

Managing assumptions during agile development*

Ireo Ostacchini¹ and Michel Wermelinger²

¹ Ramboll Whitbybird Ltd, UK

² Computing Department and Centre for Research in Computing, The Open University, UK

Abstract

The assumptions that underlie software development often go unrecorded and form part of the implicit rationale on which design and implementation decisions are based. These assumptions can fail at any time, with serious consequences. This paper presents a lightweight approach to assumption management (AM) designed to suit agile development.

Assumptions were monitored for three months within a small agile team. Two key indicators were proposed for measuring AM success but only one was detected in the research results. A number of strong correlations were found between properties of assumptions. Data collection largely depended on the subjective judgements of the first author, but they were validated with some success by his colleagues.

In some ways, assumption management was found to complement agile development. However, AM was not fully integrated into the team's development process, due to difficulty in adopting an 'assumption-aware' way of thinking. Suggestions are offered on how this transition may be eased, and on how others might wish to build on this research.

1. Introduction

When software is designed, assumptions are made about the environment in which it will operate. Often, these assumptions are not explicitly recorded, but are "built into the system" [7]. Such assumptions are unlimited in number, and can fail at any time, causing the software to fail to fulfil its purpose.

In a world increasingly dependent on software, the consequences of assumption failure can be very serious. Lehman [7, 8] and others have proposed that assumptions should be monitored throughout the useful life of a piece of software, with action taken to ease the consequences of their failure.

Most research into assumption management has focused on formal, systematic methods. This contrasts with recent trends in software development, which have shown a rise in popularity of more lightweight, 'agile' methodologies [1].

One might expect assumption management to fit naturally within agile development: AM is an essentially simple approach, concerned with responding quickly to change, so it should suit the short release cycles of agile development. Assumptions can be documented briefly in plain English [10], keeping documentation to a minimum and facilitating good communication between developer and client.

This research introduces a simple form of assumption management to a small, agile software development team. It attempts to measure whether AM improves the team's development process, and may therefore be of use to other developers.

2. Related work

Assumption management can be traced back to the late 1980s, when US military researchers devised Assumption-Based Planning [3]. Boehm advocated a similar approach for software development, suggesting that assumption analysis be used to help identify risks at each stage of a project [2].

Lehman first observed that software must evolve to remain useful in a changing world. He proposed that the assumptions on which a piece of software depends should be made explicit and monitored for change, with action taken when they fail [7, 8].

A number of researchers have sought to define properties and categories of assumptions: among these are vulnerability (likelihood of failure) and importance (negative consequences of failure) [3], explicitness [7] and the categorizations of Lago and van Vliet [6] and Lewis et al. [10]. As shown in the next section, we have adopted all these properties in our work.

* The work described herein was carried out by the first author for his part-time MSc project, supervised by the second author.

The military AM process of Dewar et al. consists of five basic tasks [3]; the software development variation of Lewis et al. involves just three – identifying assumptions, monitoring them, and acting to lessen the impact of their failure [10]. Roeller devised a “recovery” technique to retrieve assumptions from past documentation [13]. A simplified and partial version of Roeller’s technique was adopted for our work. The first author looked through documentation and was prepared to interview people, although the latter turned out to not be necessary.

Most research into assumption management has focused on more formal, systematic approaches to the subject. Lehman and Ramil suggest that developers’ “(long-term) goal should be to express specifications formally” [9], while Miranskyy [11] and Lago and van Vliet [6] offer formal models for documenting how assumptions relate to requirements and features respectively. Some have presented assumptions as structured, machine-readable data [4, 14].

Others have opted for a less formal approach to assumption management: Lewis et al. design their “non-disruptive” method for developers who typically dislike maintaining program documentation [10], while Page et al. offer agile developers a “lightweight” method for managing security assumptions [12].

However, we are not aware of previous research seeking to combine assumption management and agile development in an industrial setting. Page et al. do not conduct a case study [12] while Lewis et al. present only brief qualitative results for their research [10].

3. Research Method

A lightweight assumption management process was devised. Over a period of three months (April – June 2008), the process was implemented in the software development team managed by the first author – a small, agile team carrying out in-house development for a large engineering consultancy. Data was collected in a simple Microsoft Access database.

Drawing on the existing literature, four key AM tasks were identified:

- recording new assumptions [10]
- monitoring assumptions on a regular basis, i.e. checking for failure, but also checking for the increased likelihood of an assumption failing, and taking action to lessen the negative consequences of assumption failure [3,10]
- searching for assumptions [10]
- recovering past assumptions (and assumption failures) by looking through documentation, and conducting interviews where necessary [13]

These tasks were performed on a weekly basis over the three month period. Recovery was used to identify new, changing and failing assumptions; these were all recorded in the database.

Assumption failures were also recovered for the preceding three months (January – March 2008), allowing a comparison between assumption failure data before and during the AM implementation.

Two key indicators were devised to try to gauge whether managing assumptions had improved the software development process (Table 1).

Table 1: Key indicators

	Key indicator	Explanation
1	no. of failures of high-impact assumptions	AM aims to lessen the impact of assumptions before they fail – so this measure should decrease as more assumptions are ‘caught in time’.
2	no. of failures of previously unidentified assumptions	AM attempts to identify assumptions before they fail – so a decrease in failing assumptions that had not been previously identified would be a sign of AM’s success.

Data collection involved making subjective judgements about a number of assumption properties. In our data model, each assumption has a description and a category. We used the categorizations of Lago and van Vliet [6]: organizational, managerial and technical. Furthermore, each assumption goes through one or more states, each state being characterised by the following properties:

- Description – the new state of the assumption, described in natural language.
- Stability – rated against a list of six options: top, high, medium, low, very low and none. Low stability would mean an assumption was likely to fail, while a ‘none’ rating meant that the assumption had failed. Stability is hence the inverse of vulnerability [3], with an added value to mark the actual failure of the assumption.
- Impact – the negative consequences for the organization of the assumption’s failure, rated using the same list of options as stability. If the stability is ‘none’, i.e. the assumption *has* failed, then the impact is the actual one, otherwise it is the potential impact of failure.
- Source – the cause or symptom of a change in assumption state or, for the assumption’s initial state, the source of the assumption. Sources were

selected from the following possibilities: bug fix, change request, design decision, management decision, specification.

- Explicitness – a Boolean attribute stating whether the assumption is explicitly stated, e.g. in some specification or management document.
- Action – a description of the action to be taken in response to the change of state.
- Task ID – a numeric field with the ID of a relevant record in the project task database.
- Code Revision ID – a numeric field containing the ID of a relevant code commitment in the source code repository.

To sum up: the assumption state data entity captures the (usually external) cause for an assumption to change state, the new stability value, the action to be taken, and the new impact value due to the action.

The task and revision ID fields enable traceability between assumptions, tasks and code. Code commitment records contain details of all source code changes, and also contain references to project task records. Hence, it was possible to navigate from an assumption change record to a related task record, on to a list of source code changes, and then drill down into individual code changes.

The six options used to rate stability and impact were treated as an interval scale – a ‘top’ rating would be worth 5, ‘high’ would equal 4, down to ‘none’ equalling zero. This allowed the mean average of the impact and stability of a group of assumptions to be calculated (as used in Table 5).

Subjective estimates made by the first author were later verified by two of his colleagues – a developer within the team, and the IT department manager. Both were given copies of the database stripped of data in the category field of assumption records, and the impact, stability and source field of assumption state records. The data they entered in these fields was compared with that entered by the author.

2.1 Examples

Two examples of assumptions recorded during the research are presented at the end of the paper.

Example 1 (Table 6) shows how an implicit technical assumption suddenly failed due to an application being updated. The first assumption state record is entered retrospectively on assumption failure – it estimates the state of the assumption at the time it was first made. Its ‘high’ stability rating is effectively saying: ‘at the time the assumption was first made, it would have been considered highly stable’.

In the second (failed) assumption state, the impact rating refers to the actual impact of assumption failure, after action was taken to deliver the images another way. For the first (live) assumption state, no action was taken; here, the impact rating indicates the potential impact of the assumption’s failure.

Example 2 (Table 7) shows how a decision was made to change the security subsystem used by an application. The second assumption state record shows that the assumption was not judged to have failed – i.e. stability was ‘low’ rather than ‘none’. Arguably, this assumption should also have been marked as failed; it survived because the action (to swap subsystems) had not been carried out, and in fact never has been – the assumption lives to this day. On hindsight, the data model should have included a Boolean field to record whether the action had been carried out. Assumption state records were entered when a change of state was detected and an action was decided upon, assuming it would always be executed – in a handful of cases this turned out to be not true.

These examples are similar in that they both involve parts of a system being replaced with minimal negative impact. In Example 1 the functionality was only used in one place in the application, and was therefore not difficult to replace. In the second example, the subsystem cut across the whole application, but as it was originally implemented using the “facade” design pattern [5] it would have been easy to substitute.

4. Results

A simple breakdown of the data recorded (Figure 1) shows that during the three months in which assumption management was performed:

- 14 previously unidentified assumptions, i.e. assumptions made previously but only detected during the AM period, failed;
- 8 previously unidentified assumptions changed without failing, but 2 of them then went on to fail before the end of the period and hence count as failed for the quantitative analysis described later;
- 11 new assumptions, i.e. made for the first time during the AM period, were recorded, one of which changed by the end of the period.

At the end of this period, 17 assumptions remained live. Further 17 failures were recovered from the three months prior to the assumption management trial. Overall, 50 assumptions were recorded.

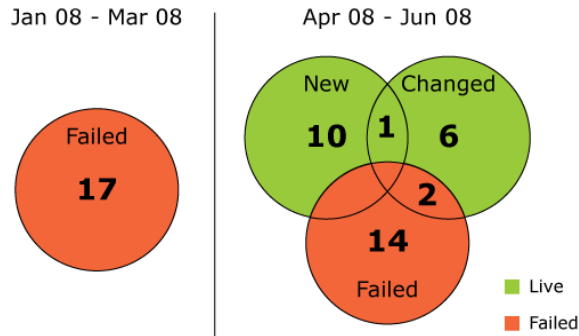


Figure 1: Breakdown of assumption data

We made a month-by-month analysis of when assumptions failed, and which was their impact and explicitness. Only one of the proposed key indicators of successful assumption management (Table 1) was evident in the data collected, namely indicator 2 – a decrease in failures of previously unidentified assumptions.

Action taken on assumption failure was shown to reduce the impact of failure. However, action taken when assumptions changed state (but did not fail) did not reduce impact. This may be due to differences in perceiving the potential impact of a live assumption and the actual impact of a failed assumption.

Among the 33 failed assumptions identified, a number of frequently occurring combinations of assumption properties were observed.

First, assumptions regarded as more stable tended to have a higher impact when they failed (see Figure 2, where the circle size is proportional to the number of assumptions).

Second, managerial assumptions tended to fail due to change requests, while technical assumptions most often failed because of bug fixes (Table 4).

Third, failing assumptions that stemmed from design or management decisions were considered higher in impact, less likely to fail and were made explicit less often than those originating in specifications (Table 5).

Fourth, there were three particular combinations between the source of a failed assumption (first column of Table 5) and the ‘source’ (i.e. cause or symptom) of their failure: 42% of all failed assumptions originated in specifications and failed via change requests, 80% of the assumptions that originated with a management decision also failed via management decision, and 66% of the assumptions stemming from design decisions failed via bug fixes.

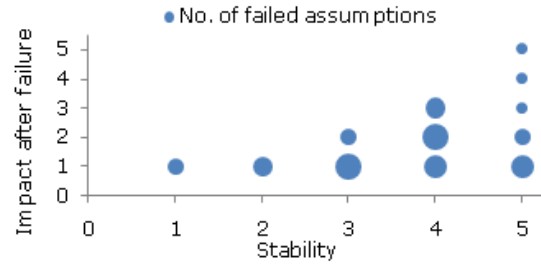


Figure 2: Stability vs impact after failure

As for the validation of the first author’s subjective estimates of assumption properties, they were matched with some success by his department manager and a developer with whom he worked closely on a daily basis. The developer’s ratings for stability, category and source matched the author’s very closely, while their impact ratings showed a moderate positive correlation (Table 2). The department manager’s ratings matched moderately well with those of the author except for stability, for which there was almost no correlation (Tables 3). The developer later said he found the database “very easy” to use, while the department manager said he had found it “quite difficult” to use.

Table 2: Pearson correlation with colleagues’ ratings for stability and impact

	Stability	Impact
Developer	0.95	0.44
Department manager	-0.13	0.44

Table 3: Matches between first author’s and colleagues’ ratings for category and source

	Category	Source
Developer	96%	84%
Department manager	48%	51%

Our explanation for the discrepancy between the colleagues’ responses is that the developer and the first author worked closely together on the team’s software projects, and were constantly in discussion about the projects. The other developer would therefore have had a good understanding of the issues that the assumptions related to, whereas the department manager was not involved at all in the day-to-day work of the team.

The first author did not succeed in integrating assumption management into his daily work – he found it difficult to adjust to thinking in an assumption-oriented way. This meant that, for example, he was unable to spot a new implicit assumption while in the middle of a requirements meeting, or to steer a project client through an assumption-oriented discussion of their change requests. Instead, he resorted to performing assumption recovery once a week. Gradually, however, an ‘assumption-awareness’ did develop, and began to inform his requirements-gathering work.

4.1 Lessons learnt

Reflecting on the difficulties encountered during the process, we offer the following suggestions to practitioners wishing to improve the assumption management approach described in Section 3.

- Enrich the data model so that the cause/effect relationship between actions and assumptions is better captured. For example, measure the effects of actions explicitly and identify whether actions result from AM or if they would be carried out anyway.
- Apply assumption management to the business environment beyond the software team, because of the many organisational and managerial assumptions that arise and their impact on the software solutions.
- Sell the idea of assumption management to senior managers, in order to execute the previous item.
- Ease the introduction of ‘assumption-aware’ thinking, by starting with a checklist of typical assumptions[†], making AM a scheduled team-based activity, defining assumption properties clearly, in particular stability and impact, and publishing policies for identifying assumptions and their changes / failures.

From a research perspective, the lessons learned from this work that might be useful for future work are:

- Due to the implicit nature of assumptions and how they evolve, any study about their management requires a long time frame in order to get sufficient quantitative data.
- The data model is not yet rich enough to capture the full assumption life-cycle and the subtle interdependencies between socio-technical artefacts (from bug fixes to management

decisions) and the assumptions’ properties (e.g. impact).

5. Conclusions

Assumptions are often implicit. They are an important kind of knowledge to be managed during the whole software development life-cycle, because assumption failure impacts organisational, architectural and implementation decisions. This paper contributes to assumption management (AM) as follows.

First, it proposes a rich AM model, based on states with several properties, that includes several previous proposals. Furthermore, the model allows traceability to development tasks and source code.

Second, the proposed method remains simple and lightweight enough to manage and share knowledge about assumptions in a pragmatic way via a form-based database interface.

Third, the paper presents, to our knowledge, the first industrial case study of AM in an agile setting. The weekly monitoring routine, using a simple database that captures assumptions in short English descriptions, was adequate for the agile approach and its short release cycles.

Fourth, the quantitative analysis of the collected assumptions and their properties supports one key indicator for successful AM, and shows that many failed assumptions tend to fit a limited number of ‘profiles’, i.e. combinations of properties. If confirmed by subsequent research, this may help developers to focus their AM efforts more narrowly and efficiently.

The results make us confident that it is possible to develop an AM approach that allows the capture, sharing and reuse of knowledge about assumptions and their evolution in a pragmatic and lightweight way that fits agile development practice. However, the AM experience obtained during the 6 month period, and the comparison with colleagues’ judgements, shows that some improvements are still needed, and we distilled them into lessons that practitioners and researchers may wish to take on board.

6. References

- [1] S. Ambler, “Survey Says... Agile Has Crossed the Chasm”, *Dr. Dobbs’ Journal* 32(8), August 2007.
- [2] B.W. Boehm, “Software Risk Management”, *2nd European Software Eng. Conf.*, Springer-Verlag, 1989, pp. 1-19.
- [3] Dewar, J.A., H. Builder, M. Hix and H. Levin, *Assumption-based Planning: A Planning Tool for Very Uncertain Times*, RAND, Santa Monica, 1993.

[†] We have put together such a list of assumptions, by generalising from the concrete 50 assumptions observed, but omit it due to space constraints.

- [4] S. Fickas and M.S. Feather, "Requirements monitoring in dynamic environments", *2nd Int'l Symposium on Requirements Eng.*, IEEE, 1995, p. 140.
- [5] Gamma, E., R. Helm, R. Johnson, J.M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1994, p. 185.
- [6] P. Lago and H. van Vliet, "Explicit Assumptions Enrich Architectural Models", *27th Int'l Conf. on Software Eng.*, ACM, 2005, pp. 206-214.
- [7] M.M. Lehman, "Uncertainty in computer application and its control through the engineering of software", *Journal of Software Maintenance: Research and Practice*, John Wiley & Sons, Ltd, 1989, pp. 3-27.
- [8] M.M. Lehman, "Uncertainty in computer application is certain-software engineering as a control", *Int'l Conf. on Computer Systems and Software Eng.*, IEEE, 1990, pp. 468-474.
- [9] M.M. Lehman and J.F. Ramil, "Rules and Tools for Software Evolution Planning and Management", *Annals of Software Eng.*, 2001, pp.15-44
- [10] G.A. Lewis, T. Mahatham, and L. Wrage, "Assumptions Management in Software Development", Carnegie Mellon University, 2004.
- [11] A. Miranskyy, N. Madhavji, M. Davison and M. Reesor, "Modelling assumptions and requirements in the context of project risk", Technical Report 645, Dep. of Computer Science, Univ. of Western Ontario, 2005.
- [12] V. Page, Dixon, M. and Choudhury, I., "Security Risk Mitigation for Information Systems", *BT Technology Journal*, 2007, pp. 118-127.
- [13] R. Roeller, P. Lago and H. van Vliet, "Recovering architectural assumptions", *The Journal of Systems and Software*, Elsevier Science Inc., 2006, pp. 552-573.
- [14] A. Tirumala, T. Crenshaw, L. Sha, G. Baliga, S. Kowshik, C. Robinson and W. Witthawaskul, "Prevention of failures due to assumptions made by software components in real-time systems", *ACM SIGBED Review*, ACM, 2005, pp. 36-39.

Table 4: Assumption failures – category of assumption v source of assumption failure

		Source of assumption failure				
		Bug Fix	Change Request	Design Decision	Management Decision	Spec.
Category	Managerial	-	12	-	3	1
	Organizational	2	2	-	2	-
	Technical	8	2	-	1	-

Table 5: Assumption failures - grouped by source of assumption

Source of assumption	No. of Assumption Failures	Average Stability before failure	Percentage of assumptions explicit before failure	Average Impact before failure	Average Impact after failure
Bug Fix	-	-	-	-	-
Change Request	2	1.00	100	3.00	1.00
Design Decision	6	4.17	0	3.00	1.83
Management Decision	5	4.60	20	3.40	3.00
Specification	20	3.60	50	1.85	1.45

Table 6: Example 1 - assumption failure

Assumption	
ID	56
Title	Emailed zip files are an appropriate medium for delivering images requested from the Image Library
Category	Technical
Project	Image Library
Assumption state 1	
Date	-
Description	The image library delivers requested images via emailed zip files. This is an appropriate way to deliver smaller image files to staff - larger files have to be processed manually
Explicit	No
Stability	High
Action	-
Action Authorized by	-
Impact	Low
Task Database ID	-
Code Revision ID	-
Source	Specification
Assumption state 2	
Date	19/06/2008
Description	A Microsoft Office update now blocks Windows from opening zip file email attachments - and most staff do not have an archive program to open the files for them. Also, the new Image Library manager wants to deliver larger image files automatically
Explicit	Yes
Stability	None
Action	Deliver the images another way - by using appropriately secured network folders
Action Authorized by	Image Library manager
Impact	Very Low
Task Database ID	920
Code Revision ID	-
Source	Change Request

Table 7: Example 2 - assumption change

Assumption	
ID	39
Title	JMS permissions are appropriate to use in the MMR system
Category	Managerial
Project	MMR
Assumption state 1	
Date	-
Description	Permissions from the Job Management System (JMS) should be used for implementing security in the Monthly Management Reports (MMR) system
Explicit	Yes
Stability	Top
Action	-
Action Authorized by	-
Impact	Low
Task Database ID	-
Code Revision ID	-
Source	Specification
Assumption state 2	
Date	09/05/2008
Description	These permissions are no longer appropriate - for example, team secretaries need MMR rights that they should not have in JMS
Explicit	Yes
Stability	Low
Action	Amend the MMR system to use the permission settings used by the HR system
Action Authorized by	Business analyst
Impact	Low
Task Database ID	968
Code Revision ID	-
Source	Change Request