ISSN 1744-1986

Technical Report N° 2008/ 27

Managing assumptions during agile software development

I. Ostacchini

28 June, 2008

Department of Computing
Faculty of Mathematics, Computing and Technology
The Open University

Walton Hall, Milton Keynes, MK7 6AA
United Kingdom

http://computing.open.ac.uk

Managing assumptions during agile software development

A dissertation submitted in partial fulfilment of the requirements for the Open University's Master of Science Degree in Software Development

Ireo Ostacchini

X3846591

7 February 2009

Word Count: 13,580

Preface

I’m grateful to my supervisor, Michel Wermelinger, for his guidance and encouragement throughout this project. I’d also like to thank my employer for granting me permission to perform the case study, and my colleagues for taking part. Finally, I’d like to dedicate this dissertation to my family – thanks for putting up with me!

Contents

List of figures

List of tables

Abstract

Software plays an increasingly critical role in our world, yet the assumptions that underlie software development often go unrecorded. These assumptions can fail at any time, with serious consequences.

This research evaluates a lightweight approach to assumption management (AM), designed to complement the agile software development methods that are gaining in popularity. Key AM tasks were drawn from previous research, and implemented over three months within a small, agile software development team. A simple database was developed for recording and monitoring assumption information.

Thirty-three assumptions were recorded during the three months; a further 17 failed assumptions were recovered from the preceding three months. Two key indicators were proposed for measuring whether AM had been successful. Only one of these indicators was detected in the research results; a longer research timeframe would be required for a more conclusive analysis.

A number of strong correlations were found between properties of assumptions. While the data collected depended to a large degree on the subjective estimates of the author, these judgements were validated with some success by his colleagues.

In some ways, assumption management was found to be a good fit for agile development; however, the AM process was not successfully integrated into

the team's development process, due to a difficulty in adapting to the required 'assumption-aware' way of thinking. Advice is offered to researchers seeking to ease this transition, and to those looking to conduct further studies in assumption management.

Chapter 1 Introduction

1.1 Background to the research

Assumption - "a fact or statement… taken for granted" (*Merriam-Webster, 2007*).

In life, when things don't go to plan, it's often because circumstances change in a way we don't foresee. Whether we realize it or not, our plans depend on assumptions we make about the world. For example, when farmers plant their crops, they assume that enough sunlight and rain will occur to make them grow.

But the world is an ever-changing place, and things that we take for granted can fail. When this happens, plans that had served us well are suddenly no longer useful.

This phenomenon also affects software development. When software is designed, assumptions are made about the environment in which it will operate – for example, a company buying a financial system might assume it will only send invoices in a single currency.

Often, these assumptions are not explicitly recorded in requirements specifications, but are "built into the system" (Lehman, 1989).

And because software operates in the real world, with its complexity and constant change, then there's no limit to the number of things that might go wrong – i.e. the number of assumptions on which a program depends (ibid.).

If these assumptions fail, software may no longer be capable of doing the job required of it; in a world increasingly dependent on software, the consequences of this can be very serious.

Some have advocated that assumptions be made explicit when specifying requirements for a new system – for example, Boehm's software risk management approach (Boehm, 1989) and the Volere requirements template (Robertson and Robertson, 2007).

But assumptions can fail at any time, even after a system has been in use for many years. Lehman (1989, 1990) was the first to point this out, and to recommend that assumptions be monitored on an ongoing basis throughout the useful life of a piece of software, with action taken to ease the consequences of their failure.

1.1.1 Assumption Management

So how do we manage assumptions – what tasks are to be performed, when and by whom? And how can assumption management (AM) best fit into a development process?

Assumption management can be broken down into three basic tasks (Lewis et al., 2004, Dewar et al., 1993):

- Identify and record assumptions
- Monitor them for change
- Act to lessen the consequences of assumption failure (hedging actions) or to affect the probability of failure (shaping actions)

In theory, there is no limit to the number of assumptions that can be identified, so we can't hope to capture every assumption embedded within a software project. In practice, we can try to capture the assumptions that we think may be most important – basing our judgement on their vulnerability (likelihood of failure) and impact (consequences of failure).

Despite the essential simplicity of assumption management, a number of researchers have proposed systematic approaches to the subject, heavy on documentation and procedure – although they have sought to lighten the developer workload by using tool support (Miranskyy et al., 2005). Some have argued for assumptions to be included in formal specifications (Lehman and Ramil, 2001), while others have developed meta-models describing how assumptions can be linked to lower-level artefacts such as requirements (Miranskyy et al., 2005), features (Lago and van Vliet, 2005) and source code (Yang Li et al., 2001).

1.1.2 Assumption Management and agile development

The formal nature of much research into assumption management contrasts with the prevailing trend in software development over the last ten years – a move away from a rigorous, process-oriented mentality towards a more 'agile' approach. As agile development becomes more widespread (Ambler, 2007), developers may be less likely to adopt techniques that require them to spend their time updating large amounts of documentation.

Some researchers have taken this on board, advocating lightweight approaches to assumption management. Lewis et al. (2004) offer a "non-disruptive" AM method, Yang Li et al. (2001) focus on capturing only the most important (i.e. vulnerable / high impact) assumptions, and Page et al. (2007) design a security-oriented AM method for agile developers. However, to my knowledge, researchers have yet to evaluate assumption management in an industrial agile development team.

There are a number of reasons why assumption management may fit naturally with agile development:

- Risk management is a core theme in agile development (Beck 1999); AM could be considered a form of risk management
- AM consists of simple procedures performed regularly - this provides "timely feedback" suitable for the short development cycles of agile development (Lewis et al., 2004)
- Like agile development, AM is designed to cope with a changing environment – i.e. to deal with constantly changing requirements

- Agile development places emphasis on keeping documentation to a minimum (Beck et al., 2001); AM offers a lightweight form of documentation - a typical assumption description may be brief, and written in everyday language (Lewis et al., 2004)
- Using everyday language to describe assumptions would also facilitate close communication between developer and client
- AM may be applied to the software development process itself

1.2 Aims and objectives of the research project

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

In pursuit of this aim, the following objectives have been set:

- Investigate previous research into assumption management
- Drawing on earlier research, design a simple set of AM techniques suitable for an agile team
- Implement AM in a software development team for a limited period of time
- Identify lessons learnt from the case study – present and analyse the data collected during the research, reflect on the AM process, and make suggestions for future research

1.2.1 Contribution to knowledge

The intended audience for this research consists of three groups:

- Software developers – in particular, agile developers who are looking to improve their process, perhaps by adding an element of risk management. The AM approach proposed here may offer them a simple way of doing this without burdening themselves with too much documentation
- Researchers studying software development – hopefully, the research methods and analysis presented here will be seen by researchers as a valid contribution to the (very slim) body of work on agile assumption management, and that it will encourage others to pursue similar investigations
- People at the company I work for – I hope that my work will lead to a more effective software development process for them, and lessen their exposure to the unpleasant surprises that occur when implicit assumptions fail

The research contains a number of artefacts that may be of interest to its intended audience. These include:

- A lightweight assumption management method aimed at agile developers
- A simple lifecycle model for assumptions

- Data structure and user interface designs for a simple assumption management database
- Real-world examples of data collected during the research
- A list of generic assumptions, offered for use as an 'assumption checklist' for developers new to assumption management

A statistical analysis of the quantitative data collected during the research is performed. The aim is to show whether AM had a positive effect on the team's development (I propose and measure key indicators for this), and to identify correlations between the properties of changing / failing assumptions.

I also present my reflections on the AM process - in particular on its suitability for agile development – and a list of suggestions for future researchers seeking to evaluate assumption management.

1.3 Overview of the dissertation

Chapter 2 presents a review of the literature on assumption management; the concept is traced back to its origins and tracked through subsequent literature to the present day.

Chapter 3 outlines the research methods employed, and relates them to the literature. In particular, I discuss the method used to implement

assumption management, and examine the structure of the database used in this research. Three examples of assumption data are discussed.

Chapter 4 presents the quantitative and qualitative results to emerge from the AM evaluation. Data collected during the research is analyzed, key performance indicators are proposed and evaluated, and a list of reflections on the AM process is presented.

Chapter 5 draws conclusions on the successfulness of the research, and offers suggestions for future research into assumption management.

1.4 Research question

The research question follows directly from the project aim stated above, and reads:

> 'How does the introduction of assumption management affect the work of a small, agile software development team?'

Chapter 2 Literature review

2.1 Concepts

The concept of assumption management can be traced back to the late 1980s, when US military researchers devised Assumption-Based Planning. Instead of trying to identify the most-likely future scenario, army planners would focus on "the assumptions on which current operations and plans are based" (Dewar et al., 1993 p. xi). Those assumptions would be identified and ranked, with priority given to the most vulnerable assumptions that had the most severe consequences of failure. Contingency plans would be drawn up, and the assumptions monitored for signs of change.

At the same time, Boehm was advocating a similar approach for software development. He devised the Spiral Model, an iterative development process in which risks were to be identified, monitored and acted upon at each stage of a project (Boehm, 1988). Boehm proposed assumption analysis as a technique for identifying those risks (Boehm, 1989).

However, as Lehman first observed, software can run into trouble even after its initial development has finished (Lehman, 1969). The software itself may be unchanging, but it must operate in an uncertain, ever-changing world (Lehman 1989, 1990). Therefore, if it is to retain its quality and usefulness, software must undergo continuous adaptation – it must evolve (Lehman, 1969, 1974).

Like Boehm, Lehman saw the software development process as one that involves making assumptions about the real world – these assumptions are often not explicitly recorded, but they are "built into the system" (Lehman, 1989), and need to be valid every time the program is executed.

But the world is changeable, and assumptions fail. For Lehman, as for Dewar and Boehm, assumptions must be made explicit and monitored for change, with action taken when they fail (Lehman, 1989).

2.2 Practical investigations

Once the concept of assumptions was established, researchers began to focus on more practical investigations. A much-referenced introductory document for an IEEE conference panel raises the following questions (Johnson et al., 1993):

- Who needs to keep track of assumptions, and why?
- How do we obtain assumptions?
- How do we record, manage and use assumption information?
- Does this make developers' jobs easier or not?
- What tools can be used?
- How will development processes be affected?

Other researchers have followed this practical approach, investigating the properties of assumptions and how they typically differ from each other - that is, how they can be categorized.

Properties identified include:

- A textual description and category are the two properties used by Lewis et al. (2004) in their simple assumption management implementation
- Format - in some research, assumptions are presented as free text (Lewis et al., 2004, Lago et al., 2004). Other researchers see them as structured, machine-readable data (Fickas and Feather, 1995, Tirumala et al., 2005)
- Explicitness / implicitness (Lehman 1989). An assumption may or may not be explicitly stated in program code or documentation
- Initial validity / invalidity (Miranskyy et al., 2005). An assumption may be invalid from the start, or may become invalid over time
- Vulnerability – the probability that the assumption will fail (Dewar et al., 1993)
- Impact (Lehman, 2005) – the effect of assumption failure (labelled importance by Dewar et al. (1993))

Lewis et al. (2004) identify five categories of assumption, relating to control, environment, data, usage, and convention. Lago and van Vliet (2005) recognize that assumptions exist at different levels of abstraction; at the highest level are organizational assumptions that "reflect the company as a whole", followed by managerial assumptions that reflect "decisions

taken to achieve business objectives". At the lowest level are technical assumptions covering areas such as "programming languages, database systems, operating systems".

2.3 Assumption management

Most assumption management methods offer variations on a basic risk management process: identify risks, monitor them, and take action to reduce the likelihood or impact of those risks materializing.

For example, the 'Assumption-Based Planning' method of Dewar et al. (1993) sets out a five-stage procedure:

- Identify important assumptions
- Identify assumption vulnerabilities
- Define signposts - signs that an assumption is changing in importance / vulnerability
- Define shaping actions - actions to change the likelihood of an assumption failing
- Define hedging actions - actions to lessen the negative impact of assumption failure

While the approach of Dewar et al. is based in the plan-oriented culture of the US army, Lewis et al. (2004) define their AM process with software developers in mind. They identify three key elements of the process:

- "an approach to record assumptions in 'real-time' in a manner that is not disruptive for the developer
- a mechanism to validate the assumptions by the people that possess the knowledge to declare each assumption valid or invalid
- a tool to search for assumptions by other developers and people involved in the development effort"

Lewis et al. aim their method at developers who typically dislike maintaining program documentation. This approach is in tune with the increasing adoption of agile software development practices (Ambler, 2007) which place "working software over comprehensive documentation" (Beck et al., 2001) and favour a "just good enough" approach to documentation (Ambler, 2004). Page et al (2007) offer a "lightweight" method for (security) assumption management that can easily be adopted by agile developers, while Yang Li et al. (2001) also hint at this trend by capturing only the most important (i.e. vulnerable / high impact) assumptions.

However, much of the research into assumption management has focused on more formal, systematic approaches to the subject. Lehman and Ramil (2001) suggest that developers' "(long-term) goal should be to express specifications formally", with assumptions being made explicit as part of that process – "they must, as far as possible, be detected, captured, validated, formally approved and added to the specification". Where it is not possible to define a whole system formally, Lehman and Ramil suggest that developers should compose programs using small subsystems – termed "bricks" - that can be formally specified and validated.

2.4 Modelling assumptions

This tendency towards formal assumption management is evidenced by the number of researchers seeking to model assumptions, particularly in relation to other development artefacts such as requirements, features and source code.

Miranskyy et al. (2005) develop a mathematical model of the relationship between assumptions and requirements, in a bid to predict the risk of system failure over time. They claim that this use of assumptions "cuts through the barrier solidly experienced by practitioners that documenting assumptions does not have a short-term payback".

Lago and van Vliet (2005) add assumptions to architectural models of a software product family, modelling the dependencies between assumptions and features; they find that an assumption may not only "cross-cut" several features, but also have dependencies among several structural elements (such as components and interfaces). From this model, they derive a meta-model for allowing assumptions to be documented in relation to features and program structure.

Lago and van Vliet claim that this formal, explicit modelling of assumptions allows architectural decisions to be more fully documented; it provides a traceability that allows proposed changes to a system – that is, to its features - to be validated against the assumptions linked to those features.

Others, while making no specific mention of assumptions, take a similar tack in seeking to model the relationship between "evolvable domain knowledge" and source code (Yang Li et al., 2001). Yang Li et al. suggest that this can be done automatically, using a pre-defined set of rules to match data from a "domain knowledge base" to source code elements. Heisel and Von Schwichow (2004) suggest a more labour-intensive approach, with developers constructing (and maintaining) a model to link "pure problem domain concepts" to code via several intermediate layers of abstraction.

2.5 Recovering assumptions

Roeller et al. (2006) focus on recovering assumptions from past development work. In contrast to the automated approach to assumption recovery of Yang Li et al. (2001), they argue that most assumptions are not explicitly stated, and so "we cannot expect a tool to recover tacit knowledge". Instead, they identify assumptions by first searching through a variety of information sources - including project documentation, code, code metrics and version control notes – for "suspicious effects" - for example, "a component that scores badly on a given set of quality metrics". These suspicious effects are then used as a basis for interviews with developers.

Roeller et al. found that without a deep knowledge of the system – and the problem domain – it was very difficult to identify the most important assumptions. They also found it hard to identify who the experts were – i.e. who made the design decisions in the first place.

2.6 Assumptions and requirements

Many researchers explore the relationship between assumptions and requirements. Lehman and Ramil (2001) assert that assumptions are typically not part of a requirements specification, but are part of the resulting system.

Seacord (2003) also claims that requirements leave much unsaid about the reality of a system: "requirements are only a top-level statement of need, and the road between requirements and code is paved with assumptions".

Some authors recommend searching for assumptions while capturing requirements for a system (Alexander, 2006; Robertson and Robertson, 2007). Seacord (2003) notes that "an argument can be made that nothing in a software system should be assumed", and that any assumptions should be explicitly recorded as requirements. Robertson and Robertson (2007) share this view; their Volere requirements template states that "the assumptions are intended to be transient. That is, they should all be cleared by the time the specification is released — the assumption should have become either a requirement or a constraint".

However, Lehman and Ramil (2001) note that, in reality, it is not possible to eliminate all assumptions in this way. As time passes, "situations requiring the de facto introduction of new assumptions cannot be avoided"; inevitably, "some assumptions will slip through the net". But Lehman and Ramil argue that assumptions must be recorded and monitored "as far as

possible". For them, assumptions can never be managed completely – but that should not stop us trying to do so.

According to van Lamsweerde (2001), there is an essential difference between assumptions and requirements with regard to responsibility; requirements are under the control of the system (or part of it) while assumptions are not. "Unlike requirements, assumptions cannot be enforced by the software-to-be; they will hopefully be satisfied thanks to organizational norms and regulations, physical laws, etc".

As noted above, Miranskyy et al. (2005) attempt to model the relationship between assumptions and requirements. Fickas and Feather (1995) also attempt to make this connection; they present a tool for monitoring requirements at run-time, based on a mapping of requirements to assumptions and the "insertion of code into a running system" to monitor for the failure of requirements (and by extension assumptions). Ultimately, their goal is for a system to 'know' when it needs to evolve, and to take appropriate action automatically.

2.7 Rationale management

Rationale management is a concept closely related to assumption management, albeit more broadly defined: "rationale methods aim at capturing, representing, and maintaining records about why developers have made the decisions they have" (Dutoit and Paech, 2001).

Dutoit and Paech (ibid.) make a number of points about rationale that can be equally applied to assumptions:

- Rationale often remains implicit
- Recording rationale can help developers retain and share knowledge related to a system – this is useful when evaluating changes during the maintenance phase of a system's lifecycle
- Rationale management involves an initial investment, with the payback coming later
- "The systematic integration of rationale into software engineering processes and tools has yet to happen" (Dutoit and Paech, 2001)

Ramesh and Jarke (2001) define a set of reference models to enable requirements traceability; at the heart of their "Requirements Management Model" they place these two key relationships: requirements are based on rationale, and rationale is based on assumptions. The implication for assumption management is that if one wants to analyse the dependencies between requirements and assumptions, rationale must not be left out of the equation.

2.8 Architectural mismatch

Another line of research focuses on the assumptions that reusable components make about the components they interact with, and the environments they are used in. Garlan et al. (1994) describe this phenomenon as "architectural mismatch", and say that such assumptions

almost always remain implicit because "software engineers have neither the proper vocabulary nor the structure to help them express these assumptions".

Tirumala et al. (2005) claim that architectural mismatch is "a prime source of failures" in large-scale real-time systems. They give the example of the Ariane V rocket disaster, caused by the reuse of a component from a previous mission. "An implicit assumption made by the component that a variable would never overflow 16 bits was violated in Ariane 5. This assumption, though documented, was not validated in the new system."

Garlan et al. (1994) suggest that the answer may come from research "work on architecture description languages and formal underpinnings for software architecture". Uchitel and Yankelevich (2000) explore this idea, and investigate how these assumptions might be modelled so as to avoid mismatch. They find that existing architecture description languages are not expressive enough – extensions would be required to accurately model assumptions. But when they consider the wide range of assumptions that a component might have to make, they conclude that "the spectrum of assumptions is too big to manage in one formalism", and limit themselves to modelling a small subset of possible types of assumption.

Tirumala et al. (2005) also explore the possibility of embedding assumptions in the formal description of components. They highlight the inadequacy of component interfaces for describing assumptions, and devise "property interfaces" through which "each component publishes its

assumptions and its guarantees. The assumptions of each component are verified against the guarantees provided by the other component".

They develop a tool that performs this validation, and that combines components into validated subsystems with their own software and property interfaces – building the "bricks" proposed by Lehman and Ramil (2001).

2.9 Variability

Where developers foresee that a system might need to be changed in the future, they may build in "variability points" (Roeller et al., 2006) – i.e. they may structure the code in a way that facilitates such change. According to Lago and van Vliet (2005), developers need to make "assumptions about what will change, and what will not". If enough flexibility is not built into a system, "we may get into trouble if something changes that we had not foreseen". This practice can be likened to the "hedging actions" of Dewar et al. (1993) – actions taken in advance to minimize the impact of assumption failure.

2.10 Trust assumptions

Viega et al. (2001) approach the question of assumption management from the point of view of security: "A trust relationship is a relationship involving multiple entities (such as companies, people, or software components)". These entities can make dangerous security-related "trust assumptions"

about each other – often due to "incomplete requirements and miscommunication between development groups".

Trust assumptions seem similar in many ways to the assumptions described above. And researchers offer similar solutions for managing them: Haley et al. (2003) emphasize the need to define "security requirements" explicitly, and suggest "capturing trust assumptions in some semi-formal way". Page et al. (2007) offer a lightweight method for doing just this; they identify trust assumptions implicit within requirements use cases, and model "obstacle cases" – use cases that model security threats associated with assumption failure. They then draw up "mitigation cases" – similar to Dewar's hedging actions (Dewar et al., 1993).

2.11 Agile development

To my knowledge, no previous research has sought to combine assumption management and agile development in an industrial setting. Page et al. (2007) aim their security-oriented AM method at agile teams, but do not conduct a case study.

The AM implementation of Lewis et al. (2004) is limited to the "coding phase" of a project – they do not state the development methodology used, but one would expect it not to be agile if a coding phase were involved.

Lewis et al. (2004) present only brief qualitative results for their research. The developers they questioned said that:

- AM provided then with "an easy way to get early feedback" – errors were caught earlier than they might otherwise have been
- They "did not feel they were doing a lot of extra work by documenting these assumptions - 'It's just like writing comments.'"

Lewis et al. go on to suggest agile development as a potentially suitable environment for assumption management, because AM offers the "timely feedback" that is so useful to short-iteration development.

Risk management (of which AM might be thought of as a subsidiary discipline, with its similar 'identify-monitor-act' methodology) has close links with the agile approach: "Most agile development methods claim to be risk driven" (Smith and Pichler, 2005). Examples of this are Scrum and Extreme Programming (XP) – indeed, Beck's original book on XP begins: "The basic problem of software development is risk" (Beck 1999, p.3).

But the extent to which agile processes manage risk is open to question: Smith and Pichler (2005) find that "many books available on agile methods... have remarkably little to say" about managing risk. Nyfjord and Kajko-Mattsson (2007) begin their comparison of agile and risk management process models thus: "On the surface, agile and risk management process models seem to constitute two contrasting approaches. Risk management follows a heavyweight approach whereas agile process models oppose it."

However, Nyfjord and Kajko-Mattsson find that, despite their differences, agile and risk-based approaches do in fact have much in common, and it may be possible to combine the two.

2.12 Summary

The concept of assumption management originates in the research of Dewar et al. for the US military during the late 1980s. Boehm quickly applied the concept to software development as part of his Spiral Model.

Lehman showed that assumptions are unlimited in number, often remain implicit, and can fail at any time, rendering a previously useful piece of software unfit for purpose. He stressed the importance of monitoring for assumption failures throughout a program's useful life.

Researchers began to study assumptions in increasing numbers, defining their properties and categories, and offering methods for recovering and managing them. Most favoured systematic approaches, and developed formal models for documenting assumptions. Others such as Lewis et al. (2004) opted for lightweight approaches to assumption management.

Other related lines of research emerged:

- The relationship between assumptions and other software development artefacts such as requirements and rationale

- Architectural mismatch – the failure of assumptions that software components make about each other
- Variability – building flexibility into software in order to cope with a changing environment
- Trust assumptions – tackling the issue of assumption management from a security viewpoint

But hardly any research has sought to combine assumption management with agile development – and I have found no industrial case study in the literature.

Chapter 3 Research Methods

3.1 The development team

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 that I manage – a small, agile team working within a large engineering consultancy. The team develops web-based information systems for internal use by the company; recent projects include invoice management, HR and financial reporting systems.

The team bases its development process on agile development techniques:

- Software is developed in iterations lasting four weeks or less.
- Each iteration results in delivery of working software
- Change requests are welcomed at all times
- The team is in daily face-to-face contact with project clients
- Refactoring is used to improve software design on a continuous basis
- Documentation is kept light and yet rigorous: all specifications and change requests are logged in a project task database, with confirmation sent to the project client via email.

3.2 The Assumption Management method

Four key assumption management tasks were identified from previous research:

- **Recording** new assumptions (Lewis et al., 2004)
- **Monitoring** assumptions on a regular basis (Dewar et al., 1993; Lewis et al. 2004)
- **Searching** for assumptions (Lewis et al., 2004)
- **Recovery** - identifying past assumptions by looking through documentation and conducting interviews where necessary for clarification (Roeller et al. 2006)

3.2.1 Recording new assumptions

New assumptions were identified via the recovery technique described in section 3.2.4. The assumption information was recorded in a simple, custom-built Microsoft Access database.

3.2.2 Monitoring assumptions

The process of monitoring assumptions was broken down into the following sequence of events:

1. An assumption changed state – this was identified via assumption recovery, and recorded in the database

2. Action was taken to reduce the negative consequences of assumption failure, or the likelihood of failure - echoing the "hedging" and "shaping" actions of Dewar et al. (1993). These actions were also recorded in the database.

Assumption monitoring was carried out once a week, in conjunction with assumption recovery. Before starting to examine the documentation generated that week, I would read through the list of live assumptions. I would then have that list to hand as I scanned through the documentation.

3.2.3 Searching for assumptions

Searching for assumptions was easy, as they were all in one place (the database) - rather than scattered throughout source code, as Lewis et al. (2004) proposed.

3.2.4 Assumption Recovery

Once a week, I would attempt to recover assumptions from documentation using a method based on that of Roeller et al. (2006). Each week, I would search for new, changing and failing assumptions in the various types of documentation generated or changed that week – these included:

- Changes to records in the team's task database
- Project documents such as specifications and design diagrams
- Emails sent or received

- Comments associated with commitments to the team's source code repository (from there I could drill down to view actual code changes)

My recovery technique differed from that of Roeller et al. in a number of ways:

- Roeller et al. were only concerned with architectural assumptions – I was looking for assumptions that might impact my team's software development in any way
- Roeller et al. employed a two-stage method, first identifying a list of "suspicious effects" from documentation, then using guided interviews to define assumptions. I tried to identify assumptions directly from documentation
- The second stage of Roeller et al.'s method (guided interviews with project stakeholders) was generally not required, as I was familiar with the issues at hand - I had managed the company's applications and overseen the team's development projects for several years. Occasionally, however, I would turn to a colleague to clarify an issue
- I did not use free interviews or source code metrics as sources of assumption information – the interviews for the reasons described above, the metrics because they seemed a 'heavy' way of doing things. The tools I had available presented complex snapshots of entire codebases, rather than capturing the effects of changes occurring within a given timeframe

3.3 Two periods of data collection

Assumption recovery was also used to identify assumption failures during the three months prior to the period of assumption management (i.e. January - March 2008).

This allowed a comparison to be made between assumption failure data before and during the AM implementation - thus providing a way to evaluate the effectiveness of managing assumptions.

To justify the comparison of data from the two periods, most aspects of the team's work would have to be similar, with the only significant variable being the introduction of the AM method.

The two periods compare as follows:

	Jan 08 – Mar 08	**Apr 08 – Jun 08**
Development Process	The team's usual development process	Same
Development Projects	Mainly small (<1 week) items of work – change requests, reports, bug fixes	Similar – except for some initial work on a larger (2 month) project
Team resources	Two people	Same
People doing AM work	One person (myself)	Same
Assumption recovery method	As outlined above	Same
Information recovered	Assumption failures and resulting actions	Same, plus new assumptions, changes in existing assumptions and resulting actions
Frequency of recovery	Once only, at the end of March	Once a week

Table 1: Two periods of data collection

Based on the information in Table 1, I would suggest that the two three-month periods are similar enough to be comparable in terms of assumption failure rate.

3.4 The database

Data was collected in a Microsoft Access database. A relational database was chosen rather than a more loosely structured data format such as XML for the sake of convenience; database data can be manipulated, searched and analysed more easily than XML. Access in particular was chosen because of my familiarity with the product and its ease of use.

Another key reason for choosing a centralised database was that I saw no need to distribute assumption data among several locations – e.g. within source code files or requirements documents.

Lewis et al. (2004) opted to embed assumptions in source code as XML comments, and developed a tool to extract them into a database. They assumed that all assumptions relating to software development could be identified by developers (rather than managers) and be placed in one area of source code.

The identification of an "organizational" category of assumption by Lago and van Vliet (2005) caused me to doubt this approach, and in fact this research's data includes a number of organizational assumptions that cannot be located in source code.

3.5 The assumption lifecycle

The design of the database was based on the simple assumption lifecycle model shown in Fig. 1. According to this model an assumption is made, it goes through any number of changes while remaining live, then it may (or may not) fail.

The assumption might go unnoticed through its entire lifetime, or it may become managed (recorded as part of assumption management) at any of the three state transitions shown in the model:

- When the assumption is first made
- When it changes (and remains live)
- When it fails

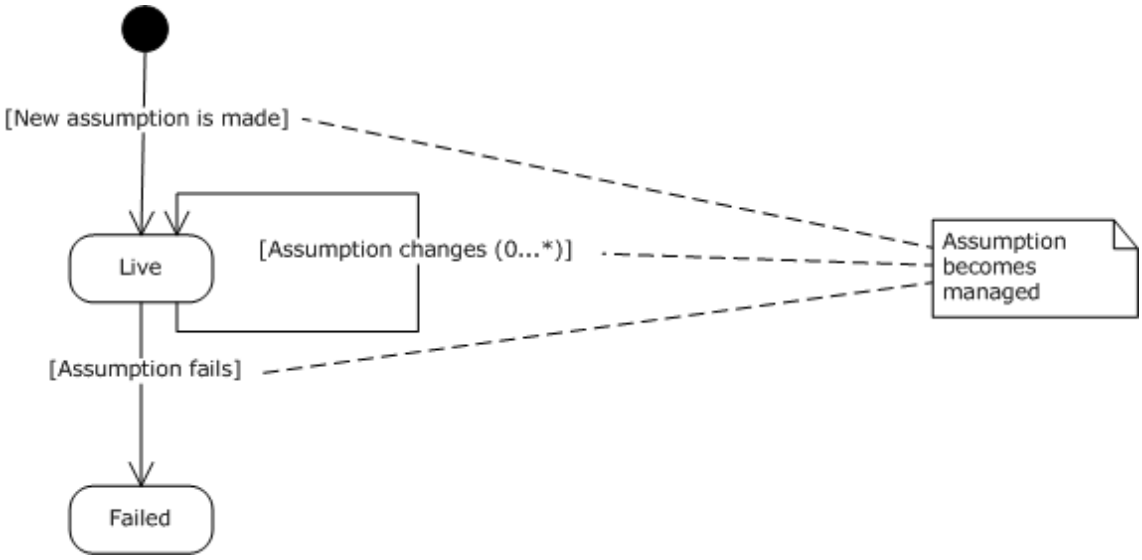

Fig. 1: The assumption lifecycle

Of course, it's possible to argue that a live assumption might become managed even though it hasn't really changed in any way. One could also argue that the mere act of placing the assumption under management causes it to change in some way – this would allow the lifecycle model to remain valid.

I think that to pursue that analysis further here may be to look too far beyond the intended purpose of the model – which is merely to present a realistic enough picture of a real-world process that it may provide a basis for designing a data structure.

Another, perhaps more significant objection to the lifecycle model might be that assumptions might return to a live state after they have failed. The lifecycle model does not allow for this, although common sense might suggest that this does happen – for example, if a management decision were to be reversed.

It was decided not to build this possibility into the lifecycle model, in the name of keeping things simple. However, it is an issue that future researchers might wish to explore.

3.6 Data structure

The data structure shown in Fig. 2 is split into two main classes: the Assumption class contains the properties of an assumption that are not expected to change during its lifetime – title, project and category.

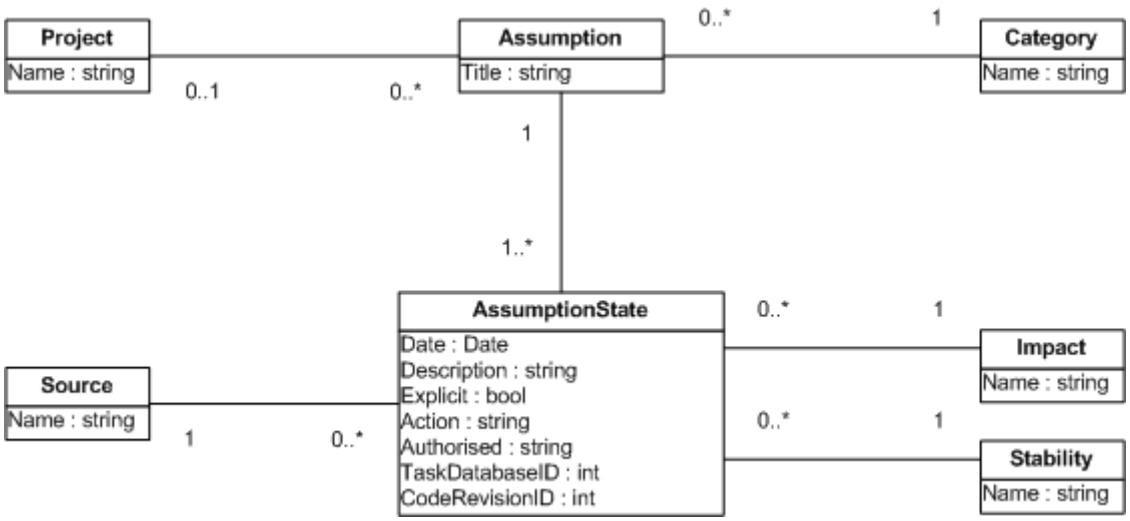

Fig. 2: Data structure

Each assumption would have one or more associated instances of the AssumptionState class. AssumptionState represents the state of an assumption at a given point in time – so each time an assumption changed in any way, a new AssumptionState record would be created in the database.

An AssumptionState record would be created for each of the state transitions defined in the Assumption Lifecycle model – i.e. when an assumption was first made, when it changed whilst remaining live, or when it failed.

It would not be possible to have an Assumption without any associated AssumptionState; whenever an assumption would be identified for the first time, both Assumption and AssumptionState records would be created together. At this point, the assumption's title, category and project properties would also be set.

The data structure was implemented in the database as a main form / sub-form combination, as shown in Fig. 3. Assumption State records were listed by date – an undated record would appear at the top of the list (undated records are explained in section 3.7.3).

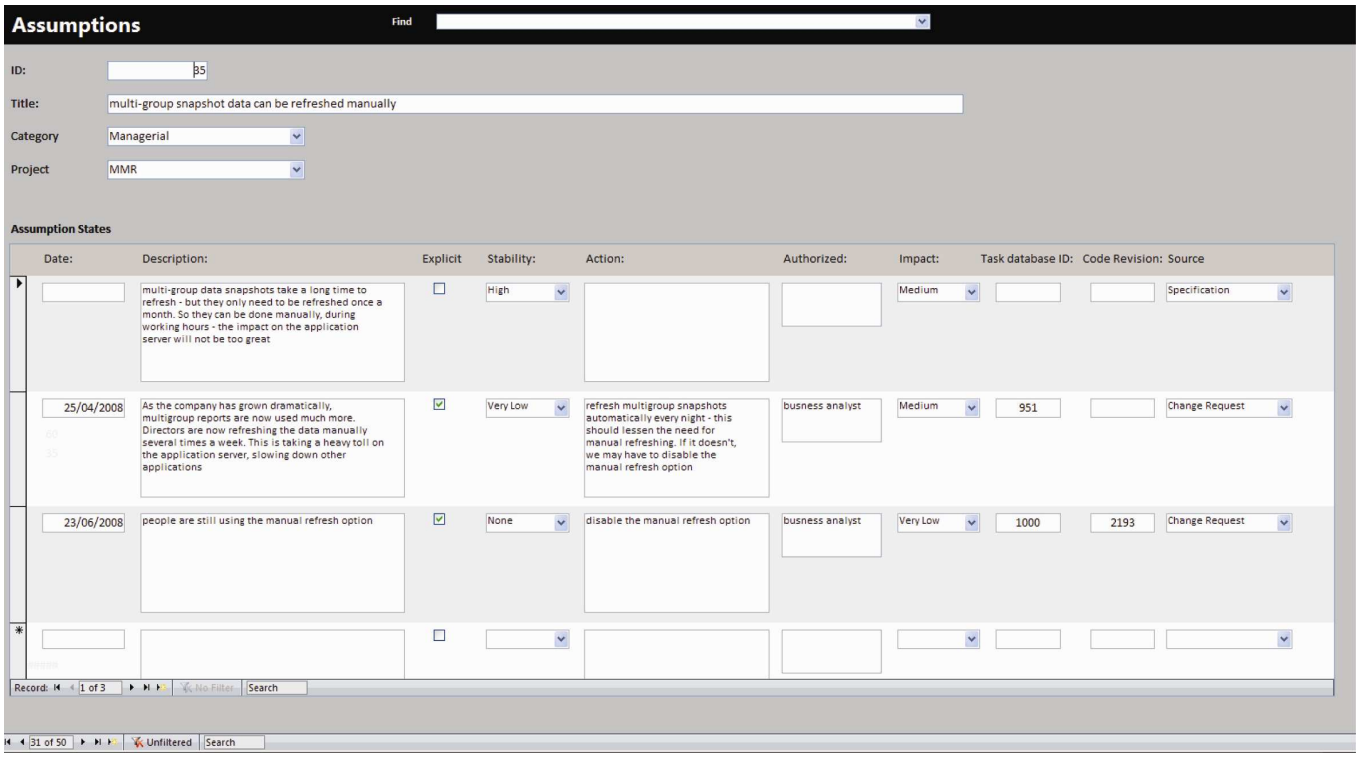

Fig. 3: Database form

3.6.1 Assumption: Title

A short phrase (10 words or so) used to give a very brief description of the assumption.

3.6.2 Assumption: Project

This indicates which of the development team's projects the assumption could be said to 'belong' to. This property was sometimes left blank.

3.6.3 Assumption: Category

When an assumption was first recorded in the database, a subjective judgement was made to assign to it one of the three categories defined by Lago and van Vliet (organizational, managerial and technical). These judgements were guided by the words of Lago and van Vliet (2005):

- Organizational assumptions "reflect the company as a whole"
- Managerial assumptions reflect "decisions taken to achieve business objectives"
- Technical assumptions cover areas such as "programming languages, database systems, operating systems"

3.6.4 AssumptionState: Date

This field captures the date on which a change in an assumption's state occurred. When performing assumption recovery, this field refers to the date that the change was originally recorded (in project documentation, emails, etc) rather than the date of recovery.

3.6.5 AssumptionState: Description

This text field captures a description of the event that triggered the transition in an assumption's state.

3.6.6 AssumptionState: Explicit

This Boolean property indicates whether an assumption was explicitly documented at the time of the transition in its state.

3.6.7 AssumptionState: Stability

The stability property is a subjective measure of the likelihood of failure of an assumption. As the stability of an assumption can change during its lifetime, each AssumptionState (rather than Assumption) record has a stability property.

Stability represents the inverse of the 'vulnerability' measure of an assumption, which refers to the likelihood that an assumption will fail (Dewar et al., 1993). So a highly stable assumption would be considered unlikely to fail, while low stability would indicate a highly vulnerable assumption.

The following ratings were used for estimating stability – these were intended to be used as a range of interval-type variables, although no numerical values were displayed in the database form:

- Top
- High
- Medium
- Low

- Very Low
- None

Stability, rather than vulnerability, was used because it allowed assumption failure to be represented easily - by using a ‘none’ rating.

3.6.8 AssumptionState: Impact

The impact property also involves a subjective estimate; it was used to try to judge the negative consequences (to the organization as a whole, rather than just the development team) of an assumption failing.

For live assumptions, this property might more accurately be labelled ‘estimated potential negative impact to the company’; for failed assumptions, one might use the same label with ‘potential’ removed.

The same rating scale was used as for stability. So, an assumption’s impact would range from ‘top’ to ‘none’.

3.6.9 AssumptionState: Source

This property aims to capture the type of event that accompanies a transition in assumption state. When creating an AssumptionState record, a subjective selection was made from the following options:

- Bug fix
- Change request

- Design decision
- Management decision
- Specification

These categorizations are my own. I chose not to use the 'assumption type' categories proposed by Lewis et al. (2004) - control, environment, data, usage, and convention – because they all relate to technical assumptions, of which they form only a partial subset.

With these categorizations I aimed to cover all possible sources of assumptions and assumption changes / failures within my organization, and to complement the assumption categories of Lago and van Vliet (2005).

3.6.10 AssumptionState: Action

This text property contains a description of the action taken in response to a change in assumption state, in order to reduce the impact of the assumption's failure (the potential impact for live assumptions, the actual impact for failed ones) or the likelihood of its failure.

3.6.11 AssumptionState: Authorised

This short text property indicates who authorized the action described in the Action property – the role of the person was recorded, rather than their name.

3.6.12 AssumptionState: Task Database ID

A numeric field containing the ID of a relevant record in the team's project task database. Double-clicking on this field in the database form opened a web page displaying the project task record.

3.6.13 AssumptionState: Code Revision ID

A numeric field containing the ID of a relevant code commitment in the team's source code repository. Double-clicking on this field in the database form opened a web page displaying details of the code commitment.

3.7 Design Issues

This section discusses a number of issues concerning the way the data structure was designed.

3.7.1 Actions change the state of assumptions

Although the monitoring process outlined in section 3.2.2 allows both hedging and shaping actions to be recorded, the stability and impact properties of AssumptionState came to be used as follows:

- For any AssumptionState record, the stability property would implicitly be taken to mean 'stability before the action was taken'

- The impact property would mean ‘impact after the action was taken’ - unless the AssumptionState had an empty action field (which would often be the case for newly recorded assumptions)

This meant that the data structure would be geared towards the recording of hedging actions rather than shaping actions – AssumptionState records would lack data for ‘stability after the action’ (as well as ‘impact before the action’).

The reason for this was that the initial work done with the database focused on the recovery of failed assumptions from January to March 2008. Action taken on assumption failure would then have been aimed at mitigating impact, as the assumption’s stability would have remained unchanged (i.e. ‘none’).

I do not believe that these limitations in stability / impact data invalidate the quantitative results in chapter 4 – any actions that were taken were recorded in the database, whatever their type. However, they do mean that I was not able to measure precisely the effectiveness of actions taken during assumption management.

The problem of measuring the effectiveness of actions is compounded by the data structure’s failure to specify whether actions were taken as a result of monitoring assumptions, or whether they would have happened in any case. Also, an AssumptionState’s Action property does not necessarily make it clear whether that action has actually been carried out – it really just describes the action that has been decided upon. In a small number of

cases during the AM trial, such actions were not carried out. Section 3.8.2 offers an example of this, where the assumption's stability was affected by the action not taking place – it would remain live until the action was performed, at which point it would be marked as failed.

An action may also have its own significant impact – for example, an expensive action might be taken to mitigate the effect of a potentially catastrophic assumption failure. In this research, the cost of action has not been considered - the issue did not arise because most of the actions recorded were inexpensive.

3.7.2 Stability and impact used as interval-scale variables

The stability and impact properties were subjective measures judged on an interval-type rating scale (Wohlin et al., 2000 p.27); in chapter 4, this allows me to calculate the arithmetic mean of a group of such variables, and to test for correlation between the variables using the Pearson coefficient.

Interval variables have "all the features of ordinal measurements, and in addition equal differences between measurements represent equivalent intervals" (Wikipedia, 2008b). As stability and impact are subjective measures, I would not make any great claims for the accuracy of calculations based on these variables; however, I would argue that those calculations have value as a 'rough guide' – my intention during data collection was to use the ratings as an interval scale rather than an ordinal scale.

In section 3.9 I describe my attempts to validate these subjective measurements.

3.7.3 Initial AssumptionState records

During data collection, it was often the case that a previously unidentified assumption would change or fail – indeed, all the assumption failures recovered from January – March 2008 were examples of this.

Where a previously unrecorded assumption changed or failed, two AssumptionState records were entered: one for the change / failure, and another describing the initial state of the assumption. This was to allow changes in assumption properties to be measured quantitatively when assumptions changed state.

This 'initial state' record would usually have an empty date field (e.g. Fig. 3) because the assumption would have been made before the start of the AM implementation. It would also usually have an empty action field.

The addition of the 'initial state' record would involve making subjective, retrospective estimates of the stability, impact, source and explicit properties – asking questions such as: 'what would I have estimated this assumption's stability to have been at the time?'

It may be argued that it is impossible to make such subjective judgements in retrospect, and that an element of hindsight bias (Wikipedia, 2008a) will always be present. It would be difficult to disprove this, although a couple

of real-world examples may suggest that such retrospective judgements are often quite straightforward, and are possible to achieve with sufficient accuracy for the purposes of assumption management. For instance, 'The Titanic is unsinkable' would surely have had a 'top' stability rating, while 'The Eastern Bloc will not collapse' would also have been considered a pretty stable assumption in, say, 1980.

3.7.4 'Potential impact' differs from 'actual impact'

For a live assumption, the impact property refers to its 'potential impact' – an estimate is made of what the negative consequences would be were the assumption to fail. For a failed assumption, the impact property is still estimated subjectively, but the judgement is made on the actual consequences of failure – we might call this 'actual impact'.

Of course, assumptions are not always going to fail in the way we expect, so we should not expect the actual impact of a failed assumption (before any action is taken) to be the same as its potential impact as estimated before its failure.

But there may also be psychological differences in the way we estimate the impact of live and failed assumptions; our judgement of potential impact may be influenced by pessimism, optimism or lack of knowledge, whereas actual impact would seem much easier to gauge.

This may make us somewhat wary of comparing the impact property of an assumption in its different states – i.e. before and after change or failure.

3.8 Examples

There follow three examples of assumptions that were identified during the research.

3.8.1 Example 1: failure of a previously unidentified assumption

Table 2 shows how an implicit technical assumption suddenly failed due to an application being updated. As explained in section 3.7.3, the first assumption state record was entered retrospectively on assumption failure, and contains an empty date property. This record 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.

3.8.2 Example 2: a previously unidentified assumption changes

Table 3 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).

3.8.3 Example 3: a new assumption is identified, then changes

Table 4 shows an example of what Lehman (1989) proposed – capturing and recording assumptions that are implicitly embedded in new software specifications.

In early May 2008, I judged the assumption to be very stable; however, my team ran into problems later that month, when we found that we did not have the security permissions we needed for the project. As the AM implementation finished, we were still trying to resolve the problem – so the assumption was still live.

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 2: Failure of a previously unidentified assumption

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

Table 3: A previously unidentified assumption changes

Assumption	
ID	48
Title	It is technically possible to access the company's Exchange data programmatically
Category	Technical
Project	Year Planner
Assumption state 1	
Date	09/05/2008
Description	The marketing team want to be able to print an Exchange calendar on different-sized paper (A4 - A1). No third-party software allows this at present, so we would have to develop a system to do it. This would involve accessing the data programmatically - the technology exists for us to do this
Explicit	Yes
Stability	High
Action	Test the technology to see if programmatic access is possible
Action Authorized by	Events co-coordinator
Impact	Very Low
Task Database ID	969
Code Revision ID	-
Source	Specification
Assumption state 2	
Date	26/05/2008
Description	The technology is not working. We think we have identified the problem - but we can't access the Exchange server, which is now controlled by the company's parent company, in Denmark.
Explicit	Yes
Stability	Low
Action	Contact the parent company's IT staff, to try to get the technical problem fixed
Action Authorized by	IT Development Manager
Impact	Very Low
Task Database ID	969
Code Revision ID	-
Source	Bug Fix

Table 4: A new assumption is identified, then changes

3.9 Validation of subjective measures

All the work done during this research project was carried out by one person (myself). Much of that work involved a large degree of subjective judgement - in identifying new assumptions, assumption changes and failures, and in assigning stability and impact ratings to those assumptions.

The obvious danger with such subjective measurement is that I might have allowed bias to affect my judgement, if not consciously then perhaps unconsciously.

In an effort to validate the reliability of this approach, I made two copies of the database, in which the category, impact, stability and source properties had been cleared of data.

I handed these database copies to two colleagues - the other developer in my team and my departmental manager. I explained the data structure to them, and asked them to rate the aforementioned properties of each assumption in each of its states, as I had done during data collection.

I also undertook this 're-rating' exercise myself, to see if my judgement had altered in the six months since the original data collection period had finished.

Chapter 4 Results

4.1 Overview

This chapter presents and analyzes the data collected during the research.

The chapter consists of the following sections:

Section	Description
Preliminary results	A brief overview of the quantitative data collected
Key indicators	An attempt to identify quantitative measures of AM success
Assumption failures	Analysis of the properties of failed assumptions
Assumption changes	Analysis of the properties of assumption changes
New assumptions	Analysis of the properties of new assumptions
Validation of subjective measures	Results of the experiment described in section 3.9
Reflections on assumption management	Qualitative feedback - a list of observations on the process of collecting data, and on the AM implementation

Table 5: Chapter 4 overview

4.2 Preliminary results

A simple breakdown of the data recorded (Fig. 4) shows that during the three months in which assumption management was performed:

- 14 previously unidentified assumptions were found to have failed

- 8 previously unidentified assumptions changed (without failing)

  Two of these then went on to fail before the end of June

- 11 new assumptions were recorded – one of which had changed by the end of the period

At the end of this period, 17 assumptions remained live. Seventeen assumption failures were recovered from the three months prior to the assumption management trial – this meant that overall, 50 assumptions were recorded.

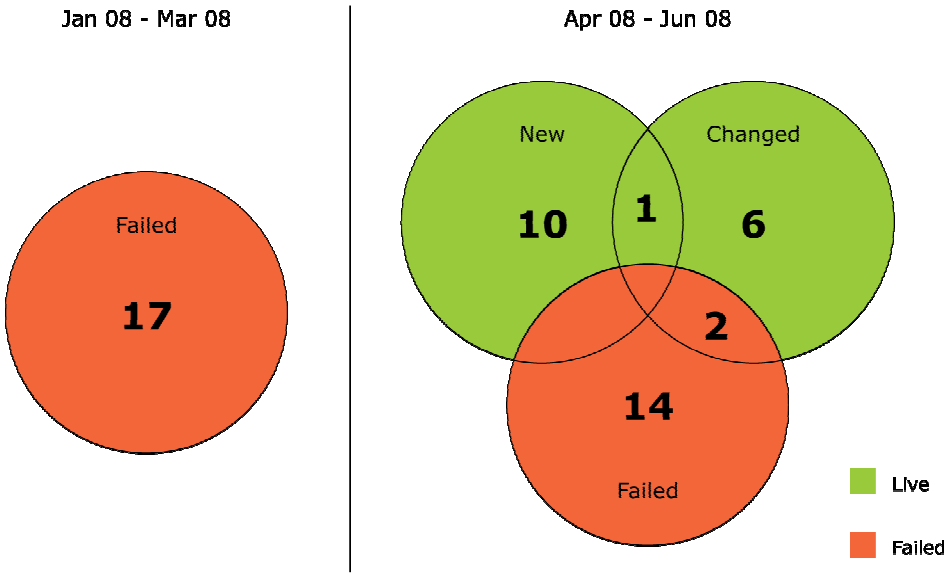

Fig. 4: Simple breakdown of assumption data

4.3 Key indicators

How can the successfulness of assumption management be measured quantitatively? As time passes, an increasing number of assumptions would be identified and monitored, with action taken to reduce their impact. In the

long term, more and more failing assumptions would already have been identified and acted upon. So one might expect the key indicators of a successful AM implementation to include:

	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 the failing assumptions that had not been previously identified would be a sign of AM's success

Table 6: Key indicators

Judging by the small number of assumptions recorded, it is likely that the timeframe of the current research was too short for the above indicators to show – only 11 new assumptions were recorded in three months of assumption management, and all of these were live at the end of that time. This suggests that assumptions might typically live undetected for many years before failing suddenly.

However, I have tried to gauge the presence of these indicators by taking the measurements presented here.

Table 7 shows for each month (from January – June 2008) the number of failures of high-impact assumptions (i.e. where impact before failure was judged to be 'top' or 'high'), the percentage of such failures of the total number of assumption failures, and the number of failures of previously unidentified assumptions.

Month	Assumption failures	High-impact failures	Percentage of high-impact failures of total failures	Failures of previously unidentified assumptions
Jan 08	6	-	-	6
Feb 08	6	1	17	6
Mar 08	5	2	40	5
Apr 08	7	1	14	7
May 08	5	-	-	4
Jun 08	4	-	-	3

Table 7: High-impact assumption failures by month

The results show no discernible trend in the numbers of high-impact assumption failures. The numbers involved are just too small – perhaps an indication that high-impact failures just do not happen very often. So, key indicator no.1 is not present.

During the AM implementation, two of the assumptions that failed had been recorded previously. In both cases, these assumptions were first identified in April, when they were found to have fallen to 'low' stability. Action was then taken to reduce the impact of assumption failure - although strangely, in both cases the impact remained unchanged; this issue is addressed more fully in section 4.5. The assumptions then failed a few weeks later.

So these results do provide some evidence of key indicator no.2 - a decrease in failures of previously unidentified assumptions – even though no decrease in impact was recorded.

In summary, key indicator no.1 was absent, while there is some evidence for the presence of indicator no.2. But the data seems insufficient to allow conclusions to be drawn - I think that a truer test of these indicators would require a longer timescale.

4.4 Assumption failures

This section analyzes the assumptions that failed between January and June 2008. The following assumption properties are considered:

Property	Scale type	Description
Category	Nominal	The category of the assumption that failed
Month	Ordinal	The month in which the assumption failed
Source of assumption	Nominal	The source property (e.g. Bug Fix, Specification) of the first AssumptionState record for the assumption.
Source of failure	Nominal	The source of the change in assumption state that accompanied the failure – e.g. Bug Fix, Specification
Stability	Interval	The estimated stability of the assumption before its failure
Explicit	Nominal (2) – Yes / No	Whether or not the assumption was explicit at its time of failure
Impact before failure	Interval	The impact of the failure of the assumption, as estimated in the AssumptionState record prior to the assumption's failure
Impact after failure	Interval	The impact of the failure of the assumption, as estimated after failure (and after any action aimed at reducing the impact)

Table 8: Assumption failures: properties analyzed

4.4.1 Assumption failures - by month

Month	Assumption failures	Average stability	Percentage of assumptions that were explicit	Average impact before failure	Average impact after failure
Jan 08	6	3.17	50	2.17	1.33
Feb 08	6	3.67	50	2.50	2.00
Mar 08	5	4.40	20	2.40	2.40
Apr 08	7	4.00	14	2.14	1.71
May 08	5	3.40	40	2.20	1.40
Jun 08	4	3.50	75	2.25	1.50

Table 9: Assumption failures by month

The failed assumptions from the 1st and 2nd periods of data collection are fairly similar in terms of quantity, stability, explicitness and impact. Assumptions that failed during March and April scored highest for stability, and lowest for explicitness.

Generally, the actions taken on assumption failure lessened the impact of failure.

4.4.2 Assumption failures - by category

Category	Assumption failures	Average stability	Percentage of assumptions that were explicit	Average impact before failure	Average impact after failure
Managerial	16	3.56	56	2.38	1.75
Organizational	6	3.50	50	1.33	1.33
Technical	11	4.00	9	2.64	1.91

Table 10: Assumption failures by category

Month	Managerial	Organizational	Technical
Jan 08	4	-	2
Feb 08	1	2	3
Mar 08	4	1	-
Apr 08	1	3	3
May 08	3	-	2
Jun 08	3	-	1

Table 11: Assumption failures by category / month

Most of the assumptions that failed were either managerial or technical. One might expect organizational assumptions to be greater in scale – perhaps affecting an organization as a whole, rather than just the software development team. This may mean that they are fewer in number, that they tend to fail less often, or that their visibility would be limited to managers at higher levels than mine.

The high-level nature of organizational assumptions might lead us to expect them to have a high impact on the company. However, these results do not bear this out, nor do they follow the expected pattern of action causing a

decrease in impact. This may be due to the small number of organizational assumptions involved.

The pre-failure stability of the assumptions seems quite high for each category – between 3.5 and 4 over the six months. It is unclear what this might mean; perhaps it says something about the way I collected the data – i.e. that the failure of a previously stable assumption might be more likely to be recorded, with more minor events going undetected. This was not my intention, but it could have happened subconsciously.

Almost all the technical assumptions that failed were implicit; perhaps this is because of the detailed nature of technical information – a fine-grained assumption may escape detection more easily. However, this might lead one to expect that most failed assumptions would be technical – this is not the case here.

4.4.3 Assumption failures - by source of assumption

Source	Assumption failures	Average stability	Percentage of assumptions that were explicit	Average impact before failure	Average impact after failure
Bug Fix	-	-	-	-	-
Change Request	-	-	-	-	-
Design Decision	6	4.17	0	3.00	1.83
Management Decision	5	4.60	20	3.40	3.00
Specification	22	3.36	54	1.95	1.41

Table 12: Assumption failures by source of assumption

Month	Bug fix	Change request	Design decision	Management decision	Specification
Jan 08	-	-	2	-	4
Feb 08	-	-	2	1	3
Mar 08	-	-	1	3	1
Apr 08	-	-	1	1	5
May 08	-	-	-	-	5
Jun 08	-	-	-	-	4

Table 13: Assumption failures by source of assumption / month

Most assumptions originated in specifications. None came from bug fixes or change requests – these tend to occur later in the software lifecycle, and may more often be symptoms of assumption failure, rather than the source of assumptions (see section 4.4.4). Also, change requests and bug fixes are often much smaller in scope than the other assumption sources, and so might be less likely to contain assumptions.

Failing assumptions that stemmed from design or management decisions were mostly implicit and considered very stable. Specification-based assumptions tended to be slightly less stable, and were more often explicit.

The difference in explicitness seems to reflect the way the development team performs its documentation: specifications and change requests are always recorded fully and precisely and confirmed by project clients. Design decisions are often taken on an ad-hoc basis, and may (or may not) be recorded in the project task database. Management decisions are usually taken outside the team, and may often be undocumented.

For each source, the impact of assumption failure tended to be reduced by action taken on failure.

4.4.4 Assumption failures - by source of failure

Source of failure	Assumption failures	Average stability	Percentage of assumptions that were explicit	Average impact before failure	Average impact after failure
Bug Fix	10	4.10	9	2.60	2.00
Change Request	16	3.13	69	2.06	1.31
Design Decision	-	-	-	-	-
Management Decision	6	4.67	17	2.50	2.50
Specification	1	3.00	100	1.00	1.00

Table 14: Assumption failures by source of failure

Month	Bug fix	Change request	Design decision	Management decision	Specification
Jan 08	1	4	-	-	1
Feb 08	2	2	-	2	-
Mar 08	1	1	-	3	-
Apr 08	4	2	-	1	-
May 08	2	3	-	-	-
Jun 08	-	4	-	-	-

Table 15: Assumption failures by source of failure / month

Hardly any assumption failures were triggered by specification work; most were brought about via change requests and bug fixes. It is perhaps unsurprising that many failures came from change requests; they often follow the realisation that a piece of software needs to change in order to cope with changes in its environment.

Looking through the ‘bug fix’ cases, many of them follow a similar pattern to the change requests – i.e. they involve software becoming faulty due to a change in its operating environment. This would explain the relatively high number of assumption failures coming from bug fixes.

These results, along with those in the previous section, suggest a tendency for assumptions to originate in specification documents, and fail when changes are required later in the software lifecycle.

Assumptions failing because of management decisions and bug fixes tended to be very stable, and were mostly implicit. This seems to indicate a correlation between stability and explicitness. A possible explanation is that more stable assumptions may are more likely to be ‘taken for granted’, and left unstated.

The assumptions that failed because of management decisions seem to have been considered particularly stable. This could be explained by considering how it can often take radical, high-level decisions to effect profound change in an organisation. However, bug fixes were the source of failure of (on average) highly stable assumptions, which appears to weaken that argument.

4.4.5 Assumption failures - by explicitness

Explicitness	Assumption failures	Average stability	Average impact before failure	Average impact after failure
Explicit	13	2.92	2.08	1.46
Implicit	20	4.20	2.40	1.90

Table 16: Assumption failures by explicitness

Month	Implicit	Explicit
Jan 08	3	3
Feb 08	3	3
Mar 08	4	1
Apr 08	6	1
May 08	3	2
Jun 08	1	3

Table 17: Assumption failures by explicitness / month

More failed assumptions were implicit than explicit; the implicit assumptions tended to be considered more stable than the explicit assumptions.

As seen in the previous section, 9 of the 12 assumption failures in March / April occurred either via bug fixes or management decisions – this would account for the high number of implicit assumptions failing during these months, and also the negative correlation between stability and explicitness indicated by these results.

4.4.6 Assumption failures – by category / source of assumption / source of failure

		Source of assumption				
		Bug fix	**Change request**	**Design decision**	**Management decision**	**Spec.**
Category	**Managerial**	-	-	2	3	11
	Organizational	-	-	1	1	4
	Technical	-	-	3	1	7

Table 18: Assumption failures – category v source of assumption

		Source of failure				
		Bug fix	**Change request**	**Design decision**	**Management decision**	**Spec.**
Category	**Managerial**	-	12	-	3	1
	Organizational	2	2	-	2	-
	Technical	8	2	-	1	-

Table 19: Assumption failures – category v source of failure

		Source of failure				
		Bug fix	**Change request**	**Design decision**	**Management decision**	**Spec.**
Source of assumption	**Bug fix**	-	-	-	-	-
	Change request	-	-	-	-	-
	Design decision	4	2	-	-	-
	Management decision	1	-	-	4	-
	Specification	5	14	-	2	1

Table 20: Assumption failures – source of assumption v source of failure

Of all the possible combinations of assumption category, assumption source and source of failure, a few occur much more frequently than the others:

- 12 managerial assumptions failed via change request
- 8 technical assumptions failed via bug fix
- 14 assumptions originated in specifications and failed via change request
- 11 managerial assumptions originated in specifications (of these, 8 failed via change request – this is not shown in tables 18 - 20)
- 4 of the 5 assumptions that originated in a management decision failed via management decision

These results suggest that most assumptions tend to fit one of a limited number of 'assumption profiles' – i.e. combinations of category, source of assumption and source of failure.

If further research supports the existence of these profiles, they may prove useful in helping to focus assumption management efforts where they are most likely to be needed; for instance, change requests might be examined for failed management assumptions, while developers carrying out bug fixes might focus on identifying failed technical assumptions.

4.4.7 Assumption failures - stability / impact

In this section I check for correlation between the three interval-type rating scales used:

- Impact before failure – the impact property of the AssumptionState record prior to the assumption's failure
- Stability before failure - the stability property of the AssumptionState record prior to the assumption's failure
- Impact after failure – the impact property of the AssumptionState record in which assumption failure is recorded

I have calculated both Pearson and Spearman correlation coefficients, as these variables are based on subjective rating scales, which may cast some doubt over their validity as an interval-type scale (see section 3.7.2). In Figs. 5-7, zero refers to a 'none' rating, while 5 is equivalent to a 'top' rating. The size of the circles represents the number of assumptions.

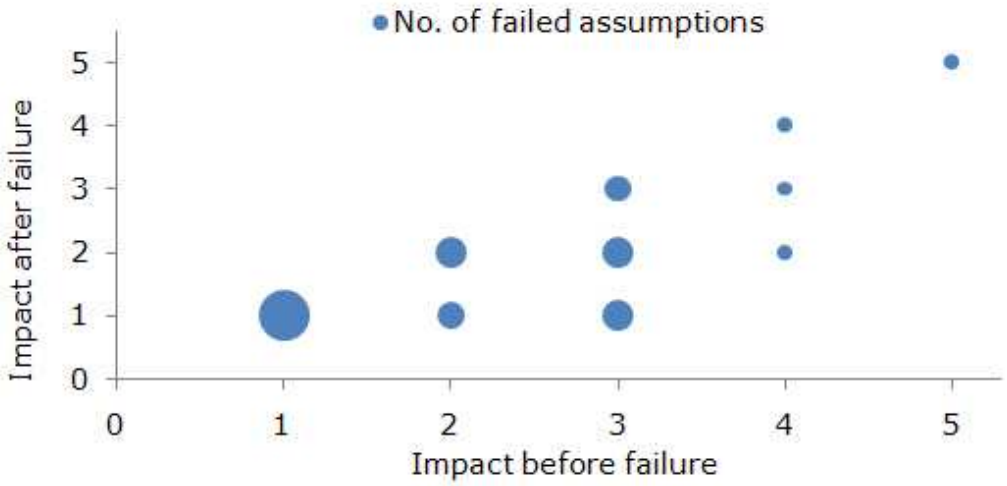

Fig. 5: Assumption failures: impact before failure v impact after failure

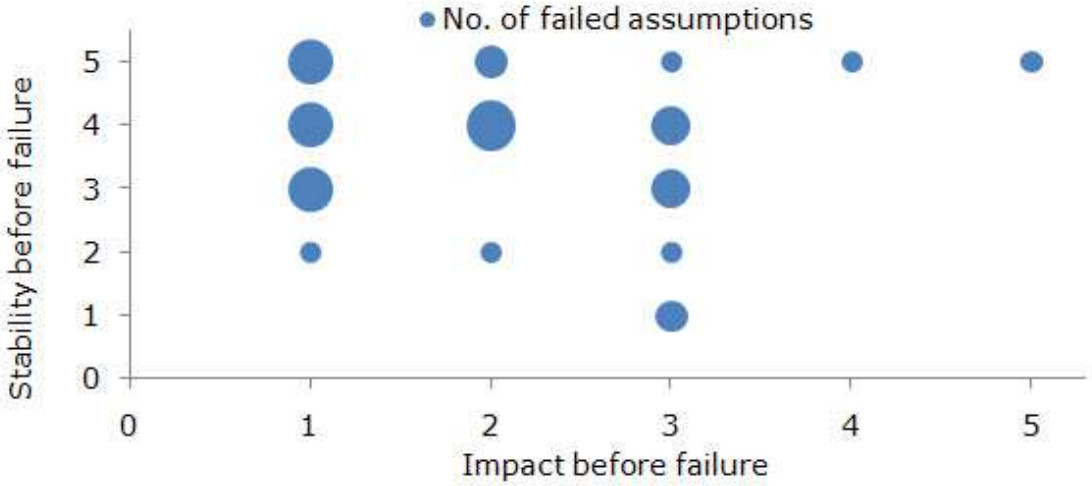

Fig. 6: Assumption failures: impact before failure v stability before failure

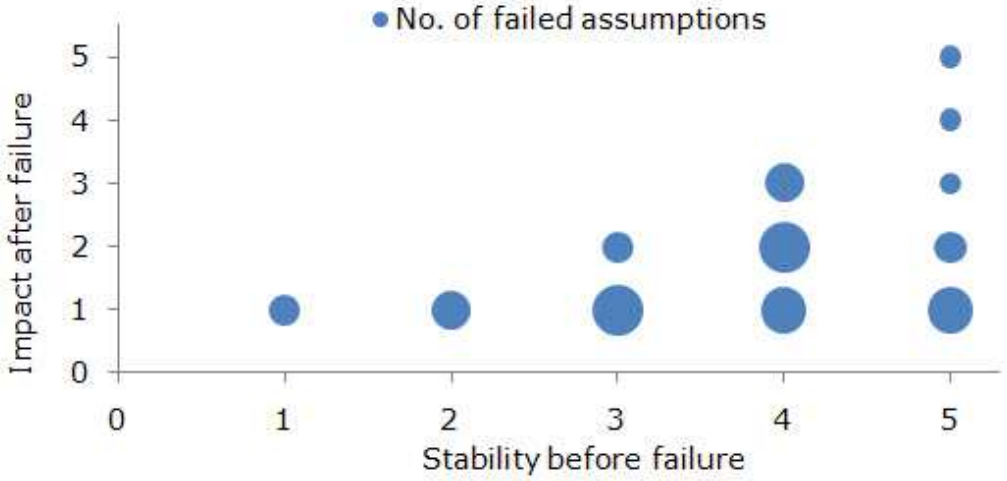

Fig. 7: Assumption failures: stability before failure v impact after failure

Figure	X axis	Y axis	Pearson	Spearman
Fig. 8	Impact before failure	Impact after failure	0.76	0.77
Fig. 9	Impact before failure	Stability before failure	0.02	-0.01
Fig. 10	Stability before failure	Impact after failure	0.44	0.51

Table 21: Assumption failures - stability / impact

These results show a very strong correlation between impact ratings before and after assumption failure; this suggests that action taken following assumption failure did not have a great effect in reducing the impact of the failure. Other results support this by showing that, in general, impact decreased only slightly as a result of those actions.

The strong positive correlation between stability before failure and impact after failure is more intriguing; Fig. 7 shows that impact after failure was never higher than the assumption's stability before failure. It suggests that unstable high-impact assumptions were already being identified and dealt with before they reached the point of failure – i.e. that the team's normal development process was taking care of such dangerous risks.

4.5 Assumption changes

There were only nine assumption changes (i.e. non-failures) during the three months of the assumption management trial, so instead of repeating the detailed set of breakdowns I performed on assumption failures, the changes are listed below.

The same properties are analyzed as for assumption failures, along with one addition: the stability of the assumption after it changed. Also, one of the assumptions had previously been identified as a new assumption – it is highlighted in grey.

Month	Category	Stability before change	Stability after change	Impact before change	Impact after change	Explicit before change	Source of assumption	Source of change
Apr 08	Technical	4	2	1	1	No	Design Decision	Bug Fix
Apr 08	Managerial	3	1	3	3	No	Specification	Change Request
Apr 08	Managerial	4	1	3	3	No	Specification	Change Request
May 08	Technical	4	4	3	3	Yes	Specification	Spec.
May 08	Organiza-tional	3	3	4	4	No	Specification	Change Request
May 08	Managerial	5	2	2	2	Yes	Specification	Change Request
May 08	Technical	4	2	1	1	No	Specification	Bug Fix
Jun 08	Managerial	4	1	3	3	No	Specification	Change Request
Jun 08	Managerial	4	2	4	4	No	Specification	Change Request

Table 22: Assumption changes

Compared with assumption failures, relatively few assumption changes were recorded (nine changes versus 16 failures). The only explanation I can offer for this is that changes were somehow harder to recognize than failures – perhaps I was mentally geared towards identifying one rather than the other. I was not aware of this during data collection.

All but one of the changed assumptions were previously unidentified. This seems to be a result of the short timeframe used; I would expect a longer period of assumption management to lead to a greater number of changes in previously-known assumptions.

The results in Table 22 suggest the existence of the same 'assumption profiles' proposed in section 4.4.6 - managerial assumptions often changed due to change requests, while technical assumptions often changed via bug fixes.

I was surprised to discover that in no case did an assumption's impact property decrease as a result of action taken following assumption change. Again, this may have something to do with perception; my suspicion is that I was less likely to downgrade an assumption's impact while the assumption was live – i.e. while the risk had not yet materialized.

This relates to the point made in section 3.7.4; in all these cases, the assumptions' impact properties all refer to 'potential impact'; it could be that the changes in assumption state, and the resulting actions, were not drastic enough to alter the estimates of the assumptions' potential impact.

4.6 New assumptions

Eleven new assumptions were identified during the AM trial period. One of these assumptions was later recorded as having changed; it is highlighted in grey.

Month	Category	Explicit	Stability	Impact	Source
Apr 08	Technical	Yes	4	1	Change Request
Apr 08	Technical	Yes	4	1	Change Request
Apr 08	Technical	Yes	3	2	Change Request
Apr 08	Technical	Yes	4	4	Management Decision
Apr 08	Managerial	Yes	3	4	Specification
May 08	Technical	Yes	4	3	Bug Fix
May 08	Organizational	Yes	3	2	Change Request
May 08	Technical	Yes	4	1	Specification
Jun 08	Technical	Yes	1	1	Design Decision
Jun 08	Managerial	Yes	3	2	Change Request
Jun 08	Managerial	Yes	1	3	Change Request

Table 23: New assumptions

Most new assumptions originated in change requests rather than specifications – this is contrary to the results for assumption failures, where most assumptions came from specification documents, and none at all originated in change requests.

This may have something to do with the short research timeframe; change requests are small and many; specifications are much larger documents and are created less frequently. Over a longer period of time, I might expect a greater proportion of new assumptions to stem from specifications.

A large proportion of the new assumptions were technical and stable; relatively few were managerial assumptions, and these were generally less stable. I think there is too little data to draw any conclusions about this, but

it seems possible that technical assumptions are many, with few changes / failures, while managerial assumptions are fewer, but more prone to failure.

4.7 Validation of subjective measures

4.7.1 Validation of impact / stability ratings

	Stability (Pearson / Spearman correlation with original research)	Impact (Pearson / Spearman correlation with original research)
Developer	0.95 / 0.90	0.44 / 0.48
Department manager	-0.13 / 0.01	0.44 / 0.49
Myself (6 months later)	0.88 / 0.91	0.60 / 0.77

Table 24: Validation of impact / stability ratings

The impact / stability ratings assigned to assumptions during the AM trial were checked for correlation with the ratings of two of my colleagues, and with my own re-rating effort performed six months after initial data collection (see section 3.9). As in section 4.4.7, both Pearson and Spearman calculations are used.

For stability, a very strong positive correlation was observed between my scores and those of my developer colleague – but there was virtually no correlation between my scores and those of my team manager.

I think these results - combined with the answers to the follow-up questions in section 4.7.3 - suggest that the stability scale used can be used

effectively throughout a team. However, the ratings, and perhaps the concept of stability in particular, need to be explained carefully, particularly to managers who may not have day-to-day involvement with the team's work.

For impact, a more uniform picture emerges. A moderate positive correlation is observed between my original impact ratings and those of colleagues, the strongest link being with the re-ratings I made six months after data collection.

A different comparison of the impact ratings is presented in Fig. 8. I grouped the AssumptionState records according to the original impact ratings I gave them during the AM trial period, and calculated (for each group) the mean of the ratings given by my colleagues (and by myself six months later).

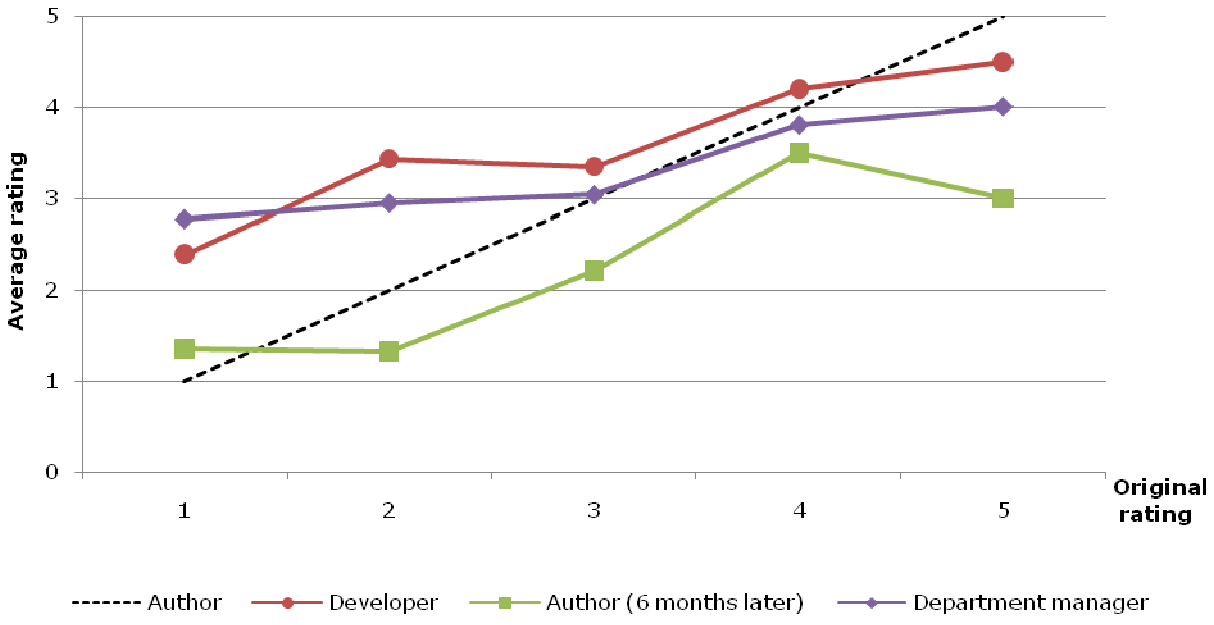

Fig. 8: Average impact ratings (grouped by original impact rating)

So, Fig. 8 shows that the AssumptionState records that I originally gave a '1' rating to received an average rating of 2.78 by my department manager, and an average rating of 2.39 by the developer.

My own later impact ratings are consistently lower than those of my colleagues, and are also lower than my original ratings. However, the graphs for all three sets of 'validation ratings' share a broadly similar trajectory, markedly flatter than that for the original ratings. This indicates a more conservative approach to rating, tending more towards the middle options than the extremes.

The fact that the 'validation rating' graphs share similar trajectories might offer some encouragement for future research into implementing assumption management throughout a development team.

4.7.2 Validation of category / source ratings

Table 25 shows how my original judgements of the category and source properties match up with those of my colleagues.

	Percentage of Assumption category ratings in agreement with original research	**Percentage of AssumptionState source ratings in agreement with original research**
Developer	96	84
Department manager	48	51
Myself (6 months later)	74	57

Table 25: Validation of category / source ratings

Because both properties involve nominal variables, I have presented the percentage of the ratings that were the same. So, my developer colleague and I assigned the same category to 96% (48 out of 50) of assumptions.

My original ratings match remarkably closely with those of the developer – more so than even my own later efforts. Agreement on ratings was lower with my department manager, who was not briefed as well as he might have been (see the following section). I believe these results suggest that the category and source options are defined well enough to be useful in a development team, but (as with stability) they also indicate that the options would require clear explanation if they were to be applied consistently.

4.7.3 Follow-up questions

After the rating exercise, I put two follow-up questions to my colleagues:

1. Did you find the database:
   (a) Very easy to use
   (b) Quite easy to use
   (c) Quite difficult to use
   (d) Very difficult to use

2. Did you find the assumption descriptions:
   (a) Very easy to understand
   (b) Quite easy to understand
   (c) Quite difficult to understand
   (d) Very difficult to understand

The answers are presented below:

	Question 1	Question 2
Developer	(a)	(b)
Department manager	(c)	(c)

Table 26: Validation of ratings - follow-up questions

This suggests that I did not explain the database – or the assumption management process - sufficiently well to my department manager. He has far less involvement with the team's work than the developer, and would evidently have benefited from greater more support in learning how to use the database.

4.8 Reflections on assumption management

This section contains my practical observations on implementing assumption management.

4.8.1 Recovering assumptions

My attitude to recovering assumptions could be summed up as: 'find as many as you can, and try not to miss anything important'. Like Lehman and Ramil (2001) I did not imagine I could find every assumption.

In the early weeks of the AM trial period, I found assumption recovery very difficult and time-consuming. I had no experience of assumption

management, and was not used to looking at situations (or documentation) in terms of assumptions - I lacked 'assumption-awareness'.

I found it particularly difficult to identify new assumptions - much more so than finding assumption changes / failures. I found that new assumptions were somehow buried more deeply in the text of the documents I was scanning, and required more effort to put into words.

A technique that I used to try to help me to identify new assumptions was to first look for risks. For example, when analysing a new system requirement, I would first ask myself 'what could go wrong here?' before tackling the question 'what assumption is implied here?'

Over the course of the three months, assumption recovery did become easier. I somehow became more adept at spotting assumptions, and found that recovery tended to take less time – down from over three hours a week at the beginning to under two hours a week by the end.

4.8.2 Using the database

I found the database very easy to use – this was not surprising, as I designed it! Some mixed feedback on the database's ease of use is provided in section 4.7.3 – it would seem that the database is easy to use as long as it - and the AM process - are explained well enough.

Overall, I felt that the database was well suited to doing the job required of it – complementing the assumption lifecycle model by allowing new assumptions, and assumption changes and failures to be recorded.

However, while using the database I became aware of a number of problems with its data structure - these are catalogued in section 3.7.

4.8.3 How agile was assumption management?

In some ways, assumption management was found to complement agile development well:

- Assumption management provided a way of managing risk and responding to change quickly
- The AM method used was simple, and yet captured enough information to enable assumptions to be monitored
- A simple (database) tool was used to capture assumption information – this was judged 'very easy to use' by my developer colleague
- Assumption descriptions were kept short, and used everyday language – allowing for easy communication with project clients
- Assumptions were monitored on a weekly basis – in keeping with the short development cycles of agile development

But there were aspects of the AM process that were not so agile:

- Assumption management was carried out by just one person. Given the emphasis that Agile Manifesto places on "self-organizing teams" (Beck et al., 2001), I think AM would need to be successfully evaluated within a team in order to be considered a fully-fledged agile method
- The database did not capture whether actions were carried out as a result of assumption monitoring – so it's hard to tell how effective AM was at providing the "timely feedback" referred to by Lewis et al. (2004)
- My department manager found the database difficult to use, and the assumption descriptions difficult to understand
- The database was not well integrated with other tools (such as the development environment, project task database and source code repository). What integration there was involved copying IDs of records in other systems
- AM was not 'non-disruptive' (see following section)

4.8.4 Was assumption management 'non-disruptive'?

My original intention was to record new assumptions, assumption changes and failures as they occurred – for example, during requirements work with project clients. This was to be achieved in a "non-disruptive" way (Lewis et al., 2004) – a way that did not disrupt the team's normal work.

In the event, I found that this was not possible. The switch in thinking required to manage assumptions 'on the fly' was too difficult for me - for

example, spotting a new implicit assumption while in the middle of a requirements meeting, or guiding a project client through an assumption-oriented discussion of their change requests.

Instead, I performed assumption recovery once a week to identify new assumptions and monitor existing ones (this usually took around two hours). This means that while the AM work was indeed non-disruptive, it was not fully integrated with the team's development process.

However, I did gradually develop an 'assumption-awareness'; I began to find myself asking assumption-oriented questions in project meetings, using phrases such as: "what are we assuming here?" or "aren't we assuming that...?" On one particular occasion, I instinctively (and successfully) turned to an assumption-based approach to explain a difficult concept to a project client.

This increased sensitivity to assumptions made assumption recovery easier and quicker, as I became more confident in my ability to scan through documentation looking for new, changed and failed assumptions.

Chapter 5 Conclusions

This research proposed a simple, lightweight method for assumption management, and sought to evaluate the method within a small, agile software development team. Assumptions were managed for a period of three months, and assumption failures recovered from the three months prior to that. Data was recorded in a simple database.

In all, 50 assumptions were identified. During the three months of assumption management 11 new assumptions, 9 assumption changes and 16 assumption failures were recorded. The three months prior yielded 17 assumption failures.

5.1 The database

Overall, I found the database adequate for capturing assumption information (a colleague in my team also found it "very easy to use"). The assumption lifecycle and data structure models that underpinned the database were generally well suited to the assumption management work performed.

However, a number of design issues concerning the data structure were noted, in particular:

- The stability and impact properties were not defined clearly

- The effects of actions on impact and stability were not clearly measured
- Initial AssumptionState records contained subjective measures made in retrospect
- An impact rating could be 'potential' or 'actual' depending on the state of the assumption - but the same single impact field was used for both
- The costs of taking action were not measured
- Some actions were in fact not carried out – but the data did not make this clear
- The single description field was not sufficient to capture all relevant details of AssumptionState – e.g. consequences of actions, details of impact

Future researchers might ensure that the data structure allows the effects of actions to be better recorded. They might achieve this in either of the following ways:

- Have two stability properties and two impact properties for each AssumptionState record. These properties would be called: 'stability before action', 'stability after action', 'impact before action', 'impact after action'
- Keep just the single impact property and single stability property per AssumptionState, but add a new AssumptionState record after an action is performed. This record would show the change in stability / impact resulting from the action, while itself having an empty action property

These measures might make it easier to define and test key AM indicators.

Two Boolean properties might be added to the AssumptionState class: one to signify whether an action (associated with a given AssumptionState record) had resulted directly from monitoring assumptions, and the other to indicate whether that action had actually been carried out. Researchers may address the issue of actions having high impacts of their own by adding a 'cost of action' property to the AssumptionState class.

Extra text properties might also be added to AssumptionState - to record the consequences of an action, or to elaborate on the precise nature of an assumption's impact. The description property could be renamed 'event description' in order to more precisely reflect its use – i.e. to describe the events that accompany the making of assumptions as well as assumption changes and failures..

5.1.1 Quality of information v ease of use

The changes outlined above would all enrich and clarify AM data structure I propose in this research. But with data design, there is always a trade-off between quality of information on the one hand, and simplicity and ease of use on the other. Future researchers may wish to explore how these concerns can best be balanced – perhaps bearing in mind Ambler's motto of "just good enough" (Ambler, 2004).

5.2 Subjective judgement

The AM process depended to a large degree on subjective judgement – identifying assumptions, assumption changes and failures, and rating them for stability and impact. What's more, some of these judgements were retrospective, and all of them were made by one person.

So, any conclusions I drew from the database data would have to be very cautious ones, and would need to take into account the subjective nature of the measurements made.

That said, my ratings for assumptions' stability, impact, source and category properties were validated by my colleagues with some degree of success. This offers encouragement for those who may wish to manage assumptions as a team.

If the stability and impact assumption properties were explained clearly, I believe team-based AM would be feasible – i.e. teams would be able to achieve a common approach to the subjective judgements required by the AM process. It could be that close-knit, communicative agile teams would find this easier than most.

5.3 Quantitative data

Two key indicators were proposed for measuring whether AM had been successful - only one of these indicators was detected in the research

results. Given the small number of assumptions involved, I think a longer research timeframe would be required to give a true test of these (or other) indicators.

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 the way the potential impact of a live assumption was perceived, compared with the actual impact of a failed assumption.

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

- Assumptions regarded as more stable tended to have a higher impact when they failed
- More assumptions were implicit than explicit
- Implicit assumptions tended to be considered more stable

These tendencies may be characteristic of failed assumptions, of assumptions in general, or they may be a reflection of my judgement; future research may help to establish which of these is the case.

Many failed assumptions tended to fit one of a limited number of 'assumption profiles' – combinations of category, source of assumption and source of failure. These included:

- Managerial assumptions originating in specifications and failing via change request
- Technical assumptions failing via bug fix
- Assumptions originating and failing via management decision

If these profiles are confirmed by future research, this may help developers to focus their AM efforts more narrowly and efficiently.

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. 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.

All 11 new assumptions recorded were still live at the end of the three months of AM evaluation, and only one of them had changed in any way. This suggests that assumptions tend to live long, and fail suddenly – again, I would recommend a longer research timeframe to gain a clearer understanding of the assumption lifecycle.

5.4 Agile development

In some ways, assumption management was found to be a good fit for agile development: assumption descriptions were easily kept short, and used everyday language. The simple database seemed 'good enough' for capturing assumptions, and the weekly monitoring routine complemented the short release cycles of agile software development.

However, AM was not integrated into the team's development process in a 'non-disruptive' way, due to my difficulty in adjusting to the new 'assumption-aware' way of thinking. Instead, I limited myself to performing assumption recovery once a week.

For assumption management to succeed, developers (and managers) would need to attune to the new conceptual approach – I believe that this would have to be a gradual process. Future research may investigate how this transition may be eased; this research makes a step in this direction by offering a generalized list of the assumptions recorded, for use as an initial checklist by developers seeking to manage assumptions (see Appendix A).

Also, as agile development is a team-oriented discipline, I believe that agile AM would also need to be conducted as a team. As indicated above, this research offers some encouragement in that respect.

5.4.1 The research question revisited

To sum up, I return to the research question posed in chapter 1:

> 'How does the introduction of assumption management affect the work of a small, agile software development team?'

I can answer this question in terms of how my own software team was affected:

- Two key indicators were proposed for measuring the beneficial effects of assumption management. Evidence was found for the existence of one of these indicators in data recorded during a trial AM implementation
- AM was not well integrated into the team's development process, and was performed by just one person, rather than the team as a whole
- I gradually developed an 'assumption-awareness' that proved to be useful, particularly during requirements meetings with project clients

5.5 Future research

This research was, to my knowledge, the first to evaluate assumption management in an industrial agile development team. As such, the list of 'things I could have done better' was always going to be a long one. Here, then, is a brief list of suggestions for those wishing to evaluate agile assumption management:

For researchers:

- Evaluate AM over a longer timeframe
- Validate the correlations between properties found in this research

For practitioners:

- Monitor assumptions as a team
- Explain stability and impact clearly to users
- Measure assumption stability and impact before and after action is taken
- Measure the costs of actions explicitly
- Explicitly identify whether actions result directly from assumption monitoring, and whether those actions are actually carried out
- Record more textual information to describe assumptions – but follow the 'just good enough' principle
- Sell the idea of assumption management to senior managers
- Expand the scope of assumption management to include the business environment beyond the software team
- 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
    - Publishing policies for identifying assumptions and their changes / failures

References

Ambler, S. (2004) 'Just Good Enough' [online], *Dr. Dobb's Portal*, http://www.ddj.com/architect/184415826 [Accessed 19 October 2008]

Ambler, S. (2007) 'Survey Says...Agile Has Crossed the Chasm' [online], *Dr. Dobb's Portal*, http://www.ddj.com/architect/200001986 [Accessed 22 March 2008]

Alexander, I. (2006) '10 Small Steps to Better Requirements', *IEEE Software*, vol. 23, no. 2 (March / April), pp. 19-21

Beck, K. (1999) *Extreme Programming Explained: Embrace Change*, Addison Wesley

Beck, K. et al (2001) *Manifesto for Agile Software Development*, http://agilemanifesto.org [Accessed 21 May 2008]

Boehm B.W. (1988) 'A spiral model of software development and enhancement', *Computer*, vol. 21, no. 5 (May), IEEE Computer Society, pp. 61-72

Boehm B.W. (1989) 'Software Risk Management', ESEC'89, Springer-Verlag, pp. 1-19

Dewar, J.A., Builder, H., Hix, M., Levin, H. (1993) *Assumption-based Planning: A Planning Tool for Very Uncertain Times*, Santa Monica, RAND

Dutoit, A.H. and Paech, B (2001) 'Rationale Management in Software Engineering', in Chang, S.K. (ed.) *Handbook of Software Engineering and Knowledge Engineering Vol.1*, Singapore, World Scientific Publishing Company, pp 787-816

Fickas, S. and Feather, M.S. (1995) 'Requirements monitoring in dynamic environments', *Second IEEE International Symposium on Requirements Engineering*, 1995, p. 140

Garlan, D., Allen, R. and Ockerbloom, J. (1994) 'Architectural mismatch: why reuse is so hard', *IEEE Software*, vol. 12, no. 6 (Nov), pp. 17-26

Haley, C.B., Laney, R.C., Moffett, J.D., and Nuseibeh, B. (2003) 'Using Trust Assumptions in Security Requirements Engineering', Second Internal iTrust Workshop On Trust Management In Dynamic Open Systems, Imperial College, London UK, 15-17 Sep. 2003

Johnson, L., Greenspan, S., Lee, J., Fischer, G. and Potts, C. (1993) 'Panel on recording requirements assumptions and rationale', *Proceedings of IEEE International Symposium on Requirements Engineering, 1993*, IEEE Computer Society, pp. 282-285

Lago, P. and van Vliet, H. (2004) 'Observations from the Recovery of a Software Product Family', *SPLC 2004*, Springer, pp. 214-227

Lago, P. and van Vliet, H. (2005) 'Explicit Assumptions Enrich Architectural Models', *ICSE'05*, pp. 206-214

Lago, P., Niemela, E. and van Vliet, H. (2004) 'Tool support for traceable product evolution', *CSMR 2004*, IEEE Computer Society, pp, 261-269

Lehman M.M. (1969) 'The Programming Process', IBM Res. Rep. RC 2722. Also in Lehman and Belady (1985) Ch.3

Lehman M.M. (1974) 'Programs, Cities, Students, Limits to Growth?', *Imperial College of Science and Technology Inaugural Lecture Series*, vol. 9, pp. 211-229. Also in Lehman and Belady (1985) Ch.7

Lehman, M.M (2005) 'The Role and Impact of Assumptions in Software Development, Maintenance and Evolution', *Proceedings of the IEEE International Workshop on Software Evolvability*, IEEE Computer Society, pp. 3-14

Lehman, M.M. (1989) 'Uncertainty in computer application and its control through the engineering of software', *Journal of Software Maintenance: Research and Practice*, vol. 1, no. 1, John Wiley & Sons, Ltd, pp. 3-27

Lehman, M.M. (1990) 'Uncertainty in computer application is certain-software engineering as a control', *Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering*, IEEE Computer Society, pp. 468–474

Lehman, M.M. and Ramil, J.F. (2001) 'Rules and Tools for Software Evolution Planning and Management', *Annals of Software Engineering*, Vol. 11 No. 1, pp.15-44

Lewis, G.A., Mahatham, T. and Wrage, L. (2004) *Assumptions Management in Software Development*, Carnegie Mellon University

Merriam-Webster Online Dictionary (2007) *assumption* [online], http://www.merriam-webster.com/dictionary/assumption [Accessed 14 Mar 2008]

Miranskyy, A., Madhavji, N., Davison, M. and Reesor, M. (2005) 'Modelling assumptions and requirements in the context of project risk', *University of Western Ontario web site* [online], http://publish.uwo.ca/~amiransk/tr645.pdf [Accessed 22 March 2008]

Nyfjord, J. and Kajko-Mattsson, M. (2007) 'Commonalities in Risk Management and Agile Process Models', *ICSEA 2007*, IEEE Computer Society, p.18

Page, V., Dixon, M. and Choudhury, I. (2007) 'Security Risk Mitigation for Information Systems', BT Technology Journal, vol. 25, no. 1 (January), pp. 118-127

Ramesh, B. and Jarke, M. (1999) 'Towards Reference Models for Requirements Traceability', *Software Engineering*, vol.27, no.1, pp. 58-93

Robertson, J. and Robertson, S. (2007) 'Volere Requirements Specification Template' [online], http://www.volere.co.uk/template.htm , Edition 13 (August), [Accessed 14 March 2008]

Roeller, R., Lago, P. and van Vliet, H. (2006) 'Recovering architectural assumptions', *The Journal of Systems and Software*, vol. 79, no. 4 (April), Elsevier Science Inc., pp. 552-573

Seacord, R. (2003) 'Assumption Management', *news@sei*, vol.6, no.1 (First Quarter) [online], http://www.sei.cmu.edu/news-at-sei/columns/the_cots_spot/2003/1q03/cots-spot-1q03.htm [Accessed 22 March 2008]

Smith, P.G. and Pichler, R. (2005) 'Agile Risks, Agile Rewards', Dr. Dobbs' Portal [online], http://www.ddj.com/architect/184415308 [Accessed 22 May 2008]

Tirumala, A., Crenshaw, T., Sha, L., Baliga, G., Kowshik, S., Robinson, C., and Witthawaskul, W. (2005) 'Prevention of failures due to assumptions made by software components in real-time systems', *ACM SIGBED Review*, vol.2, no.3 (July), Association for Computing

Machinery, pp. 36-39

Uchitel, S. and Yankelevich, D. (2000) 'Enhancing architectural mismatch detection with assumptions', *ECBS 2000*, pp.138-146

van Lamsweerde, A. (2001) 'Goal-Oriented Requirements Engineering - A Guided Tour', *Proceedings, Fifth IEEE International Symposium on Requirements Engineering, 2001*, pp. 249-262

Viega, J., Kohno, T. and Potter, B. (2001) 'Trust (and mistrust) in secure applications', Communications of the ACM, vol. 44, no. 2 (February), pp. 31-36

Wikipedia (2008a) *Hindsight bias* [online], http://en.wikipedia.org/wiki/Hindsight_bias [Accessed 26 December 2008]

Wikipedia (2008b) *Level of measurement* [online], http://en.wikipedia.org/wiki/Level_of_measurement [Accessed 30 October 2008]

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B. and Wesslén, A. (2000) *Experimentation in Software Engineering: An Introduction*, Boston, Kluwer Academic Publishers

Yang Li, Hongji Yang, Chu, W. (2000) 'Generating Linkage between Source Code and Evolvable Domain Knowledge for the Ease of Software Evolution', *ISPSE 2000 Proceedings*, IEEE Computer Society, pp.196 – 205

Glossary

Assumption

"A fact or statement… taken for granted" (*Merriam-Webster, 2007*). This research is concerned with assumptions on which software programs depend in order to fulfil their purpose.

Assumption lifecycle model

A model that describes the stages that an assumption can pass through in its lifetime – once the assumption is first made, it can go through any number of changes while remaining live, then may (or may not) fail (see section 3.5).

Assumption management

The practice of identifying and recording assumptions that are made during software development, monitoring them for change, and acting to change the impact or likelihood of their failure (see section 1.1.1).

Assumption profile

A combination of values of assumption properties that is frequently observed during assumption management (see section 4.4.6)

Assumption recovery

The practice of identifying past assumptions by looking through documentation and conducting interviews (see section 3.2.4)

Assumption-awareness

An intuitive ability to identify assumptions (see section 4.8).

AssumptionState

The name given to the data element representing the state of an assumption at a given point in time. When an assumption is created, or changes in any way, a new AssumptionState record is created (see section 3.6).

Hedging action

Action taken to mitigate the impact of an assumption failing.

Impact

A property of AssumptionState that indicates the scale of the negative consequences of failure of an assumption, as judged at a given point in time (see section 3.6.8).

Shaping action

Action taken to reduce the likelihood of an assumption failing.

Source

A property of AssumptionState indicating the type of event that accompanies a transition in the state of an assumption (see section 3.6.9).

Stability

A property of AssumptionState that indicates the likelihood of failure

of an assumption as judged at a given point in time. Assumptions that are likely to fail have a ‘low’ stability rating; failed assumptions have a ‘none’ rating (see section 3.6.7).

Appendix A: List of Generic Assumptions

This list is derived from the assumptions recorded during this research; they have been grouped according to the categorizations of Lago and van Vliet (2005).

Organizational assumptions

- A project / system will always be owned by a certain person
- A system is used in the same way throughout an organization
- Everyone is subject to the same security restrictions / permissions
- People will always perform the tasks expected of them
- People will never leave the organization
- System administrators are authorized to access confidential information
- The names of organizational entities will never change (e.g. company, department names)
- The person designated as project client actually has the authority to specify the system

Managerial assumptions

- A variable implicitly used by a system will remain constant (e.g. A financial system might assume only one currency is used)
- A given authentication model is the correct one to use in a system
- A given process step may be performed manually

- A given use case will never occur - and should therefore not be built into the system
- A process step will never need to be undone (e.g. an invoice will never need to be cancelled)
- A task owned by a certain person will always be performed by that person
- A given task will never be formalized and scheduled - it will always be performed on an ad-hoc basis
- A variable used by a system has a fixed number of options
- All instances of a given piece of information fit a given format
- Data will be entered into a system correctly
- Every instance of a given piece of information is similar in some respect (e.g. every field in a table will hold a non-null value)
- Information is ordered by a given field or combination of fields (e.g. Date, Title)
- People will never want to delegate a given task to someone else (e.g. a secretary)

Technical assumptions

- A certain technology is appropriate for / accessible to all users
- A given technology will always be used for a given task
- A link between systems can actually be achieved in the way envisaged
- A subsystem / service is always used under the same conditions

- A technology will not suddenly become obsolete
- An exceptional occurrence is just that - and not part of a pattern
- An infrastructure change will not affect dependent systems
- Certain infrastructure changes will not happen (e.g. servers will not be renamed)
- Incorrect data is not doing any harm
- Two systems do not affect each other in any way