in agile requirements engineering. In *International Conference on Agile Software Development*, pages 37–51. Springer, 2017.

[15] L. Sterling and K. Taveter. *The art of agent-oriented modeling*. MIT Press, 2009.

[16] T. Tenso, A. H. Norta, H. Rootsi, K. Taveter, and I. Vorontsova. Enhancing requirements engineering in agile methodologies by agent-oriented goal models: Two empirical case studies. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 268–275. IEEE, 2017.

[17] J. Tuya and J. García-Fanjul. Teaching requirements analysis by means of student collaboration. In *Frontiers in Education Conference, 1999. FIE'99. 29th Annual*, volume 1, pages 11B4–11. IEEE, 1999.

[18] A. Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 249–262. IEEE, 2001.

[19] E. S. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*, pages 226–235. IEEE, 1997.

[20] D. Zowghi and C. Coulin. Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and managing software requirements*, pages 19–46. Springer, 2005.

[21] D. Zowghi and S. Paryani. Teaching requirements engineering through role playing: Lessons learnt. In *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, pages 233–241. IEEE, 2003.