Managing assumptions during agile development

Conference Item

How to cite:

For guidance on citations see FAQs

© 2009 IEEE
Version: Accepted Manuscript

Link(s) to article on publisher’s website:
http://dx.doi.org/doi:10.1109/SHARK.2009.5069110

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online’s data policy on reuse of materials please consult the policies page.

oro.open.ac.uk
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” [9]. Such assumptions are unlimited in number, and can fail at any time, causing the software to fail to fulfill its purpose.

In a world increasingly dependent on software, the consequences of assumption failure can be very serious. Lehman [9, 10] 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 [2].

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 [12], 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 [4]. Boehm advocated a similar approach for software development, suggesting that assumption analysis be used to help identify risks at each stage of a project [3].

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 [9, 10].

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) [4], explicitness [9] and the categorizations of Lago and van Vliet [8] and Lewis et al. [12]. As shown in the next section, we have adopted all these properties in our work.

The military AM process of Dewar et al. consists of five basic tasks [4]; the software development variation of Lewis et al. involves just three – identifying assumptions, monitoring them, and acting to lessen the

* This work was carried out by the first author for his part-time MSc project, supervised by the second author.
impact of their failure [12]. Roeller devised a “recovery” technique to retrieve assumptions from past documentation [15]. 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. Haley et al. deal with trust assumptions in the context of security requirements engineering with Problem Frames [7]. Lehman and Ramil suggest that developers’ “(long-term) goal should be to express specifications formally” [11], while Miransky [13] and Lago and van Vliet [8] offer formal models for documenting how assumptions relate to requirements and features respectively. Some have presented assumptions as structured, machine-readable data [5, 16].

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 [12], while Page et al. offer agile developers a “lightweight” method for managing security assumptions [14].

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 [14] while Lewis et al. present only brief qualitative results for their research [12].

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 [12]
- 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 [4, 12]
- searching for assumptions [12]
- recovering past assumptions (and assumption failures) by looking through documentation, and conducting interviews where necessary [15]

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>
<thead>
<tr>
<th>Key indicator</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 no. of failures of high-impact assumptions</td>
<td>AM aims to lessen the impact of assumptions before they fail – so this measure should decrease as more assumptions are ‘caught in time’.</td>
</tr>
<tr>
<td>2 no. of failures of previously unidentified assumptions</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 [8]: 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 [4], 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’
equaling zero. This allowed the mean average of the
impact and stability of a group of assumptions to be
calculated (as used in Table 6).

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 8) 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 8) 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 [6] 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.
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 5).

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 6).

Fourth, there were three particular combinations between the source of a failed assumption (first column of Table 6) 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.

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 2 and 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>
<thead>
<tr>
<th>Developer</th>
<th>Stability</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.95</td>
<td>0.44</td>
</tr>
<tr>
<td>Department manager</td>
<td>-0.13</td>
<td>0.44</td>
</tr>
</tbody>
</table>

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 new implicit assumptions 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; the first author became more adept at spotting assumptions, and took less time to perform the weekly assumption recovery, from over three hours at the beginning of the AM period to under two hours by the end of the three months.

This increased assumption awareness also began to influence the first author’s work more generally – he would find himself instinctively exposing implicit assumptions during project meetings, and searching for them in specification documents.

4.1 Lessons learnt

A number of lessons were learnt during the assumption management process. These are outlined below, along with suggestions for practitioners and researchers wishing to improve the AM approach described in section 3.

The assumption data model was found to have a number of inadequacies:
1. The impact and stability properties did not allow the effects of actions on assumption state to be captured precisely.
2. It was not clear whether actions took place as a direct result of assumption management activity, or whether they would have been carried out anyway.
3. Some actions were not actually carried out – the data model does not make this clear.

One way of solving problem no.1 would be to have two impact and two stability properties per assumption state, indicating the properties’ values before and after action was taken. For items 2 and 3, the addition of a Boolean property to assumption state data would solve the problem in each case.

Over half the assumptions recorded belonged to the organizational or managerial categories (Table 4). Also, around 15% of failed assumptions originated in management decisions – these tended to be implicit and high-impact (i.e. dangerous and hidden). This shows that software developers do not operate in a vacuum – they need to take account of assumptions made outside their own working environment.

### Table 4: Assumptions - by category

<table>
<thead>
<tr>
<th>Category</th>
<th>No. of assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managerial</td>
<td>22</td>
</tr>
<tr>
<td>Organizational</td>
<td>8</td>
</tr>
<tr>
<td>Technical</td>
<td>20</td>
</tr>
</tbody>
</table>

Based on this, a case could be made for extending assumption management into the organization beyond the software development function, or at least ensuring that managers have an input into the AM process.

The difficulties that the first author experienced in developing ‘assumption-awareness’ suggest that this may be a key factor in the successful introduction of assumption management. This process may be eased in the following ways:
- Starting with a checklist of typical assumptions.†
- Making AM a scheduled, team-based activity.
- Defining assumption properties clearly, in particular stability and impact.
- Publishing policies for identifying assumptions and their changes / failures.

From a research perspective, future work may seek to evaluate AM over a longer time frame in order to get sufficient quantitative data; this point is highlighted by the low number of assumptions recorded (50) and the fact that only one of the 11 new assumptions identified had changed by the end of the three months.

Also, researchers may wish to implement the data model improvements suggested above; this may help them to capture the subtle interdependencies between socio-technical artefacts (from bug fixes to management decisions) and assumption properties (e.g. impact).

As for the first author and his development team, they have not continued to practice assumption management. One reason is the relative lack of clear, conclusive evidence of the benefits of AM. This makes it difficult to sell AM to a team that already enjoys a successful development process, and that employs an agile, “just barely good enough” approach to documentation – for example, development artefacts such as models and specifications are not updated as a matter of course, only when there’s a pressing need to do so [1].

† We have put together such a list of assumptions, by generalising from the concrete 50 assumptions observed, and present a subset in Table 7.
The first author does intend to reintroduce his team to assumption management at some point in the future, using an improved data model that will allow the effects of AM to be better measured. In the meantime, he encourages an ‘assumption awareness’ to permeate all aspects of the team’s work.

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

Table 5: Assumption failures – category of assumption v source of assumption failure

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Managerial</td>
<td></td>
<td>-</td>
<td>12</td>
<td>-</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Organizational</td>
<td></td>
<td>2</td>
<td>2</td>
<td>-</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>Technical</td>
<td></td>
<td>8</td>
<td>2</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6: Assumption failures - grouped by source of assumption

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

Table 7: Generalized assumptions

<table>
<thead>
<tr>
<th>Category</th>
<th>Generalized assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managerial</td>
<td>A given use case will never occur - and should therefore not be built into the system.</td>
</tr>
<tr>
<td></td>
<td>A task owned by a certain person will always be performed by that person</td>
</tr>
<tr>
<td></td>
<td>All instances of a given piece of information fit a given format</td>
</tr>
<tr>
<td>Organizational</td>
<td>A project / system will always be owned by a certain person</td>
</tr>
<tr>
<td></td>
<td>A system is used in the same way throughout an organization</td>
</tr>
<tr>
<td></td>
<td>People will never leave the organization</td>
</tr>
<tr>
<td>Technical</td>
<td>A certain technology is appropriate for / accessible to all users</td>
</tr>
<tr>
<td></td>
<td>Two systems do not affect each other in any way</td>
</tr>
<tr>
<td></td>
<td>A technology will not suddenly become obsolete</td>
</tr>
</tbody>
</table>

Table 8: Examples of assumption data
<table>
<thead>
<tr>
<th>Example 1</th>
<th>Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assumption</strong></td>
<td><strong>Assumption</strong></td>
</tr>
<tr>
<td>ID</td>
<td>56</td>
</tr>
<tr>
<td>Title</td>
<td>Emailed zip files are an appropriate medium for delivering images requested from the Image Library</td>
</tr>
<tr>
<td>Category</td>
<td>Technical</td>
</tr>
<tr>
<td>Project</td>
<td>Image Library</td>
</tr>
<tr>
<td><strong>Assumption state 1</strong></td>
<td><strong>Assumption state 1</strong></td>
</tr>
<tr>
<td>Date</td>
<td>-</td>
</tr>
<tr>
<td>Description</td>
<td>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</td>
</tr>
<tr>
<td>Explicit</td>
<td>No</td>
</tr>
<tr>
<td>Stability</td>
<td>High</td>
</tr>
<tr>
<td>Action</td>
<td>-</td>
</tr>
<tr>
<td>Action Authorized by</td>
<td>-</td>
</tr>
<tr>
<td>Impact</td>
<td>Low</td>
</tr>
<tr>
<td>Task Database ID</td>
<td>-</td>
</tr>
<tr>
<td>Code Revision ID</td>
<td>-</td>
</tr>
<tr>
<td>Source</td>
<td>Specification</td>
</tr>
<tr>
<td><strong>Assumption state 2</strong></td>
<td><strong>Assumption state 2</strong></td>
</tr>
<tr>
<td>Date</td>
<td>19/06/2008</td>
</tr>
<tr>
<td>Description</td>
<td>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</td>
</tr>
<tr>
<td>Explicit</td>
<td>Yes</td>
</tr>
<tr>
<td>Stability</td>
<td>None</td>
</tr>
<tr>
<td>Action</td>
<td>Deliver the images another way - by using appropriately secured network folders</td>
</tr>
<tr>
<td>Action Authorized by</td>
<td>Image Library manager</td>
</tr>
<tr>
<td>Impact</td>
<td>Very Low</td>
</tr>
<tr>
<td>Task Database ID</td>
<td>920</td>
</tr>
<tr>
<td>Code Revision ID</td>
<td>-</td>
</tr>
<tr>
<td>Source</td>
<td>Change Request</td>
</tr>
</tbody>
</table>