

Analysing Requirements in a Case Study of Pairing

Yijun Yu
Department of Computing
The Open University
y.yu@open.ac.uk

Helen Sharp
Department of Computing
The Open University
h.c.sharp@open.ac.uk

ABSTRACT

Agile software development methods suggest that sharing tasks between a pair of developers has advantages over letting them work as individuals. This effect has been observed in designing and coding tasks too. However, it is not yet known whether or not pairing on requirements engineering tasks could deliver a similar benefit at an early phase of software projects. Based on first-hand experience in pairing development of the OpenOME tool, we analyse the evolving stakeholder requirements using the i*/Tropos method. Our findings show that sharing tasks of different roles becomes more effective when those roles are played by one stakeholder, possibly due to smaller communication delay at a shorter distance.

Categories and Subject Descriptors

D.2.1 [Software Engineering]: Requirements/Specifications

General Terms

Agile Software Development, Open-Source Development

Keywords

Pairing, i* methodology, Case Study

1. INTRODUCTION

One of the best known practices of Agile software development is “pair programming”: where one task is shared between two programmers, their understanding is improved because of earlier and quicker feedback [1]. A pair of programmers exchange their roles regularly. Because the programmers sit next to each other, watching a shared display, they are forced to communicate and provide feedback early.

Through programming aptitude tests, Lui et al found that the beneficial effects of pairs arise from both design and coding activities [2]. Daniels et al [3] reported a session where pairing was found to be effective in non-programming creative tasks. Although these earlier results are illuminating,

it is yet to be established whether pairing in requirements analysis is beneficial, and whether there is any interesting challenge to pairing requirements analysts.

One of the authors was involved in a software engineering research project, OpenOME, which aims to provide tool support for the i* requirements engineering method. Unlike many open-source projects in the public domain, the authors have direct access to its internal knowledge; and unlike projects where researchers have limited access to the logs, the OpenOME project provides us with a unique opportunity. We know which roles different stakeholders play in this project, and which rationale (goals and softgoals) is concerned when certain events occurred. Since the project is in the public domain, anyone can extract the time gaps of such events from the public project logs.

In this paper, we present three recorded scenarios to demonstrate the involvement of different developers as they play each of the different roles: researchers, developers and users. These scenarios each took a different amount of time from start to finish. The rationale behind the delays can be explained by considering how the roles were paired.

The case study benefits from the idea of four levels of requirements engineering for and in dynamic adaptive systems [19]. Berry et al argued that the four levels of requirements engineering involve: (1) the human who analyzes the general requirements/behavior of the system; (2) the system itself that can analyze and adapt to its environmental change; (3) the human who decides when, how and where the system should adapt; and (4) the human who does research for adaptive systems. Even though their ideas are oriented towards adaptive systems, we found its application to an agile RE process helpful. In our case, the four levels are reflected by supporting a requirements engineer as (1) User of an RE tool; (2) Developer of the RE tool; (3) Researcher of an RE methodology; and (4) Agent of an integrated validator of the tool that supports the RE methodology.

Although, the i* requirements language used in this study has been created before the OME tool development starts, it was not until the start of the OpenOME project that the researchers felt the need to analyse the requirements of the existing tool support (OME) and new tool support (OpenOME) using the i* language. It was partly due to the requirements language getting more mature to use, and partly due to the ‘open’ initiative by the project manager (Eric Yu himself) who was also one of the core researchers. As it turns out, there is still a strong element of pairing in the development of the research ideas along with their tool support.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Agile RE'11, July 26, 2011, Lancaster, UK.

Copyright 2011 ACM 978-1-4503-0890-8/11/07 ...\$10.00.

It is our hope that these examples can illustrate a research agenda for adopting one of the best known Agile practices, pairing, in requirements engineering.

2. COLLABORATIVE SCENARIOS

Like most open-source projects, the OpenOME project has a community of stakeholders. Some of them are researchers, some are users, and a few of them are actual developers. Unsurprisingly, the players of these roles have overlap: there are researchers who can develop extensions to the OpenOME tool, and there are users who can contribute code and patches.

According to the maintenance tickets in the project’s wiki, this community sometimes works together on requirements engineering activities. By tracking the changes involving these different roles we try to explain some observed sequences of events (scenarios) that happened in the project history, and try to illustrate the impact pairing has on the requirements engineering process.

We anonymised and simplified three of the real scenarios by aggregating their details as shown in Figure 1. The scenario involves four individuals: Jim is a researcher as well as a developer, Sarah is a developer as well as a user. They form the OpenOME development team. Mike is a user with some software development skills, and Tom is another user. At the top of this sequence chart, four different icons are used to represent these four individual stakeholders, whose characteristics are listed respectively in Table 1.

Table 1: Stakeholders of the Scenario

Name	Team distance	Roles	Has development skills
Jim	Internal	Researcher, Developer	Yes
Sarah	Internal	Developer, User	Yes
Mike	External	User	Yes
Tom	External	User	No

Qualitatively, we assess the ‘distance’ between different persons as whether or not their communications are internal or external to the boundary of the development team. When the communications between roles are happened inside one’s brain because the same person is playing different roles, the distance is zero and internal. In Table 1, we just indicate whether or not a person is internal or external to the team boundary in the ‘team distance’ column.

The swimlanes of the three scenarios are deliberately drawn on the basis of roles – Researcher, Developer and User, rather than on the basis of individuals. We highlight here how the different roles of the stakeholders are influenced by the pairing activities. On these swimlanes, one can see who is involved in which activity, ranging from creating a ticket, to handling a ticket by finishing the development task. The time gaps between these events are obtained from the ticket system on the project’s wiki.

These scenarios illustrate a history of three related tickets in response to consequent change requested by the users. Such requests often lead to iterations in the OpenOME development. For some of these changes, Table 2 lists their triggering roles as well as the rationale (the *why*).

Here we select three requirements related to the “layout”

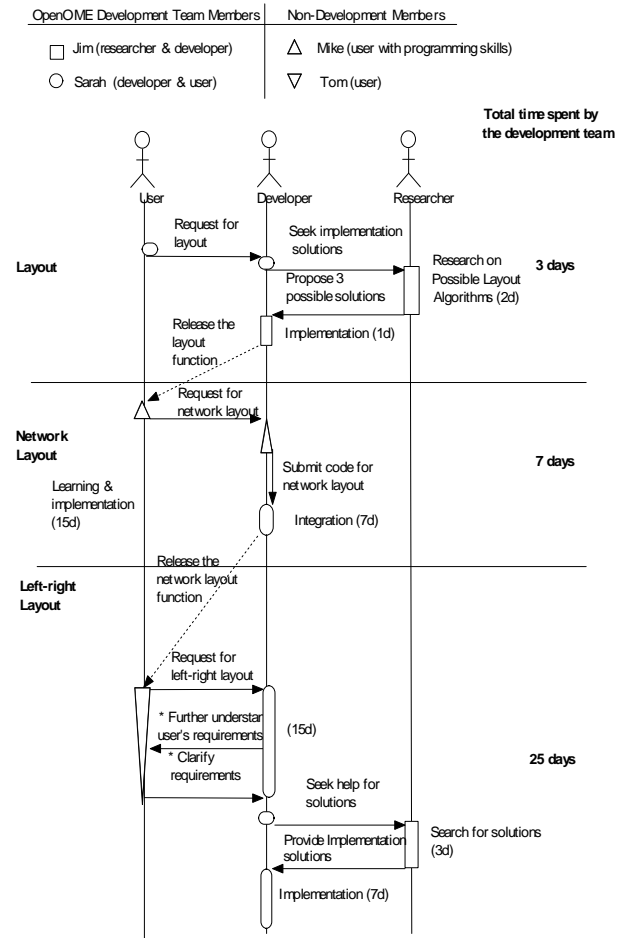


Figure 1: Scenario of OpenOME development illustrated by a sequence chart

functionality. The complexity of implementing them is similar. However, it took a shorter time (3 days) to achieve when all the project members were involved in the communication; and it took longer (7 days and 25 days) to achieve when some members involved were outside the research group. Interestingly, whilst both the 2nd and the 3rd scenarios involve external users, the 2nd one is much quicker in getting the requirements sorted out. In fact, the external user involved in the 2nd case could perform some development tasks, thus saving the delays that would have been incurred if he had to communicate with others playing the researcher role (which in this case was himself).

Here we select three requirements related to the “layout” functionality. The complexity of implementing them is similar. However, it took a shorter time (3 days) to achieve when all the project members were involved in the communication; and it took longer (7 days and 25 days) to achieve when some members involved were outside the research group. Interestingly, whilst both the 2nd and the 3rd scenarios involve external users, the 2nd one was much quicker in getting the requirements sorted out. In fact, the external user involved in the 2nd case could perform some development tasks, thus saving the delays that would have been incurred if he had to communicate with others playing the researcher role (which in this case was himself).

Table 2: Some changes requested in the OpenOME development

Iteration	Trigger	Delegation
Separating logic from presentation	User: simplify the entry of goal models through text editors	Researcher: define the syntax of a textual language that can express an i* goal model; Developer: implement a parser for the language.
Separating cross-cutting concerns in i* models	User: further reduce the complexity of the i* graphs; Researcher: identify early aspects from goal models [4]	Researcher: enrich the i* model with an aspect construct, as well as enrich the textual representation into Q7 [5]; Developer: change the meta-model of the i* notations, modify the parser to accept wildcard pointcut definition, implement the weaving algorithm.
Automatic layout of the i* graph	User: provide an initial layout of the i* graph loaded from the textual representation for presentation	Researcher: select a possible layout algorithm; Developer: integrate the chosen layout algorithm into OME
Integrating requirements models with the business process models	Researcher: integrate goal-oriented RE with the downstream business process workflow	Researcher: enrich with control dependencies [6]; Developer: convert enriched Q7 into business process workflow
Serializing the goal decomposition ordering	User: make the ordering of dependent subgoals persistent	Developer: convert the collection of children from unordered hashtable into an ordered vector, constrain the layout tool by creating hidden dependency edges
Visualizing the non-functional requirements on functional ones	User: simplify the quality problem determination tasks	Researcher: quantifying correlations between goals and softgoals and link the degree of quantification with visual clues; Developer: map the quantified degrees through colors in the HSV scale;

3. REQUIREMENTS ANALYSIS

In order to analyse the reasons for these differences in delay in the example scenarios, we performed a requirements analysis based on our internal knowledge of the project. The requirements modeling language we chose here is i*/Tropos [7, 9] which is a goal-oriented requirements engineering method for analysing the intentions and dependencies amongst stakeholders. The tool offers some analysis facilities to keep a focus on project objectives, to help understand the work demanded by different roles and to help understand how tasks should be prioritized to satisfy all stakeholders’ goals. The i* language represents actors, intentional elements (goals, softgoals, tasks, resources) and their relationships (dependencies, AND/OR decompositions) [8].

3.1 The i* language

In the i* requirements modeling language, an *actor* refers to a unit to which intentional dependencies can be ascribed. Actors may be further classified into roles, agents, and positions. A *role* is an abstract actor embodying expectations

and responsibilities, e.g. a developer. An *agent* is a concrete actor with specific capabilities and functionalities. An agent can play one or more roles. A set of roles that are packaged together to be assigned to an agent is called a *position*.

Actors ascribe intentional elements, such as goals, softgoals, tasks and resources. A *goal* is a condition or state of affairs in the world that an actor would like to achieve. A *softgoal* is similar to a *hard* goal except that the criterion for its achievement is not a clear-cut ‘yes’ or ‘no’. Softgoals are often used to model non-functional requirements (NFRs) [10, 11, 12, 9] or quality attributes. *Tasks* are used to represent the specific procedures to be performed by the agents, e.g. testing and debugging. They are used to achieve goals or to “operationalize” softgoals. A *resource* is a physical or informational entity about whether it is available. An intentional element can be decomposed into sub-elements through the *refinement* relationships. Elements of different actors may also have *dependency* relationships.

Using the above notations in i*, a requirements model can be organized into two kinds of modular views. A *strategic dependency* (SD) view describes the dependency relationships among various actors in an organizational context. A *strategic rationale* (SR) view describes how the intention of an actor can be achieved by fulfilling sub-elements [7]. In an SR view, a goal can be decomposed into sub-elements through *means-ends* (OR) links, i.e. the goal is satisfied if any sub-elements are satisfied. A task may be decomposed into sub-elements through *decomposition* (AND) links, i.e. all sub-elements of the task must be satisfied in order to accomplish the task.

Associated with each intentional element is a label of satisfaction. These labels can propagate from one intentional element to other elements in the model by following the dependency and decomposition links [13, 14]. Thus expressing the reasoning rules in the i* models helps to make prioritization explicit and traceable.

3.2 The Open initiative since OME

The i* modeling was originally supported by the Organization Modeling Environment (OME)¹. Even though the i* language can be used to analyse software requirements, the OME requirements have not been thoroughly analyzed using i* as the research language was not mature enough at the early stage of the tool development. OME does support the creation of custom modeling frameworks by the “power users” with the help of a publicly-available plugin mechanism. However, it was impossible for others to touch the code base of the OME tool. The OpenOME project, as the new prefix indicates, was started by Eric Yu in 2004, to make all artifacts of OME public, encouraging collaboration within the RE community. Since then, the initiative of the OpenOME project has evolved beyond simply providing a goal graph editing tool for the research community missioned by the OME project.

3.3 Analysis of OpenOME requirements

The requirements analysis of the OpenOME project presented here has an element of reverse engineering – to uncover the original intentions of the OME development – as well as an element of forward engineering – to prescribe the optative requirements envisaged by the OpenOME researchers – which amounts to changes to the existing require-

¹<http://www.cs.toronto.edu/km/ome/>

ments. In order to achieve agility in accommodating requirements changes, the development process for the OpenOME tool includes several changes to the existing OME development.

First, all design artifacts including the source code, should be accessible by the community. After spending 2 man-months to re-implement the API of a propriety knowledge base component of OME, all the source code was hosted at SourceForge² and we were using its public repository to share the artifacts among developers.

Second, we adopted some of the extreme programming practices to OpenOME development and turned it into an Eclipse plugin. We used an incremental test-driven development to add test cases for any changes, either implementation of new requirements or fixing existing implementation. These test cases allowed us to do regression testing and continuous testing. Furthermore, we employed the CruiseControl process for continuous integration to support iterative development of rather short intervals (daily and weekly). The results of building, testing and releasing were presented to the whole team through a Web-based dashboard.

On the other hand, users started to learn from the dashboard which version of the system can be used to choose a stable version of higher quality or to choose a development version of more functionality. These forms of feedback are light-weight but useful as they directly reflect the progress of the tasks, supporting the Agile processes. The above tasks were carried out by developers aiming to achieve the “Agility” softgoal (Figure 2).

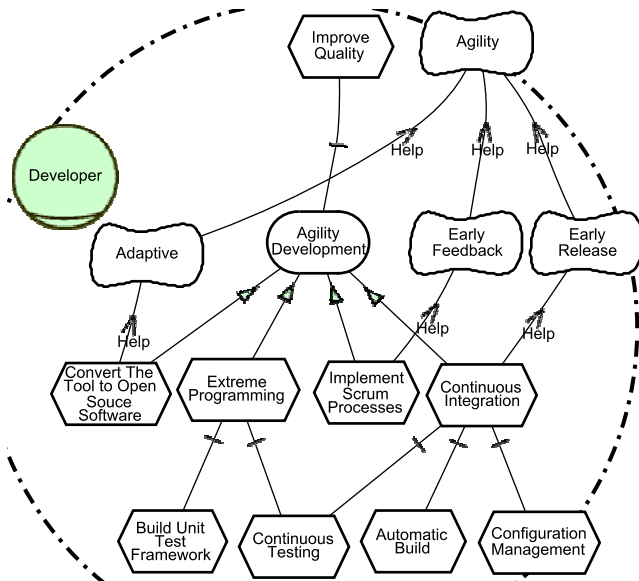


Figure 2: Agility requirements for developers

In a strategic dependency analysis, we found that the developers’ agile tasks could have impact on the researchers and users by following the strategic dependency links. Since different variations of the initial RE methodology are being actively developed, specifically by customising different application domains, an RE researcher would generally like to be able to adapt an existing methodology accordingly. For any change to the methodology, it is necessary to per-

²<http://www.sf.net/projects/openome>

form some empirical case studies to evaluate its validity in terms of external, internal, and construct validity criteria. To fit within an agile environment, feedback needs to be received quickly and regularly. To check its *external validity*, a researcher needs to get feedback from potential users; to check its *internal validity*, a researcher relies on the developers to provide early feasibility evaluation and agile implementation; and to check its *construct validity*, the researcher should quickly get early reviews by sharing the intermediate results with other researchers. This meta-requirements analysis is illustrated in Figure 3.

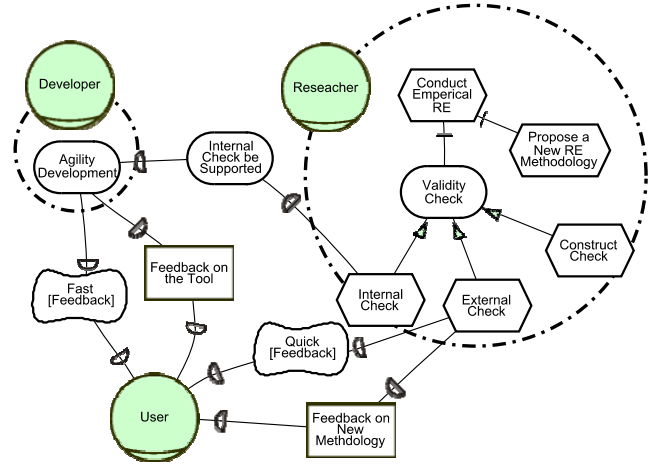


Figure 3: Changes in other roles due to agility

From this scenario, we observed that not all logical dependencies required by different roles led to heavy communication. Especially when a dependency involved two roles that are played by the same person, communication happened ‘inside the brain’. As an *intra-person* example, Jim was both a research and a developer for the Q7 language syntax and parser. Comparing the cases where dependencies happen among different people, the communication among people within a team is typically easier than communication among people across an organisational boundary. As an example of *intra-team* iterations, Jim was the researcher for graph layout algorithms for *i** graphs and Sarah was the developer for the import/export of *i** graph through Q7 and the integration between the 3rd-party graph layout tool. In comparison, as an example of *inter-team* iteration, Sarah was the developer for the ordering of subgoals whereas Tom was the user who requested such ordering.

Thanks to open-sourcing the code, some development tasks could now be fulfilled by a user of OpenOME. Since the local team developers often know their own requirements better than the remote team members, *intra-team* communication is more cohesive for the development of the additional functionality. Therefore it seems better to let capable users develop for their own goals. For that to happen, however, such users should be trained with adequate development skills as well as be granted access to all artifacts involved. As an example, since Mike (outside our team) had the ability to develop, when he needed a network layout functionality, it took little communication with Sarah (inside our team) for him to implement the functionality as an extension to the OpenOME tool.

An agile *i** analysis of the goals of stakeholders helps to

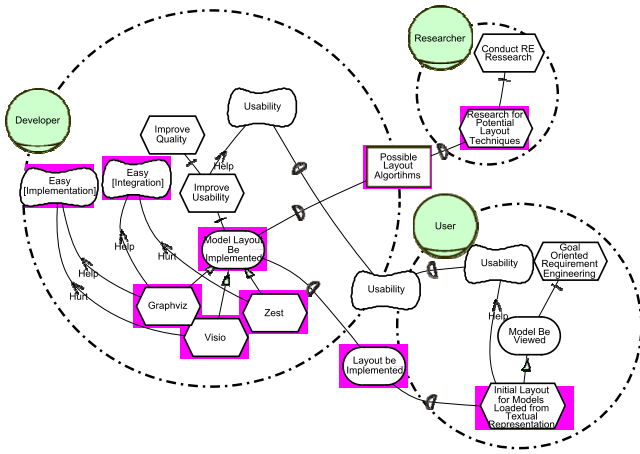


Figure 4: Changes required by the Layout goal

prioritise the TODO tasks. For example, after Jim (researcher) was initially consulted by Sarah (developer) for a graph layout algorithm, he was presented with three alternatives: integrating OpenOME with either one of the AT & T graphviz, Microsoft Visio, or Eclipse Zest widgets. Sarah found that these tools were all capable of doing graph layouts, however, they contributed differently to Sarah’s soft-goals “Easy to integrate”, “Easy to implement” (Figure 5). To meet Sarah’s primary goal, graphviz was chosen, so the tasks for integrating the layout tools in Visio and Zest were postponed for future iterations.

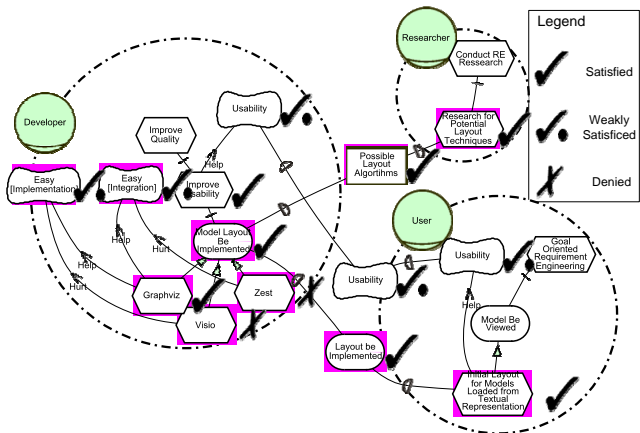


Figure 5: Prioritize alternatives for Layout

Later on, another user, Mike, found that a simple tree layout did not always look good. In several cases, a networked layout would work better. Even though to the development team Mike was just a user, he had sufficient programming background and decided to implement this networked layout function all by himself. This is workable since at that time, OME had been converted into an open source software and had gained a new name OpenOME. However, Mike was not familiar with how OME worked. He had to spend some time looking up existing OpenOME documentation posted on the Sourceforge website. After spending about 15 days to learn OpenOME, he managed to implement the networked layout algorithm as an extension to OpenOME. It then took

the development group about one week to integrate this new feature into OpenOME.

Some time later, another user, Tom, needed another way of laying out elements when integrating a requirements model with a business process model. He still wanted the tree structures, but at each level of the trees, the elements should be laid out from left to right according to a special criterion. However, he was neither a member of the development team, nor had he the skills to develop this feature by himself. As a result, he had to talk to one of the developers of OpenOME to explain what he wanted to achieve, and this process needed to go back and forth several times to ensure that the requirement was fully understood by both sides. In this case, the developer also needed to consult the researcher for solutions, which led to several more iterations. As a result, it took almost a whole month for this new feature to be implemented to Tom’s satisfaction.

4. RELATED WORK

A number of agile software development methodologies have emerged including Extreme Programming [1], Lean Development, Adaptive Software Development, Scrum, and Dynamic Systems Development Method [15]. Principles and practices such as efficient communication [16], continuous testing [17] and continuous integration are also gaining popularity. These agile methods share some common characteristics [15] that can be supported by goal-oriented requirements engineering as exemplified by this work: 1) *Visioning*: to help the agile process stay focused. The i* approach encourages the explicit expression of user, developers and researchers’ goals to ensure that prioritized tasks are related to or contribute to the satisfaction of those goals. 2) *Project initiation*: the organizational i* models can be seen as records for the project’s overall scope, objectives and constraints. 3) *Short, iterative development cycles and constant feedback*: one member taking multiple roles shortens the distance of communication so that feedback may arrive earlier.

An example agile development process is the Eclipse IDE [18], where the team uses the Eclipse for its own development. In comparison, we add the roles of requirements engineer and researcher into the analysis of our agile development process. In our own case study, including a researcher role explicitly into the development process, researchers can obtain early feedback from users and understand their needs, developers can integrate different research methodology and tool supports with little intra-team communication cost. It will be interesting to see whether our findings can be confirmed or not in much larger Agile developer endeavors such as Eclipse development.

5. CONCLUSION

Agility matters. This paper reports a pairing case by studying the change logs of the OpenOME development and by analysing the goals among different roles of several stakeholders. It is found that adopting the pairing practice for requirement analysis encourages stakeholders to adopt more roles in return. In addition, externalising project knowledge to a community of open-source development encourages more users to adopt more stakeholder roles so that they can solve their own problems more quickly.

We are aware of several limitations of this work. First, we have the internal knowledge of the project that agility and

openness are the goals set out by the researchers among the project team, which may not be the case for general software development projects driven by the interests of business. However, we envisage that this project can be a test bed for new follow-up research questions on other Agile practices such as Scrum [20].

Acknowledgement

The authors thank the contributions of the data from the OpenOME development team, including Xiaoxue Deng, Neil Ernst and Eric Yu from University of Toronto, Canada.

6. REFERENCES

- [1] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.
- [2] K. M. Lui, K. Chan, and J. Nosek, "The effect of pairs in program design tasks," *Software Engineering, IEEE Transactions on*, vol. 34, no. 2, pp. 197–211, march-april 2008.
- [3] J. Daniels, D. Cleal, and L. Hill, "Effective pairing – which tasks are most suited to pair working?" in *Software practice advancement (SPA '08)*, 2008, <http://www.spaconference.org/spa2008/sessions/session116.html>.
- [4] Y. Yu, J. C. S. do Prado Leite, and J. Mylopoulos, "From goals to aspects: Discovering aspects from requirements goal models," in *RE 2004*, pp. 38–47.
- [5] J. C. S. do Prado Leite, Y. Yu, L. Liu, E. S. K. Yu, and J. Mylopoulos, "Quality-based software reuse," in *CAiSE 2005*, pp. 535–550.
- [6] Y. Yu, J. Mylopoulos, A. Lapouchnian, S. Liaskos, and J. Leite, "From stakeholder goals to high-variability software designs," Tech. Rep. CSRG-509, 2005.
- [7] E. S. K. Yu, "Modelling strategic relationships for process reengineering," Ph.D. dissertation, University of Toronto, 1995.
- [8] —, "Towards modeling and reasoning support for early-phase requirements engineering," in *RE*. IEEE Computer Society, 1997, pp. 226–235.
- [9] J. Castro, M. Kolp, and J. Mylopoulos, "Towards requirements-driven information systems engineering: the tropos project," *Information Systems*, vol. 27, no. 6, pp. 365–389, 2002.
- [10] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishing, 2000.
- [11] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, Jun 1992. [Online]. Available: <http://www.cs.toronto.edu/yi-jun/literature/mylopoulos92tse.pdf>
- [12] L. M. Cysneiros and J. C. S. P. Leite, "Non-functional requirements: from elicitation to conceptual models," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 328–350, May 2004.
- [13] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with goal models," *LNCS*, vol. 2503, pp. 167–181, 2002. [Online]. Available:
- [14] R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Simple and minimum-cost satisfiability for goal models," in *CAiSE*, 2004, pp. 20–35.
- [15] J. Highsmith, "What is agile software development?" *CrossTalk - The Journal of Defense Software Engineering*, Oct. 2002.
- [16] L. Rising and N. S. Janoff, "The scrum software development process for small teams." *IEEE Software*, vol. 17, no. 4, 2000.
- [17] D. Saff and M. D. Ernst, "Reducing wasted development time via continuous testing," in *Fourteenth International Symposium on Software Reliability Engineering*, Denver, CO, November 17–20, 2003, pp. 281–292.
- [18] E. Gamma, "Agile, open source, distributed, and on-time: inside the eclipse development process." in *ICSE*, 2005, p. 4.
- [19] D. Berry, B. Cheng, and J. Zhang, "The four levels of requirements engineering for and in dynamic adaptive systems," in *Proceedings of the Eleventh International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ), Porto Portugal, 13–14 June, 2005*, 2005, pp. 95–100.
- [20] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Prentice Hall, Upper Saddle River, NJ, 2001.