Cost-Benefits of Traceability

Claire Ingram and Steve Riddle

1 Introduction

Cost has been cited as a key reason why many projects neglect or abandon traceability efforts without reaping the full range of potential rewards. In this chapter we introduce some key issues behind maximising the cost-benefit from a traceability system. Achieving the optimal cost-benefit from traceability is about achieving the maximum return on the investment (ROI), as well ensuring that traceability data is sufficient to meet the project goals.

The ultimate purpose of any traceability strategy is to improve the performance of some future activity. The potential uses of traceability data are discussed elsewhere in this book, but trace data can be useful for: conducting impact analysis for estimating change effort; ensuring sufficient test coverage; supporting safety case or some other third party certification; identifying potential candidates for re-use; tracking project progress; reconstructing earlier decisions to avoid rework; and controlling requirements creep. Most projects will need to carry out at least a subset of these traceability-enabled tasks. If traceability data which adequately supports the task is available (e.g., this could be a list of components relevant to a requirement, a list of the reasons why a design decision was made, or a list of people involved in drafting a requirement), much time can be saved and the task can be completed to a high standard with more confidence. Ramesh et al. cite an extreme example of a project forced to back-hire engineers who had left in order to reconstruct the reasons behind original design rationale (Ramesh et al., 1995); this situation could perhaps have been avoided if the reasons for decisions had been more easily traceable. Although this is an extreme case, trace data can generally ensure that future activities like impact analysis or safety case preparation can be conducted to a sufficiently high standard in less time than otherwise.

C. Ingram (✉)
Newcastle University, NE1 7RU, England, UK
e-mail: claire.ingram@ncl.ac.uk

The availability of appropriate trace data thus reduces the effort required for many activities. But at the same time gathering and maintaining such data increases the initial project cost. There is therefore a trade-off between increasing the cost of collecting traceability data and reducing the later costs of carrying out these traceability-enabled activities. We can calculate the financial savings traceability can bring by estimating the cost of completing tasks such as impact analysis without appropriate traceability data; the cost to complete the same tasks *with* trace data (including the cost of creating and maintaining the data); and taking the difference between the two as the cost-benefit. This is illustrated in Fig. 1, where we represent two scenarios. The optimal scenario occurs when the combined cost of managing trace data and consuming it later to perform some traceability-enabled activity is less than the cost of completing these activities without any traceability data. The worst case scenario arises when managing and using trace data is actually *more* expensive than not using trace data at all. This chapter focuses on strategies for achieving the optimal scenario, which includes a combination of:

- keeping the cost of managing traceability data (including collecting and updating it) to a minimum
- ensuring the quality of data collected is of an acceptable quality to meet all project needs

This chapter addresses issues of controlling costs and trace quality in Sections 2 and 3 before discussing some general issues centering on estimating traceability costs in Section 4. In the discussions below we are working towards selecting the best traceability cost strategies to enable the maximum return on investment.

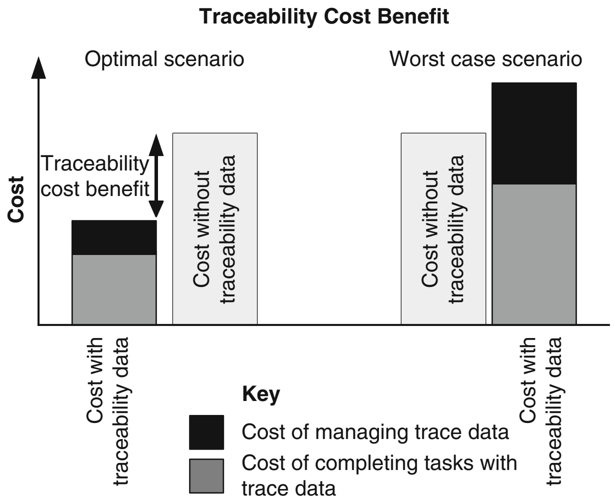

Fig. 1 Bar chart comparing the cost of completing tasks with and without traceability data

We present some possible traceability strategies in Section 5. Finally, in Section 6 we suggest a simple way to conduct a cost analysis during traceability planning.

2 Controlling Traceability Costs

In this section we discuss strategies for minimising the cost of gathering and/or storing trace data. The maximum return on investment is reached by collecting high-quality trace data at a low cost. However, there is a natural tension between these aims, since higher quality data is generally more expensive to collect and to maintain. The first step in controlling the costs of a traceability scheme, therefore, is to ensure that all trace data collected has a definite purpose, and that no effort is wasted on collecting unusable data.

2.1 Establishing Traceability Goals

Capturing, storing and maintaining *all* possible data is prohibitively time-consuming, and likely to result in an unmanageably large dataset. For even a moderately large project, systematically collecting and subsequently maintaining unnecessary data introduces a non-trivial overhead. The resultant mass of data also makes it more difficult to pick out and manage the data which is actually useful. On the other hand, it would be equally undesirable to discover at a later date that the trace data which has been collected is inadequate to support specific activities. Therefore some thought must be given to the subject of traceability at the project outset, and the purpose of the trace data clearly identified. Traceability systems may then be tailored appropriately for the individual project's needs to avoid situations where too much or too little trace data is created. In particular, if trace data is not needed in order to meet a specific goal, then it should not be collected and/or stored (Cleland-Huang et al., 2004).

Any traceability effort should start with the question: what is the main aim/purpose of the traceability data? The main activities which will later be supported by traceability, and the tools that will be used, must be identified beforehand. We outline some key questions to ask in Section 6 in this chapter, which may be helpful in determining the potential use of traceability data.

Egyed et al. introduce the notion of a threshold of usefulness (Egyed et al., 2005). That is, the tools and/or processes which will be used will normally dictate a minimum quality for the trace data. This threshold must be identified for each type of trace data. If the data will not meet this threshold then gathering it is a cost that yields no benefits at all and it should be omitted. In Section 6 of this chapter we discuss some possible pitfalls. It's worth noting, however, that there are some situations in which imperfect traceability data may still be useful – this is discussed further in Section 3.

Once the traceability aims are clear, the actual collection of data can be considered.

2.2 Trace Creation and Evolution

There are two major activities involved in ensuring that trace data is available:

- *Trace creation*, the act of creating links to link two development artifacts
- *Trace maintenance*, activities necessary to keep existing traces up to date

It's generally quicker and more accurate to create trace links between artifacts at the time that those artifacts are created (Heindl and Biffl, 2005). This is when knowledge of artifacts' purposes and interactions is at its most keen. Attempting to create trace links later, after system development has been completed, will most likely take longer because staff who originally worked on the artifacts may have left, or have forgotten (and have to search for) small details regarding linkages. There is also greater chance that useful trace links will be omitted by creating trace data later.

Any data which is captured during trace creation must then be kept up to date carefully. Anecdotal reports and academic research (for example, (Cleland-Huang, 2006; Cleland-Huang et al., 2003)) have indicated that, without active maintenance to evolve the trace links, they will gradually become less and less reliable as the system evolves and the trace links cease to reflect that. Creating more trace data commits the project to a greater maintenance effort in the future, although tools are available to support maintenance of links (Cleland-Huang et al., 2003).

2.3 Using Automated Tools

It has been suggested (Cleland-Huang et al., 2004) that a sensible tracing strategy is to "maximise the usage of dynamic link generation", such as information retrieval technology and heuristic traceability. These types of technology may not apply in all scenarios, however – for example, the precision or recall of a tool may not meet project requirements. A simple optimisation strategy is to use the tool on all requirements, and manually check the links on a subset of prioritised requirements (see Section 3.1). Egyed et al. automated the checking process, optimising an automated trace generator using a filter (Egyed, 2005). Their approach involved measuring the "strength" of each trace link generated; this was calculated as the ratio of the number of methods implementing a requirement and the number of methods that two requirements share as part of their implementation. The filter then determines a threshold of "weak" links, which can be eliminated as they are far more likely to be false positives. The optimisation proved helpful; they found that the weakest 10% of trace links contained only 1% of true traces.

3 Trace Quality

In the introduction to this chapter we said that the cost-benefits of traceability are not reached if data does not reach an acceptable quality for the project's requirements. What is meant by "acceptable quality" varies between different projects.

For example, an avionics system and an online ordering system may have different expectations regarding thoroughness of testing coverage.

We can define the "quality" of traceability data as a function of the following factors:

- the granularity of trace links
- the recall and precision of the links – that is, the number of false positives and/or false negatives that are retrieved
- the level of coverage they achieve for the system as a whole

These attributes are not fixed: the applicable minimum level can be varied for all or any of these factors. In general terms, increasing the quality (by making data more complete, more fine-grained, more accurate or more precise) tends to increase cost. One way, therefore, to control costs is to define carefully the minimum acceptable quality threshold for each type of trace data (bearing in mind that different traceability goals and different areas of the system may have differing requirements) and ensure that data gathered does not exceed it unless there is a clear extra benefit in doing so.

Varying the granularity of trace links is one way to control costs. For example, studies have been completed estimating the effort needed to link requirements variously to packages; to class files; and to individual methods (Egyed, 2005; Egyed et al., 2005, 2007). Increasing the level of granularity of the trace links (e.g., creating trace links that link to class-level as opposed to package-level) tended to increase the effort required by an order of magnitude. However, the researchers detected a decreasing marginal return on investment as the trace links become finer-grained (Egyed et al., 2007). Heindl and Biffl describe a case study where requirements were given one of three priorities (Heindl and Biffl, 2005). Trace links created for level 1 requirements traced from requirements to individual methods, whilst lower-priority requirements were linked to classes or packages. This approach reduced effort required by 30–70%, compared to a fully traced system. In many cases, the reduction in precision was not large, because some classes had few methods, and so the difference between tracing to methods and tracing to classes was small. For requirements where trace data will not be needed frequently, collecting detailed trace links may not yield a good return on investment, because most of the data gathered will never be needed. Storing some granular links initially and improving them on demand later can help to reduce wastefulness. Thus it's important to consider carefully whether a coarse collection of links would be sufficient for some (or all) traceability purposes.

There are other ways to vary the quality of the trace data which is collected. For example:

- The coverage of the system can be varied, so that traceability links do not cover all areas of the system
- The frequency of updates can be varied. Thus the most high-priority trace links may be re-evaluated and updated each time a change is enacted, whilst lower-priority links may not be updated at all after they are created, or only re-evaluated at sparse intervals.

- The use of automated link generation can be varied (discussed in Section 2.3 of this chapter).
- The recall and/or precision of links can be varied. Precision is concerned with achieving a low number of false positives. A false positive is encountered when a trace link is created that is invalid (in reality, no link should exist). Recall is concerned with achieving a low number of false negatives, or "missed" links; a false negative arises where a trace link is not created when a dependency does, in fact, exist.

Varying the recall and precision of trace data is particularly pertinent when using automated tools, as many tools can be calibrated to favour one or the other. Recall and precision frequently exist in a state of mutual tension. For example, 100% recall can be achieved simply by returning all possible links, but this results in a very low level of precision (and is not very useful).

For any traceability strategy, managers should consider whether recall or precision is more important. For safety critical projects, for example, recall will probably be more important. During many traceability-enabled activities on such a project, engineers will not want to run the risk that a link is missed, and are prepared to spend time eliminating false positives. On the other hand, a non-safety critical project with a tight deadline may prefer to favour precision, assuming that an automated tool is likely to return the most important links, and the less important can be detected later.

3.1 Ranking Requirements for Selective Traceability

So, significant cost savings can be made by focussing the higher-quality trace gathering effort on key system areas only. This strategy involves ranking the requirements. Ranking can be conducted using a variety of criteria, for example:

- Ranking on predicted volatility (Heindl and Biffl, 2005). Change-prone components are more likely to be the subject of change impact assessments, which will be much easier to carry out if good quality trace data is available. However, this tactic relies on being able to predict change-prone areas of the system. Some major sources of changes can include: customer expectations; changes to platforms and/or third-party systems; changes to relevant regulations; and market or organisational changes. Informed stakeholders may be able to predict some change-prone requirements if they are aware of these factors. Other approaches for predicting volatility use software metrics which measure complexity or software size to determine which components are more likely to change frequently (Arisholm et al., 2004; Basili et al., 1996; Briand et al., 1999; Chaumun et al., 1999; Han et al., 2008; Ingram and Riddle, 2011; Li and Henry, 1993; Ratzinger et al., 2007; Wilkie and Kitchenham, 2000).
- Ranking on predicted risk. This may be calculated see (Cleland-Huang et al., 2004) as probability multiplied by impact, where *probability* is an estimate of a requirement's likelihood of changing, and *impact* is an estimate of the business impact.

- Ranking on required reliability. Huang and Boehm, for example, assign one of five categories to requirements: loss of human life; high financial loss; moderate recoverable loss; low, easily recoverable loss; and slight inconvenience (Huang and Boehm, 2006). Applying enhanced tracebility to these areas allows critical parts of the system to benefit from better quality traceability, ensuring that later tasks can meet higher expectations of accuracy.
- Asking end users to rank requirements in terms of importance to them (Boehm and Huang, 2006; Heindl and Biffl, 2005). Boehm and Huang suggest using the DMR group's "benefits realisation approach", which provides a framework for estimating the contributions and initiatives of a requirement.

Once requirements have been prioritised, then decisions can be made as to the minimum quality of trace data needed for differently ranked requirements. In some cases, reducing the quality of trace data to be collected results in a reduction in effort but not a commensurate reduction in later cost savings, and these types of compromises may be worth making. We discuss estimations of costs and savings in the next section.

4 Cost Estimation

Estimating the effort needed for a given traceability strategy and scheduling adequate time is an important factor in achieving the maximum return on investment. It's been suggested that traceability is often the first item to be squeezed from a tight project schedule (Jarke, 1998). This could potentially represent the worst case scenario for traceability cost benefit; time spent creating links at the project's outset doesn't result in significantly less effort on later traceability-enabled activities if traceability is abandoned part-way.

As discussed in Section 3.1 of this chapter, the best traceability strategy for a given project may be one in which some compromises are made in order to ensure that other, less acceptable compromises are not necessary. For this reason, estimating the costs of different traceability tactics is important for calculating which compromises bring worthwhile savings, and which compromises are unacceptable. Determining which compromises are likely to be worth making involves:

- a good understanding of the relative importance and impact of different project areas (ranking of requirements is discussed in Section 3.1)
- understanding the cost impacts of various traceability tactics (including the costs of creating trace links and the potential savings in effort later on) compared with the costs of activities without trace data

We briefly introduce some key ideas underpinning cost estimation and traceability in this section. Cost estimation is a major subject in its own right, however, and we don't attempt a full introduction to the subject here.

A number of techniques and models for estimating development effort in general have been proposed, although large scale surveys have shown that no model is likely to be completely accurate all the time (Kemerer, 1987). Most cost estimation models, for maximum effectiveness, require careful calibration to a particular organisation's working culture and problem domain (Kemerer, 1987), since one type of development and business may have substantially different overheads and minimum quality thresholds than another. Data which can be useful for estimating costs includes:

- System size, clearly one of the most important determinants of total effort. This is a basic input used by estimation models such as the COCOMO and SLIM models (Kemerer, 1987). "Size" can be difficult to measure; some models use lines of code (LOC) as a metric. "Function points" was developed as an alternative metric (Kemerer, 1987) by Albrecht (1979); Albrecht and Gaffney (1983). Function points capture features such as the number of input transactions types and the number of reports to be output. This has the advantage over LOC that it is easy to determine at the design stage.
- Expert judgement, which is normally provided by someone with similar previous experience or detailed knowledge of the project at hand. In many cases this may require a mental "rehearsal" of the steps required to complete the task (Hughes, 1996). However, even a competent, experienced and well-informed developer on a project may forget (or be unaware of) small aspects that will need to be investigated.
- Analogy – comparing the new system to be developed to a previous, similar system (Hughes, 1996). Good practice should dictate that estimated and actual costs should be saved from the current project for use in future estimating tasks.
- A selection of "cost drivers" – that is, factors specific to the project that may affect costs. This can include: team size; the productivity/experience of personnel; project complexity; requirements volatility; tools available and so on. Many cost models use this type of data to "calibrate" a model to a particular working environment.

Finally, project "cost" can include the addition of some penalty should a task fail to be carried out to a sufficiently high standard. For example, there are likely to be either direct or indirect financial consequences of failing to ensure some appropriate level of testing coverage.

There's a difference between estimating the effort involved in traceability-related tasks, and estimating the actual costs. We need to understand the effort required to create and evolve trace links so that project schedules can be planned realistically. Traceability *costs*, however, offset the initial cost of that effort against estimated future savings and are useful for selecting appropriate traceability optimisations. We discuss both separately below.

4.1 Estimating Effort for Traceability

Total effort for traceability can be represented by two separate estimates. These can be adjusted independently. There should be an esimate for the time taken to create trace links at the outset, and another estimate for time taken to maintain the links as the system evolves. These two figures will each have an impact on the overall costs, since they will affect positively or negatively the effort required to perform traceeability-enabled activities later. For example, one option is not to perform any updates on links, or to update very infrequently, but this will require more manual checking when the trace data is used to support some activity. Whether this is a worthwhile trade-off will depend on the predicted frequency with which updates will be required.

There are not many studies showing how much time it takes to create or maintain trace links. Heindl and Biffl found that generating trace links from requirements to methods averaged around 45 min per requirement, whilst generating trace links from requirements to classes required a much lower effort, averaging 10 min per requirement (Heindl and Biffl, 2005). They do point out that this data represents time taken to capture trace links after the project's duration, and that estimates may be lower if trace links are generated during the project itself. Cleland-Huang et al. produced some estimates using a guide figure of 15 min to create a trace link (Cleland-Huang et al., 2004), although this is a hypothetical figure produced to illustrate costs for different traceability strategies.

As a more general figure, Heindl and Biffl estimate that tracing effort absorbs around 5% of the total project costs "as part of quality assurance activities" (Heindl and Biffl, 2005). Required documentation standards should already be costed in to the project cost before producing this estimate, as well as project size and duration. In addition, this figure could be refined as follows:

- volatile requirements will tend to increase traceability maintenance costs, whilst very stable requirements will reduce them. Well-informed stakeholders may be able to make some predictions about volatility in requirements to help with this.
- project duration and the estimated length of the maintenance period are likely to affect the cost estimates. A longer project will need to conduct more updates on trace links, so trace maintenance estimates should be increased for longer projects. However, greater savings can potentially be made, as the quantity of traceability-enabled activities tends to increase over time (for example, a longer project will see more change requests), so the effort saved can amalgamate.
- using automated techniques (such as information retrieval tools) is likely to reduce costs for trace creation, but increase costs associated with traceability-enabled activitites, because data may require some manual refinement at point of use. Alternatively, extra time could be spent when generating links ensure that the most important requirements are refined manually at the outset.
- an automated tool which favours recall over precision when creating or searching for trace links is likely to increase the estimate for trace-enabled activities, since more time will be spent on eliminating the false positives from trace queries. However, this will likely result in better overall accuracy, so there will be lower chances of incurring penalties associated with missing any potential links. Conversely, cost estimates for traceability-enabled activities may be lower for tools which favour precision, but there may be extra penalties (such as dissatisfied customers) incurred from dealing with any false negatives ("missed" trace

links) at a later date. This may be a cost-effective strategy, however, for products which require a rapid time-to-market.

- using techniques such as a simple traceability matrix (rather than, for example, adopting automated tools) can increase costs of traceability creation and maintenance (Cleland-Huang et al., 2004).
- more finely-grained trace data will increase trace creation and maintenance costs but is likely to reduce the time taken to complete traceability-enabled activities later. This cost could be altered by selectively varying the granularity of trace data (see Section 3.1). Increasing the granularity of trace data (from package-level to class-level, or from class-level to method-level) has been estimated to raise the required effort by 10% (Egyed, 2005; Egyed et al., 2005, 2007).
- updating trace links more regularly increases the estimate for maintenance but will decrease the estimate for later trace-enabled activities.

Finally, if the requirements are prioritised as suggested in Section 3.1 of this chapter, separate estimates may be needed for the differently-ranked groups of requirements. For example, low-priority requirements may have links auto-generated and not updated, whilst high-priority requirements may be entered manually and updated frequently.

4.2 Estimating Costs for Traceability

To come up with an estimate of the *cost* of traceability (as opposed to effort needed for scheduling), we include the cost of creating and maintaining the links, and factor in potential savings made by improving performance on future traceability-enabled activities, as was illustrated in Fig. 1. We can estimate this by looking at:

- the predicted number of times the activities supported by traceability are likely to be repeated. For example, approximately how many change requests can we expect, over what expected duration?
- estimates of the time taken to perform a task both with and without access to updated trace data. For example, a change impact analysis might be expected to take n person-hours when trace data is available, and m person-hours when it is not.

This should lead to two figures for estimated duration of later tasks, representing the effort required when supported and when not supported by trace data. The difference between the two gives the traceability cost benefit, which can be factored in (as a saving) to the final traceability cost.

Graphs produced (Cleland-Huang et al., 2004) make clear that the costs of trace creation and maintenance are generally only repaid after a period of time. This fact underpins a major problem associated with traceability: the cost of traceability is very visible up-front, whilst the financial savings made possible by trace data are not visible until a much later stage in development. However, from the traceability costing described here it should be possible to produce a prediction of when the project can expect to realise benefits from the trace data.

5 Traceability Strategies

Once estimates have been obtained, a traceability strategy can be produced in iterative stages, refining the trace quality as necessary for differently ranked requirements, until an optimal balance of cost and trace quality is achieved. A traceability strategy can make use of any techniques discussed so far, combining tactics such as:

- partitioning or ranking the system for traceability purposes, and selectively applying different traceability rules
- varying the granularity of trace links
- adopting tools where possible to create or search trace links, optimising the recall or precision as appropriate
- varying the coverage of the trace links
- varying the frequency with which trace links are maintained
- varying the application of automated link generation, and/or manual checks of the results

The costs and potential savings of different tactics can be estimated for the project; the tactics that are selected will be those that achieve the best balance between quality and cost. Developing the strategy will therefore be an iterative process (we describe this further in the next section).

Many projects – intentionally or not – adopt a strategy of generating links on an ad-hoc basis as and when required. Superficially, this strategy looks inexpensive, because it does not incur costs on the initial plans, whilst the savings incurred by completing traceability-enabled activities to a high standard are not visible. Several studies have suggested that this is in actuality not a very cost-effective strategy. Heindl and Biffl, for example, determined that, to be viable, the ad-hoc approach relies on infrequent requests for traceability data, a small project, and a high degree of domain and product knowledge among developers (Heindl and Biffl, 2005). In contrast, some studies have found that a strategy that mixes a number of varying approaches tend to produce a good return on investment. Cleland-Huang et al., for example, compared four potential trace strategies (Cleland-Huang et al., 2004), including:

- tracing and maintaining links using a simple matrix
- not maintaining links (it's assumed 15% of requirements change per year) and instead managing changes using "brute force analysis"
- tracing and maintaining links for critical requirements only, and manually tracing others

- the latter but with the addition of tools such as event-based traceability and information retrieval techniques (a "heterogenous" strategy)

The study concluded that the heterogenous strategy produced the lowest costs (for the provided case study).

6 Conducting a Practical Cost Analysis

In the rest of this chapter we suggest a simple method for conducting cost analysis to produce a traceability strategy that best meets project needs. The analysis consists of four steps, designed to fit in with existing traceability and software engineering practices:

1. Establishing traceability goals (this was discussed in Section 2.1 of this chapter)
2. Identifying the minimum data required to achieve the goals (discussed in Section 2.1)
3. Prioritising requirements and implementing traceability optimisations (discussed in Section 3).
4. Estimating effort needed to generate and maintain trace links, and refining choices from step (3) (cost estimation was discussed in Section 4).

The final two steps are envisaged as an iterative process of suggesting possible compromises and estimating which is likely to bring a reduction in traceability costs without significantly reducing traceability benefits.

The cost-analysis process is summarised in Fig. 2. The steps are designed to ensure that the maximum benefit can be achieved from the trace data (that the data is fit for purpose and high quality) whilst the total traceability cost is kept as low as possible. We discuss these steps in detail below, illustrating how the framework can be put to use by referring to the iTrust case study.

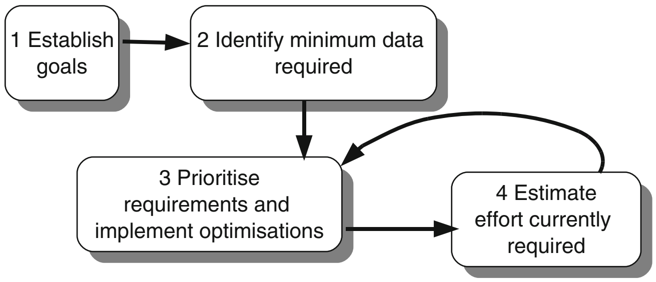
1 Establish goals
2 Identify minimum data required
3 Prioritise requirements and implement optimisations
4 Estimate effort currently required

Fig. 2 Steps in a simple traceability cost analysis

6.1 Establish Traceability Goals

Ann is a project manager in charge of a development team working on the iTrust development, tasked with implementing a cost-effective traceability strategy.

She begins by ensuring that appropriate questions are asked of the stakeholders, so that she has enough information to decide what the primary traceability goals must be. Analysis reveals that iTrust is likely to be a highly complex project, with numerous different possible roles users can adopt and potentially a large user base. It's technically challenging, and performance issues are likely to be taxing, given that the system is to support multiple simultaneous connections. Additionally iTrust is affected by strict regulations governing handling of medical data. As a complex and expensive system to develop, iTrust is expected to be in active use for a number of years (i.e., it will have a long maintenance period).

This information leads Ann to conclude that many changes to the system should be expected over its lifespan. These will be prompted by initial performance issues, general complexity, future changes to regulations and varied user demands and expectations. Change management should therefore be a key priority of the project. Strict regulation issues imply that validation and verification activities for both initial development and any future amendments will also be a key issue.

With so many potential sources of requirements and an expected long lifespan, Ann is also keen to record some basic requirements rationale for so that it's clear to customers as well as future development teams exactly why each requirement has been included. This will also help the team to identify and manage situations where users want conflicting features built into the system.

Once identified, the goals will dictate what data will be needed, how it may be managed and which optimisations will be possible. Traceability goals should be considered during requirements elicitation, when stakeholders are available to answer questions on anticipated future use. Some key questions to ask are discussed below. This is not an exhaustive list, but designed to be a practical starting point.

6.1.1 Change

Change management is a major factor underlying many traceability efforts. Answers to these questions are likely to indicate whether methods for improving change management will be a key traceability goal. Key questions here include:

- Do end users or stakeholders anticipate many requirements changes themselves?
- Are there strong expectations that the end system should be adaptable?

- Is this a rapidly developing market?
- Are there external laws or regulations affecting the project and how frequently do they change?
- How many non-functional requirements are there? These can be difficult to "design" and build into the system, and are often approached with iterative improvements to system quality.
- How big is the project/how many requirements does it implement/how long is the project duration? A larger project, or a project with a longer duration, is increasingly likely to see some requirements changes arise.
- How many stakeholders and/or users are involved with the system? Users are a major source of changes, as they discover bugs and make requests. A larger number of users tends to imply that more change requests and/or bug reports can be expected.

6.1.2 Design and Requirements Rationale

Some projects have discovered great benefit can be gained by capturing and storing the rationale behind decisions and deliberations. This type of information can be very helpful for handling major system extensions, refactoring, or preparing safety cases. Projects with high staff turnover are at particularly high risk of knowledge loss; trace data and traceability data structures can present a way to document product knowledge and minimise the loss. Even without staff turnover, developers and designers tend to forget over a period of time the original reasons why key decisions were taken. Taking this on board, Gotel and Finkelstein have suggested a system for recording the people behind major project decisions, particularly requirements rationale (Gotel and Finkelstein, 1995, 1997). Key questions to ask:

- Will the system require any kind of certification (such as a safety case)?
- How experienced are the developers with this type of development?
- Is the system expected to experience a long development/maintenance period?
- How complex is the system?
- What is the expected staff turnover? What is the recent rate of staff turnover in the same organisation?
- How stable (or otherwise) is the project expected to be? (see questions related to change above)

6.1.3 Requirements Management and Testing Coverage

Demonstrating that all requirements are implemented and tested is another very common use of trace data. As with issues relating to change management (above), the presence of many volatile requirements tends to suggest that testing coverage will need to be revisited frequently as requirements change. Key questions to ask include:

- How many requirements does the system implement, and are they stable?
- Are there many sources for requirement – e.g., external regulations or standards?
- Are there conflicts between stakeholders or end users?
- Are there many non-functional requirements? These tend to be cross-cutting (affecting many areas of the system) and are often implemented iteratively, presenting challenges for ensuring testing coverage.
- How many stakeholders and/or users are involved with the system?

6.2 Identifying the Minimum Data Needed

Now that the traceability goals have been identified, Ann can begin to identify the data she needs to capture as part of traceability efforts. Change management is a key issue: for this she will record trace links between requirements, design artifacts, code, test plans and also to requirements sources. These links will allow her to identify areas that will require revisiting should any given requirement be altered. Links to requirements sources (including people) will allow future teams to track down original decision-makers. In some cases, there may be good reasons why a requirement *shouldn't* be modified (e.g., regulations mandate it), so linking to requirements sources and/or rationale will allow developers to identify conflicts quickly and potentially avoid re-work.

Once traceability goals have been identified, subsequent steps become more straightforward, because the data needed is dictated by the goal. As we discussed in Section 2.1 of this chapter, we must aim to store the minimum of information needed. But we should also ensure that all data meets our minimum threshold of usability. Egyed et al. have suggested a list of questions that should be asked when planning a traceability strategy, such as (Egyed et al., 2005):

- is a perfect set of trace links necessary to achieve traceability goals?
- does an increase in quality of trace data justify the cost? (we discussed this point in Section 3)
- are false positives (i.e., the presence of trace links not correct) or false negatives (i.e., the absence of trace links which should be recorded) acceptable?
- what are the implications of errors in trace links?

We discussed in Section 3 how trace quality can be varied, and the potential impact on the ultimate return on investment. Answering the questions suggested by Egyed et al. should help to determine what quality level is an acceptable minimum for each proposed usage. This particular list can also be useful for projects employing automated tools to aid in the creation and/or maintenance of trace links, since tools can be calibrated to produce more complete (probably with more false positives) or more precise (probably with more false negative) sets of trace links.

We provide below a suggested list of link types to be considered. As before, this list is not exhaustive, but intended to be a practical starting point.

- Testing and requirements coverage goals commonly require (at least) trace links between requirements and code/test plans
- Re-use goals require links between requirements and code
- Requirements management also requires links between requirements and code – which can be used to identify where requirements conflict and to control scope creep
- Change management goals will require links between requirements and any other artifact that potentially requires updating as a result of a change. This will almost certainly include requirements, code and test plans. Links to use cases and design data may also be included
- Requirements rationale goals will need links between requirements and their sources
- Design rationale goals will need links between designs, decisions made and rationale behind them, and requirements and/or code affected.

It's tempting to assume that links between requirements and code components (for example) can be implemented by two stages: a link from requirements to design; and a second link from design to code. We might assume that our system could actually conduct two separate searches to find components linked to requirements: firstly, design decisions linked to the requirement; and then components linked to the design decision. This hypothetical set of trace links is illustrated in Fig. 3. However, a collection of links like this can make it impossible to conduct meaningful searches for components linked to requirements if there are many requirements and many components linked to a single design decision. Searching for components linked to Requirement 1 in Fig. 3 is likely to result in a large number of false positives, which will take time to weed out, because the design decision which is an intermediary is linked to many components which are not relevant to Requirement 1.

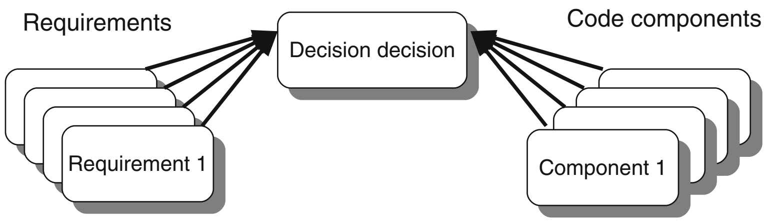

Fig. 3 Hypothetical set of trace links between design decision, requirements and componenets, exhibiting a disparity in granularity

This type of problem can arise anywhere where there is a disparity in granularity between notations used. These scenarios illustrate the need to consider at this stage how trace data will be searched, and to ensure that the quality of the links meets the minimum threshold for usefulness (we discussed this in Section 3 of this chapter).

6.3 Prioritising Requirements and Implementing Optimisations

Before Ann's team begin the task of creating and storing trace links Ann arranges a meeting for key stakeholders to agree a prioritisation of system requirements. This means determining what "value" each requirement has for the system. After discussion the stakeholders agree that, firstly, requirements imposed by medical data regulations are considered to be high priority. Failure to satisfy these requirements will impact on user trust in the system as well as raising the issue of legal penalties.

Responsiveness of the system and ease of use is also high priority; users will not accept the system if it cannot fit into pressured schedules of medical staff, and this is key to its financial success.

Finally, functional requirements are divided by key stakeholders into major (required) and minor (ideal) functions, and accorded different priorities as a result.

Now that the traceability goals, data and priorities are available, Ann can decide whether any optimisations are appropriate. The business impact of failing to meet medical data regulations is appreciated by all stakeholders for iTrust and the cost of assuring that high quality trace data is available for these requirements is accepted. Ann does not therefore need to employ traceability optimisations for tracing test coverage of these requirements. She adopts a finely-grained system here, creating trace links between requirements, individual methods in the code and test cases.

For other areas of functionality, some optimisations can be used. Requirements rationale is stored in free text. For this, Ann asks engineers to generate coarse trace links between requirements rationale and the requirements themselves. Any change to these requirements trigger an alert to the engineer to go and check for changes.

For functional requirements that were not accorded a high priority Ann instructs her team to adopt a coarsely-grained traceability approach, linking requirements to classes instead of methods to reduce the effort expended.

We discussed in Section 3.1 of this chapter some criteria for prioritising areas of the system so that different traceability techniques can be selectively applied.

6.4 Estimating Effort and Refining Choices

Ann's effort estimation overlaps with her work on identifying possible optimisations. She has calculated the effort required in tracing legally-mandated requirements' testing coverage and justified the cost. However, her estimates of the effort involved in tracing the rest of the system seem high.

Instead, Ann estimates the cost required to adopt an automated tool to generate links between requirements and documentation for low-priority areas instead of asking the team to create the links. She re-calculates the estimated effort needed to generate the links (very low now that a tool is in place) and to carry out later traceability-enabled activities (a little higher than before, since tools are not perfect and links will need some filtering from the engineers). The short-term trace creation costs are low when using the tool, but she estimates that within 18 months she will start to see medium- and long-term saving in terms of reduced effort for coping with change requests later. Ann refines her previous choice and adopts the automated tool for generating low-priority links.

The last two steps – selecting optimisations and estimating effort – are likely to become an iterative cycle as project planners recalculate the effort required and select acceptable compromises in order to bring the total costs into acceptable boundaries. Cost estimations should include both the initial cost of creating trace links as well as estimating the effort saved on carrying out later traceability-enabled activities, and a realistic estimate of the expected time-scale before the traceability system will start to return a saving in effort.

7 Conclusions

In this chapter we have introduced some key issues behind cost-benefits of traceability. In general the return on investment in traceability is maximised by keeping the quality of the trace data high in areas where it is most needed, and the cost of generating and maintaining trace data low. By "quality", we mean the recall, precision, granularity and suitability of the data for its intended purposes. There are a number of techniques which may be adopted to reduce the cost of traceability with only a minimal impact on quality. A flexible, "heterogeneous" approach to traceability is likely to achieve the best balance between achieving traceability benefits and controlling the cost. This strategy involves selecting the best available technique or strategy to achieve the current goal. Prioritising requirements is an important step, as it allows a mixture of techniques and optimisations to be selectively applied where they are most appropriate, ensuring the best quality trace data is applied where needed.

We've suggested a very simple process for analysing the cost benefit of traceability and selecting an appropriate strategy. Any cost analysis of traceability must ensure that the costs of implementing traceability are offset by the cost savings which can be incurred at a later stage when carrying out traceability-enabled activities. Projects with longer durations and/or maintenance periods, for example, might incur relatively high costs for trace creation, but this should be offset by the reduction of effort required for repeated change requests over many years.

References

Albrecht, A.J.: Measuring application development productivity. In: Proceedings, IBM Applications Development Symposium, pp. 14–17. Monterey, CA (1979, October)

Albrecht, A.J., Gaffney, J.E., Jr.: Software function, source lines of code, and development effort prediction: A software science validation. IEEE Trans. Softw. Eng. **SE-9**, 639–648 (1983, November)

Arisholm, E., Briand, L.C., Føyen, A.: Dynamic coupling measurement for object-oriented software. IEEE Trans. Softw. Eng. **30**(8), 491–506 (2004)

Basili, V.R., Briand, L.C., Melo, W.L.: A validation of object-oriented design metrics as quality indicators. IEEE Trans. Softw. Eng. **22**(10), 751–761 (1996)

Boehm, B., Huang, L.G.: Value-based software engineering: A case study. IEEE Softw. **36**(3), 33–41 (2006)

Briand, L.C., Wüst, J., Lounis, H.: Using coupling measurement for impact analysis in object-oriented systems. In: Proceedings. IEEE International Conference on Software Maintenance, (ICSM '99), pp. 475–482. ICSM, Oxford, England (1999)

Chaumun, M. Ajmal, K., Hind, K., Rudolf K., Lustman, F.: Département IRO, and Université De Montréal. A change impact model for changeability assessment in object-oriented software systems. In: Proceedings of the Third Euromicro Working Conference on Software Maintenance and Reengineering, pp. 130–138 (1999)

Cleland-Huang, J.: Just enough requirements traceability. In: 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), pp. 41–42 (2006, September)

Cleland-Huang, J., Change, C.K., Christensen, M.: Event-based traceability for managing evolutionary change. IEEE Trans. Softw. Eng. **29**(9), 796–810 (2003)

Cleland-Huang, J., Zemont, G., Lukasik, W.: A heterogeneous solution for improving the return on investment of requirements traceability. In: 12th IEEE International Conference on Requirements Engineering (RE 2004), pp. 230–239 (2004, September)

Egyed, A.: Determining the cost-quality trade-off for automated software traceability. ASE 2005:360–363 (2005)

Egyed, A., Biffl, S., Heindl, M., Grünbacher, P.: A value-based approach for understanding cost-benefit trade-offs during automated software traceability. In: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, TEFSE '05, pp. 2–7. ACM, New York, NY. ISBN 1-59593-243-7 (2005)

Egyed, A., Grünbacher, P., Heindl, M., Biffl, S.: Value-based requirements traceability: Lessons learned. In: 15th IEEE International Requirements Engineering Conference, RE 2007, pp. 115–118 (2007)

Gotel, O., Finkelstein, A.: Contribution structures. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering, pp. 100–107 (1995, March)

Gotel, O., Finkelstein, A.: Extended requirements traceability: Results of an industrial case study. In: International Symposium on Requirements Engineering (RE97), pp. 169–178. Society Press, Annapolis, MD (1997)

Han, Ah.-R., Jeon, S.-Uk., Bae, D.-H., Hong, J.-E.: Behavioral dependency measurement for change-proneness prediction in UML 2.0 design models. In: COMPSAC '08: Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference, pp. 76–83. IEEE Computer Society, Washington, DC. ISBN 978-0-7695-3262-2 (2008)

Heindl, M., Biffl, S.. A case study on value-based requirements tracing. In: Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 60–69. ISBN 1-59593-014-0 (2005)

Huang, L., Boehm, B.: How much software quality investment is enough: A value-based approach. IEEE Softw. **23**(5), 88–95 (2006, September/October)

Hughes, R.T.: Expert judgement as an estimating method. Inform. Softw. Technol. **28**, 67–75 (1996)

Ingram, C., Riddle, S.: Linking software design metrics to component change-proneness. In: WeTSOM 2011 – 2nd International Workshop on Emerging Trends in Software Metrics (WeTSOM 2011). Honolulu, Hawaii (2011)

Jarke, M.: Requirements tracing. Commun. ACM **41**(12), 32–36 (1998, December)

Kemerer, C.F:. An empirical validation of software cost estimation models. Commun. ACM **30**(5), 416–429 (1987)

Li, W., Henry, S.: Object Oriented Metrics Which Predict Maintainability. Technical Report, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia (1993, February)

Ramesh, B., Powers, T., Stubbs, C.: Implementing requirements traceability: A case study. In: Proceedings of the 2nd IEEE International Symposium on Requirements Engineering, pp. 89–95 (1995, March).

Ratzinger, J., Sigmund, T., Vorburger, P., Gall, H.C.: Mining software evolution to predict refactoring. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), pp. 354–363. IEEE Computer Society, Madrid, Spain (2007)

Wilkie, F.G., Kitchenham, B.A.: Coupling measures and change ripples in C++ application software. J. Syst. Softw. **52**(2–3), 157–164 (2000)