Exploring a Bayesian and Linear approach to requirements traceability

Inah Omoronyia, Guttorm Sindre, Tor Stålhane
Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway
{inah1, Guttorm.Sindre, Tor.Stalhane}@idi.ntnu.no

Abstract

For large software projects it is important to have some traceability between artefacts from different phases (e.g., requirements, designs, code), and between artefacts and the involved developers. This is especially critical during maintenance, when people working on the software may be different from the original developers and therefore have a harder struggle to understand the artefacts and the consequences of changes. However, if the capturing of traceability information during the project is felt as laborious to the original developers, they will often be sloppy in registering the relevant traceability links so that the information is incomplete. This makes automated tool-based collection of traceability links a tempting alternative, but this has the opposite challenge of generating too many potential trace relationships, not all of which are equally relevant. A key issue is therefore how to rank such auto-generated trace relationships. This paper presents two approaches for such a ranking: a Bayesian technique and a linear inference technique. Both techniques depend on the interaction event trails left behind by collaborating developers while working within a development tool. The advantage of our approach is that it can be used to provide traceability insights that are contextual and would have been much more difficult to capture manually. The outcome of a preliminary study suggest the advantage of the linear approach, we also explore the challenges and potentials of the two techniques. Finally we present some key lessons learnt during this research.

Key words: requirements traceability, maintenance, collaborative work
1. Introduction

Writing use cases to capture functional requirements has become a common practice during software development. While traceability between use cases and the artefacts executing them has been shown to be beneficial [1], the process of recovering and maintaining traceability among use cases, software artefacts of different types and associated system developers still remains a challenging, manual and laborious task [15].

As an example consider software developer D who needs to make some changes to a software product due to a requirements change, e.g., the customers have expressed a wish for altering a certain use case. Unless the task is trivial, there are a number of questions that D might wish to have answered. Which code artefacts (e.g., classes) are involved in implementing the use case? If the class C is modified to fulfill the requirements change, what other use cases might also be affected by this? Who were involved in writing the use case? Who were involved in writing the class C and other classes that are relevant for the use case? In this research, we refer to these requests as \textit{traceability information needs}, defined as the traceability links between entity instances; use cases, developers and code artefacts, that a stakeholder is required to know about in order to gain better understanding in carrying out some specified tasks.

In satisfying such information needs, it is essential to know when a traceability link between entity instances (use cases/system features, code artefacts and developers) is relevant or not. This is because, understanding of the extent of relevance of traceability links between entity instances in a project to a selected work context, is critical to satisfying this viewpoint of traceability information needs. But classifying an entity instance as relevant or not relevant in a traceability relation is itself demanding as it depends on the stakeholder with an information need. For instance, a project manager will be interested in knowing the relevant entities that require or are consuming more time to be realised. On the other hand, a software tester is interested in knowing the relevant entities that have been affected by recent bug fixes. Finally, a software developer is interested in knowing the relevant entities that influence their scope of work.

For a nontrivial project, there is an additional demand which is determining when a traceability relation between entity instances are relevant. This demand results from the rather complex dependencies amongst entity instances. Within such complex dependency settings, one will expect that
a code artefact can be associated with a number of developers and used to achieve a number of project tasks such as use cases and bug fixes. Similarly, a project use case can be associated with a number of collaborating developers and a number of code artefacts. Finally, it is expected that a developer can be working on a number of project use cases and using a number of code artefacts to achieve the use cases [17, 26]. Such dependencies simply increase the number of entities that can be used to describe a work context and hence make it more challenging in determining the relevance of a traceability relation.

The goal of this research is to provide insights on mechanisms for determining the relevance of traceability links between entities for a scenario where a software developer can work on multiple code artefacts to achieve different system features or use cases. Ideally the data needed to determine such relevance might have been explicitly captured during the project, in general which developer contributes to which artefacts, and which artefacts are related to each other. But this rarely happens - at best such traceability information is incomplete and outdated because many developers find it too time-consuming to update the traceability information for every little change they do to the software artefacts.

An alternative approach is to harvest data required for the determination of an entity’s relevance automatically when developers interact with development tools. The captured data is then subsequently used to provide context based insight on the extent of relevance of traceability links. This paper explores the viability of Bayesian and linear techniques for estimating the relevance of traceability links. We also explore the challenges of capturing the interaction events during project development. This paper focuses on traceability relations between three core entities: use cases, developers and code artefacts.

In the Bayesian technique, we view requirements traceability as a statistical inference problem. Here the traceability relations generated amongst use cases, artefacts and developers are modelled as a set of related nodes in a Bayesian Belief Network (BBN). The relative relevance of entities to a selected trace link is then determined based on the posterior probability of relevance of a selected node given a number of other entities as the evidence nodes. In the linear technique, we approach requirements traceability as a mathematically linear problem. In the linear approach, the relevance of entities associated with a selected work context are accumulatively derived based on attributes such as the type of interaction events the entity
has been involved in, and a size measure of other entities the selected instance has exacted its presence on (or the number of other entities to which the selected instance serves some usefulness). Although the outcome of a preliminary study using advance level Masters/Honours software engineering students suggests the advantage for the linear approach, we also explore the challenges and potentials of the two techniques.

Section 2 provides details of how requirements traceability can be modelled as a Bayesian belief network. The main points discussed in this section are methods for building a network structure (2.1), associating local probability distributions to entities based on involving interaction events (2.2), and inferring the relative relevance of entities associated with a traceability link by calculating their posterior probabilities and ranking entities according to their posteriors (2.3). Section 3 presents the linear accumulative relevance model. Section 4 details an Eclipse plugin implementation of both approaches and demonstrates that they are achievable. Section 5 presents an evaluation study while section 5.1 describes the threats to the validity of the study. Section 6 discusses the outcome of the study and the lessons we have learnt in this research. The review of related work is in section 7, whereupon section 8 concludes the paper and suggests further work.

2. Modelling traceability as belief networks

Belief networks provide statistical means to deal with processes that are prone to uncertainty, vagueness and error conditions. Diagnostic, decision support, preventive maintenance and information retrieval processes are typical examples of problems that have been modelled based on belief networks. The fundamental theory behind these networks is Bayes’ theorem as represented in equation 1.

\[
P(A|B) = \frac{P(A, B)}{P(B)}
\]  (1)

where \( P(A, B) = P(B|A)P(A) \) based on the chain rule.

\( P(A) \) is the prior probability or marginal probability of a vector \( A \) given that it does not take into account any information about the vector \( B \). \( P(A|B) \) is the conditional probability of \( A \), given \( B \). It is also called the posterior probability because it depends upon the specified value of \( B \). In this paper, \( B \) is referred to as the evidence entity while \( A \) is the query entity. \( P(B|A) \) is the conditional probability of \( B \) given \( A \). \( P(B) \) is the prior or marginal probability of \( B \), and acts as a normalizing constant [12].
Belief networks provide a graphical means for explicitly representing interdependencies among the variables of a joint probability distribution. The probability distribution is represented through a directed acyclic graph whose nodes represent the random variables of the distribution. Thus, two random variables, \(A\) and \(B\), are represented in a Bayesian network as two nodes in a directed graph. An edge directed from \(A\) to \(B\) represents the influence of the node \(A\), the parent node, on the node \(B\), the child node. Each node stores a local probability distribution table (PDT) \(P(Node|Parents(Node))\) to quantify the effects or influence of parents on child. Finally, we make the local Markov assumption: nodes are independent of its non-descendants given its parents [20]. Thus, assuming a first order Markov process and given a network structure as described, the joint probability distribution over a set of random nodes \((A_1, A_2, \ldots, A_n)\) is given as:

\[
P(A_1, A_2, \ldots, A_n) = \prod_{i=1}^{n} P(A_i|Parents(A_i)) \tag{2}
\]

To enable the modelling of requirements traceability as a belief network we present a realistic system development scenario as described below:

Bill, Amy and Ruben are members of a team collaborating to develop an online cinema ticketing system called TickX. There are two front-end use cases required to accomplish TickX: Purchase Tickets and Browse Movies. In addition, there will be some use cases for system administrators that are not included here. Furthermore, a number of code artefacts are being developed to realise TickX, including Ticket.java, Customer.java, Account.java, Booking.java, Movie.java, MovieCatalog.java, and Cinema.java.

While Amy and Bill have been collaborating to implement the Purchase Tickets use case, Ruben has been responsible for the Browse Movies use case. The following interaction trails were observed as these collaborators worked on their associated use cases:

- While Amy was collaborating on Purchase Tickets she created and updated the Account.java and Customer.java code artefacts. She viewed and updated Booking.java a number of times. She also viewed MovieCatalog.java and Cinema.java.

- In the initial phase of Bill’s collaboration on the Purchase Tickets use case, he viewed the Account.java and MovieCatalog.java code artefacts. This was subsequently followed by his creation and update of the Ticket.java and Booking.java code artefacts.
Ruben’s implementation of the Browse Movies use case involved the creation and further updating of the MovieCatalog.java, Cinema.java and Movie.java code artefacts. Ruben also viewed Ticket.java a number of times.

In this paper, developers within the collaboration space are the team members that work within a project context - in this case Amy, Bill and Ruben. A use case is a coherent unit of functionality provided by a system, a subsystem, or a class that the project aims to achieve. Finally, artefacts are project components such as software modules and documents that are manipulated by developers (examples include every code artefact in TickX).

From the described scenario and taking into account the evolving developer activities, requirements traceability can be modelled as a belief network from both the viewpoints of use case relations to artefacts and to developers as shown in figure 1, labelled a and b. Each node $C_i$ models a use case, the node $A_j$ models the artefacts used to execute use cases, and the $D_k$ nodes models the system developers that implement use cases using artefacts. In this research, each of these viewpoints is used to define the work context perspective for a defined traceability link. There exist many-to-many relationships amongst these entities while same entity type relationships are not enforced. For example, the Purchase Tickets use case $C_2$ has a trace link to Bill and Amy in figure 1b and with a number of code artefacts in 1a. Other symmetric trace viewpoints exist; for example, MovieCatalog.java has a trace link to the three collaborators as well as the two collaboration use cases. This generates belief networks from the viewpoints of artefact relations to use cases as well as to the system developers.

Each node in figure 1 is associated with a local PDT. Use cases are associated with marginal probability tables since they have no parent as demonstrated for the use cases $C_1$ and $C_2$. Artefacts and developers are associated with conditional probability tables assuming that their relevance is determined by the use cases they implement. All nodes assume binary states in the network; thus, they are either relevant (denoted by $R$) or not relevant (denoted by $\neg R$). The PDT for each node is listed beside the node as shown in figure 1a. The joint probability distribution of the nodes at time $t$ in the network, is then used to determine the extent of relevance of artefacts or developers associated with use case traceability links or vice versa. The joint probability distribution of the belief network in figure 1a and b considering independence assumption (e.g $A_1$ is independent of its non-descendant such
as $C_1$ given $C_2$ etc.) at time $t$ is as shown in equations 3 and 4 respectively.

$$P_t(C_1, C_2, A_1, A_2, A_3, A_4, A_5, A_6, A_7) = P_t(C_1)P_t(C_2)P_t(A_1|C_2)P_t(A_2|C_2)$$

$$P_t(A_3|C_2, C_1)P_t(A_4|C_2)P_t(A_5|C_2)P_t(A_6|C_2)P_t(A_7|C_2, C_1)$$

$$P_t(C_1, C_2, D_1, D_2, D_3) = P_t(C_1)P_t(C_2)P_t(D_1|C_2)P_t(D_2|C_2)P_t(D_3|C_1)$$

(3)

As an example, consider a situation where we want to determine the probability that the artefact $A_1$ is relevant at time $t$ given that $C_2$ is the use case of interest and that $C_2$ has been achieved using a number of other artefacts. In this case, $C_2$ acts as the evidence node, that is the state of $C_2$ is known ($C_2 = R$). Based on Bayes’ theorem, the posterior probability of relevance of $A_1$ to $C_2$ is the probability that $A_1 = R$ given $C_2 = R$ as shown in equation 5.

$$P_t(A_1 = R|C_2 = R) = \frac{P_t(A_1 = R, C_2 = R)}{P_t(C_2 = R)}$$

(5)
Using the joint probability stated in equation 3 we got

\[ P_t(A_1 = R, C_2 = R) = \sum_{i \in \{R, \neg R\}} P_t(C_1 = i)P_t(C_2 = R)P_t(A_1 = R|C_2 = R) \]
\[ \times P_t(A_2 = i|C_2 = R)P_t(A_3 = i|C_2 = R, C_1 = i) \]
\[ \times P_t(A_4 = i|C_2 = R)P_t(A_5 = i|C_2 = R) \]
\[ \times P_t(A_6 = i|C_2 = R, C_1 = i) \]
\[ \times P_t(A_7 = i|C_2 = R, C_1 = i) \]
\[ (6) \]

and

\[ P_t(C_2 = R) = \sum_{i \in \{R, \neg R\}} P_t(C_1 = i)P_t(C_2 = R)P_t(A_1 = i|C_2 = R) \]
\[ \times P_t(A_2 = i|C_2 = R)P_t(A_3 = i|C_2 = R, C_1 = i) \]
\[ \times P_t(A_4 = i|C_2 = R)P_t(A_5 = i|C_2 = R) \]
\[ \times P_t(A_6 = i|C_2 = R, C_1 = i) \]
\[ \times P_t(A_7 = i|C_2 = R, C_1 = i) \]
\[ (7) \]

Similar probabilities as shown in equation 5 can also be derived for other artefacts including \(A_2, A_3, A_4, A_6\) and \(A_7\) and developers \(D_1\) and \(D_2\) that have been associated with \(C_2\) at a specified time. The ranking of these posterior probabilities can then be used to derive the relative relevance of artefacts and developers that have worked on \(C_2\) at time \(t\). On the whole, suppose a stakeholder identifies a use case of interest, then the retrieval of relevant artefacts and developers traceability links associated with the selected use case can be divided into four main steps:

1. Build a network structure (based on the observed dataset) representing the use case from separate viewpoints of related artefacts and developers.
2. Associate estimated local PDT to each node in the network.
3. Calculate the posterior probability of relevance of artefacts and developers associated with the identified use case of interest.
4. Rank the artefacts and developers according to the posteriors.

The symmetric variation of these steps is implied. For instance, given that a stakeholder’s interest is from an artefacts perspective, then step 1 will involve building a network representing the artefact from separate viewpoints of related developers and use cases. Two obvious challenges can be seen in achieving the listed steps. The first is a modality for the specification of
the belief network structure as described above. This would also involve making explicit the dataset underlying how the network structure has been formed (both the network structure and the underlying dataset are critical to the accurate determination of PDT). The second is the estimation of local PDT’s. These estimates can then be used to make inference on the relevance of entities based on their posterior probabilities.

2.1. Deriving underlying dataset defining trace belief network structure from developer event trails

The modality for making explicit the dataset underlying a belief network structure is achieved by monitoring the core interactions of developers as they carry out actions on artefacts within a specified development tool to realise use cases. The framework needed to enable this step had been specified and implemented in previous work [31]. The core interactions monitored are creates, views, updates and deletes. The following assertions were made about these interaction events:

- A create event causes the manifestation of a tangible artefact within a collaboration space, adding a node to the graph structure.
- An update event affects the state of an entity instance directly. The update delta is defined as the absolute difference in the number of characters of the artefact before and after the event.
- A view event indirectly affects the state of entity instances as it can enhance a developer’s understanding in order to update this or other artefacts.
- A delete event transforms an entity to an intangible state, where it is unable to receive any further events. This removes the node from the graph structure so that the entity is not involved in subsequent relevance calculations.

During collaboration processes it is expected that the core interaction event types will be associated with different levels of importance. For instance, an interaction event where a developer created an artefact could be considered more important than an event where a developer viewed the same artefact. To further investigate the properties of interaction events, and their weighted influence on the relevance of entities in a collaboration space, we performed a study of CVS records associated with real development projects. These records were derived from a group project software engineering class and using open source Eclipse IDE technology and tools projects. CVS
repositories of 200 artefacts from a combination of the Eclipse Communication Framework (ECF), Dash, Mylar, Equinox, and Eclipse Modelling Framework (EMF) open source projects were analysed. Only artefact check-ins with version repositories associated with more than one project member were considered. The results showed that for the CVS checked in versions that were analysed, collaborators associated with the first artefact checked in were also associated with 49.6% of subsequent checked in versions. This result implies that, assuming the collaborator associated with the first checked in version is the artefact creator, the creator of an artefact is associated with 49.6% of subsequent updates. From a collaborative relevance standpoint, this strongly suggests that granular interaction types that have a direct effect on the state of entity instances, such as create and update, can be used to derive relevance orderings.

The outcome of analysing CVS records suggests that although a create interaction event occurs once in the lifetime of an entity instance, the creator of an entity, in a significant number of cases (49.6%), is subsequently associated with the greater number of further interaction types. This makes the create event particularly important relative to other interaction types. Furthermore, while it is expected that view events can enhance understanding of project related processes, studies conducted by Zou and Godfrey [37] also suggested that cases of random view events that are irrelevant to an on-going development work can occur. In weighting the influence of view events on the relevance of entities in a collaboration space, it is important that the effects of such irregularities are inhibited.

Based on the insight obtained from the analysis of view, update and create interaction types, initial influence-based weightings as shown in Table 1 are assigned to each interaction event type. Highest weightings are assigned to a create event while the least was assigned to a view. Related work by Fritz et al. [10] has also suggested importance of create or authors of code artefacts. The weights in table 1 were further refined based on feedback from usage statistics generated while implementing subsequent revisions of the prototype discussed in section 4.

Assuming that the interaction event trails shown in figure 2 were the events used to achieve TickX described previously, any selected time-point corresponds to at least one event associated with a developer, a use case and an artefact. For instance, at time 1, a create event associated with Account.java was executed by Amy while working on the Purchase Tickets use case. Similarly, time 7 has two events: Ruben updated Cinema.java (update
Table 1: Interaction type weightings

<table>
<thead>
<tr>
<th>Interaction Type</th>
<th>View</th>
<th>Update</th>
<th>Create</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weighting factor</td>
<td>0.001</td>
<td>0.0001*</td>
<td>0.01</td>
</tr>
<tr>
<td>δ - Absolute update delta (magnitude of the update)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

delta 50) while working on Browse Movies, and Bill viewed Account.java as he worked on Purchase Tickets.

As shown in figure 2, the achievement of Browse Movies use case ($C_1$) using the artefact MovieCatalog.java ($A_6$) has generated one create event and a total update delta of 650. The conversion of these events based on the weights shown in table 1 gives 0.075 as the equivalent number of interaction events, (i.e $(0.01 \times 1) + (0.0001 \times 650)$). Similarly, Purchase Tickets ($C_2$) generated a total of three view events and a total update delta of 60 using the same artefact $A_6$ giving 0.009 on conversion. These values can further be normalised to 75 and 9 interaction event units respectively by multiplying by 100. Figure 3 represents the belief network of use case to artefacts trace links annotated with normalised event units based on the event trails shown in figure 2.
2.2. Trace network local PDT estimation by learning BBN parameters

Given a belief network structure of nodes and some training dataset of interaction events, the goal of learning is to find the values of each local PDT that maximises the likelihood of a training dataset. This is essentially a belief network learning problem for a case where the network structure is known and data fully observed. The proposed learning method for such problem is the Maximum-likelihood estimation (MLE) - a method to identify the probability distribution that makes the observed data most likely [29].

The calculation of MLE requires that a probabilistic model that fits the data distribution is known and the Probability Density Function (PDF) determined. The collaboration scenario described for TickX project in section 2 can be modelled as a series of Bernoulli trials (binomial experiments) assuming the following properties are satisfied:

- The experiment consists of $n$ repeated trials. This corresponds to the $n$ number of event units that are traced from a set of parent entities represented as $\text{parent}\{e_{p1}, e_{p2}, \ldots, e_{pn}\}$ to its set of children entities represented as $\text{children}\{e_{c1}, e_{c2}, \ldots, e_{cn}\}$. For instance, from figure 3 the set $\text{parent}\{C_1, C_2\}$ can be traced to the set $\text{children}\{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9\}$.
\( A_5, A_6, A_7 \) and \( n = 252 \). Similarly, the parent\( \{C_1\} \) can be traced to the children\( \{A_3, A_5, A_6, A_7\} \) and \( n = 110 \).

- Each trial can result in one of just two possible outcomes \( (R, \neg R) \) for members of a selected set. For any single trial, only one member from either parent and children is relevant \( (R) \) while every other member of the two sets are not relevant \( (\neg R) \). For instance, based on the events used to achieve TickX shown in figure 2, assuming time 13 is the trial event of interest and that use case to artefact trace links is the focus of analysis (figure 1a): Then from the parent set, \( C_2 = R \) and \( C_2 = \neg R \). Also, from the children set, \( A_6 = R \) and \( A_1, A_2, A_3, A_4, A_5, A_7 = \neg R \).

On the whole, any single event generated from a use case can only be traced to one relevant artefact. Other symmetric views also hold.

- The probability of relevance, denoted by \( P_{rel} \), is the same on every trial.

- The trials are independent; that is, the outcome on one trial does not affect the outcome on other trials.

The binomial probability distribution or the PDF that a binomial experiment results in exactly \( x \) relevance instances for an entity \( e \) denoted as \( x_e \) is as stated in equation 8. \( P_{rel} \) is the probability of relevance of \( e \) on any one trial. To determine \( x_e \) and \( n \) for conditional entities (or entities associated with local conditional probabilities such as \( A_1, A_1, ..., A_7 \)), the attributes of the relationship between \( e \) and members of its parent set are analysed. For instance, the number of relevance instances for \( A_1 \) denoted as \( x_{A_1} \) is 56 and \( n = 142 \). Given that \( A_6 \) parents consist of \( C_1 \) and \( C_2 \), assuming only \( C_1 \) is the relevant parent of interest \( (i.e \ C_1 = R, C_2 = \neg R \) and \( A_6 = R \)) then \( x_{A_6} = 75 \) and \( n = 110 \). Conversely, if only \( C_2 \) is the relevant parent of interest \( (i.e \ C_1 = \neg R, C_2 = R \) and \( A_6 = R \)) then \( x_{A_6} = 9 \) and \( n = 142 \). Finally if \( C_1 \) and \( C_1 \) are both the relevant parents of interest \( (i.e \ C_1 = R, C_2 = R \) and \( A_6 = R \)) then \( x_{A_6} = 84 \) and \( n = 252 \).

For determination of \( x_e \) and \( n \) for marginal entities (or entities associated with local marginal probabilities such as \( C_1 \) and \( C_2 \)), it is assumed that the events resulting from use cases in a shared collaboration workspace are aimed at achieving the goal of a relevant software project of interest: \( proj \). Thus, given that \( C_1 \) and \( C_2 \) are marginally independent for the project TickX, if analysing for \( C_1 \) \( (i.e \ C_1 = R \) and \( TickX = R \)) then \( x_{C_1} = 110 \) and \( n = 252 \).
Figure 4: Plot of \((0 \leq P_{rel} \leq 1)\) versus Likelihood \(L(P_{rel|A_6}|n = 110, x_{A_6} = 75)\) for \(C_1 = R\), \(C_2 = \neg R\) and \(A_6 = R\).

Similarly, if analysing for \(C_2\) (i.e. \(C_2 = R\) and \(TickX = R\)) then \(x_{C_2} = 142\) and \(n = 252\).

\[
f(x_e|n, P_{rel}) = \binom{n}{x_e} P_{rel}^{x_e} (1 - P_{rel})^{n - x_e}
\]

\((0 \leq P_{rel} \leq 1; x_e = 0, 1, \ldots, n)\)  

(8)

Since the interaction events data has already been observed and formally represented (see section 2.1), we are faced with the inverse problem: Given the observed interaction event data and a probabilistic model that fits the data distribution, find the one PDF, among all the probability densities that the model prescribes that is most likely to have produced the events data. This inverse problem is solved by reversing the roles of the data vector \(x_e\) and the probability vector \(P_{rel}\). This process generates an inverse function defined as the likelihood function stated as \(L(P_{rel|x_e}) = f(x_e|P_{rel})\). Thus \(L(P_{rel|x_e})\) represents the likelihood of \(P_{rel}\) given the observed data \(x_e\).

For example, based on equation 8 the likelihood function for \(x_{A_6} = 75\) and \(n = 110\), given that \(C_1 = R\), \(C_2 = \neg R\) and \(A_6 = R\) is as shown in equation 9. Figure 4 is the graph shape of the likelihood functions for equation 9.

\[
L(P_{rel|A_6}|n = 110, x_{A_6} = 75) = f(x_{A_6} = 75|n = 110, P_{rel|A_6})
\]

\[= \frac{110!}{75!35!} P_{rel|A_6}^{75} (1 - P_{rel|A_6})^{35}\]

(9)

Finally, the likelihood function for \(x_{A_6} = 84\) and \(n = 252\), given that \(C_1 = R\), \(C_2 = R\) and \(A_6 = R\) is as shown in equation 11. Figure 6 is the graph shape of the likelihood functions for equation 11.
Figure 5: Plot of \(0 \leq P_{rel_{A6}} \leq 1\) versus Likelihood \(L(P_{rel_{A6}} | n = 142, x_{A6} = 9)\) for \(C_1 = \neg R, C_2 = R\) and \(A_6 = R\). Also, the likelihood function for \(x_{A6} = 9\) and \(n = 142\), given that \(C_1 = \neg R, C_2 = R\) and \(A_6 = R\) is as shown in equation 10. Figure 5 is the graph shape of the likelihood functions for equation 10.

\[
L(P_{rel_{A6}} | n = 142, x_{A6} = 9) = f(x_{A6} = 9 | n = 142, P_{rel_{A6}}) = \frac{142!}{9!133!} P_{rel_{A6}}^9 (1 - P_{rel_{A6}})^{133}
\]

Figure 6: Plot of \(0 \leq P_{rel_{A6}} \leq 1\) versus Likelihood \(L(P_{rel_{A6}} | n = 252, x_{A6} = 84)\) for \(C_1 = R, C_2 = R\) and \(A_6 = R\).
\[ L(P_{relA_6}|n = 252, x_{A_6} = 84) = f(x_{A_6} = 84|n = 252, P_{relA_6}) \]
\[ = \frac{252!}{84!168!} P_{relA_6}^{84} (1 - P_{relA_6})^{168} \]  

Once the interaction event data have been collected and the likelihood function of a model given the data is determined, one is in a position to make statistical inference about the local probability distribution that underlies the data. Given that different likelihood function values index different probability distribution values as shown in figures 4, 5 and 6, the aim of MLE is to find the probability distribution that maximises the likelihood function. This probability distribution value is denoted as \( P_{MLE} \). For example, in figure 4 involving local probability for \( C_1 = R, C_2 = \neg R \) and \( A_6 = R \), the MLE estimate is \( P_{MLE}^{A_6} = 0.68 \) for which the maximum likelihood value is \( L(P_{relA_6}|n = 110, x_{A_6} = 75) = 0.08 \). Similarly, in figure 5 involving local probability for \( C_1 = \neg R, C_2 = R \) and \( A_6 = R \), the MLE estimate is \( P_{MLE}^{A_6} = 0.06 \) for which the maximum likelihood value is \( L(P_{relA_6}|n = 142, x_{A_6} = 9) = 0.138 \). Finally, in figure 6 involving local probability for \( C_1 = R, C_2 = R \) and \( A_6 = R \), the MLE estimate is \( P_{MLE}^{A_6} = 0.34 \) for which the maximum likelihood value is \( L(P_{relA_6}|n = 252, x_{A_6} = 84) = 0.053 \).

For computational convenience, MLE is obtained by maximising the log-likelihood function, \( ln L(P_{relA_6}|x) \). That is \( ln L \equiv l \). Furthermore, assuming the log-likelihood function is differentiable if \( P_{MLE} \) exist, then its partial differential is equal to zero as shown in equation 12.

\[ \frac{\partial l(P_{relA_6}|x_e)}{\partial P_{relA_6}^{x_e}} = 0 \text{ at } P_{relA_6}^{x_e} = P_{relA_6,MLE}^{x_e} \forall \ i = 1, 2, \ldots, k \]  

By taking the logarithm of the likelihood function \( L(P_{relA_6}|n = 110, x_{A_6} = 75) \) in equation 9 for \( C_1 = R, C_2 = \neg R \) and \( A_6 = R \), the log-likelihood in equation 13 is obtained.

\[ l(P_{relA_6}|n = 110, x_{A_6} = 75) = ln \frac{110!}{75!35!} + 75 ln P_{relA_6} + 35 ln(1 - P_{relA_6}) \]  

The first derivative of the log-likelihood for equation 13 is calculated as in equation 14. By requiring equation 14 to be zero, the desired MLE estimate is obtained as \( P_{MLE}^{A_6} = 0.68 \).

\[ \frac{\partial l(P_{relA_6}|n=110,x_{A_6}=75)}{\partial P_{relA_6}} = \frac{75}{P_{relA_6}} - \frac{35}{1-P_{relA_6}} = \frac{75-110P_{relA_6}}{P_{relA_6}(1-P_{relA_6})} \]  

16
In summary, the value for $P(A_6=R|C_1C_2)$ in figure 1 given that $C_1 = R$, $C_2 = \neg R$ and $A_6 = R$ is 0.68. When $C_1 = \neg R$, $C_2 = R$ and $A_6 = R$ then the value of $P(A_6=R|C_1C_2)$ is 0.06. When $C_1 = R$, $C_2 = R$ and $A_6 = R$ then the value for $P(A_6=R|C_1C_2)$ is 0.34. Finally, when $C_1 = \neg R$, $C_2 = \neg R$ and $A_6 = R$ then the value for $P(A_6=R|C_1C_2)$ is zero ($A_6$ is only relevant within the context of any combination of $C_1$ and $C_2$). Figure 7 is an update of the two level BBN representation of TickX use case to artefact trace links showing their respective local PDT based on the interaction events generated in figure 2.

2.3. Inference of entity relevance based on calculated posterior probabilities

By making explicit the underlying dataset we can see how the network structure that has been formed (section 2.1) and the estimation of local probability distribution over the belief network structure (section 2.2) provides the information required to infer the relevance of entities based on their posterior probabilities. This is achieved by using the Bayes’ theorem as stated in equation 1 for a selected set of evidence and query entities. The respective joint probability distributions (see equation 2) are simplified based on dependencies existing in the belief network. Furthermore, the calculation
of the posterior of a query entity based on existing evidence set is achieved by inserting the estimated local probabilities from the PDT into the joint probability distribution.

The variable elimination algorithm [36, 9] can be used in the determination of posterior probabilities in a tractable manner. The key to this algorithm is the summing of variables that appear in only one factor out of the distribution. The results presented in this paper is based on the variable elimination algorithm implemented in the JavaBayes Bayesian networks inference tool for learning about belief networks [6].

In our modelling of requirements traceability as a belief network, two approaches are used in deriving the evidence set, which is then used in the determination of the posterior probability of relevance of a query entity. In the first approach, which is also referred to as single variable evidence, the evidence set consist of only a single entity instance of interest. In this approach, it is assumed that only the relevance state of the entity of interest is known while every other entity is unknown. Equation 5 is an example where single variable evidence is implied. The equation computes the posterior probability of the relevance of $A_1$ given that $C_2$ is the use case of interest. Here the evidence set consist of only $C_2 = R$ which is the use case of interest. Figure 8 represent the posterior probability of relevance of artefacts based on single use case evidence. Figure 8(a) provides insight into the relative relevance of artefacts assuming that $C_1$ is the use case of interest. For instance, the plot in figure 8(a) suggest the relative higher relevance of $A_6$ compared to other artefacts that have been associated with the use case. Similarly, figure 8(b) provides insight into the relative relevance of $A_1$ compared to other artefacts given that $C_2$ is the use case of interest.

The second approach used in deriving the evidence set, also referred to as multiple variable evidence, involves the evidence set consisting of more than one entity instance of interest. In this approach, the relevance state of a number of entity instances are known. Typical examples are shown in figure 9, each of which involves the knowledge of the relevance state of more than one use case. The plot in figure 9(a) shows the posterior probability of relevance of artefacts given the states of $C_1$ and $C_2$ (i.e $C_1 = R$ and $C_2 = \neg R$). Similarly, figure 9(b) shows the posterior probability of relevance of artefacts given that $C_1 = \neg R$ and $C_2 = R$. Finally, figure 9(c) shows the posterior probability of relevance of artefacts given that $C_1 = R$ and $C_2 = R$. In each of these scenarios, different posterior values are obtained.
3. Linear accumulative relevance model

The second approach proposed in this paper for relevance ranking of use cases, developers and code artefacts associated with a selected traceability link is based on the linear accumulation of relevance values of entities. In addition to relying on the monitoring of core view, update, and create interaction events that affect the state of entities (as presented in the bayesian approach), the linear model also relies on capturing a context size dimension for the entities involved in each interaction event. We first formally describe how this context is captured and defined, this is followed by the modality used to represent the size of the context.

During collaborative software development projects, different work contexts (associations between use cases, developers and artefact entities) are formed that characterise the situation of entities in a collaboration space. These work contexts are constantly changing in response to events, and entities may participate in one or many work contexts.

From the example scenario described in section 2 it is possible to create work contexts (represented as graphs) for each entity to capture the relational properties between them. Each interaction event related to an entity can contribute a node to the context graph (if an interaction event refers to an entity instance not yet represented in the graph, a node for the instance is added to the graph). For example, the context graph of Amy will consist of every use case participated in (just one - 'Purchase Tickets') and code artefacts that she has created, updated or viewed (there are five of them). Similarly, the context graph of 'Purchase Tickets' use case will consist of every
code artefact that was created, updated or viewed and the developers that carried out the interaction events while working on the use case. Finally, the context graph of each artefact (consider MovieCatalog.java, for example) will consist of every use case and developer associated with the views, updates and create events carried out on MovieCatalog.java. Figure 10 illustrates these work context graphs. Similar graphs are created by the accumulative relevance model for the other developers, use cases and code artefacts.

During collaborative software development, it is expected that the size of an entity’s work context or the number of other entities to which the selected entity serves some usefulness, is proportional to the relative influence that such an entity exacts on the collaboration space. For example, a use case that has existed for a long time in a collaboration space and has several developers implementing the use case using a number of artefacts, is considered to hold more information about the state of the project compared to a use case that is newly introduced into the collaboration space and has a small number of
associated developers and artefacts. A similar line of reasoning holds for artefacts and developers. This size dimension is captured by the concept of sphere of influence (SOI).

SOI is a general concept used to capture both geographic and semantic groupings, and provides a well-defined boundary for interactions. For example, Gutwin et al. [16], in their work on workspace awareness for groupware systems, refer to SOI as where collaborators can make changes within a shared artefact. SOI in this research refers to a region over which an entity exacts some kind of relevance (which is in turn determined by the interaction events), is defined by its work context and is directly proportional to the number of entities that constitute a work context.

The SOI ratio is used to represent the relative influence an entity exacts on the collaboration space. The SOI ratio of an entity is defined as the ratio of the total number of unique entity instances directly associated with an entity (the size of its work context) compared to the total number of unique entity instances in the whole collaboration space (excluding same-type associations - developer-developer etc.). To take an extreme case, a developer who worked on all artefacts associated with all use cases would have a SOI ratio of 1.

The rationale for SOI is to provide a factor in our linear traceability model that also captures a sense of an entity’s trace relevance based on the range of other entities (irrespective of their type) it had previously been traced to. A typical scenario will be a case where two developers D₁ and D₂ work on the same project. Assume D₁ has a long history of work on the project (evidenced by the number of code artefacts and use cases D₁ has worked on) and D₂ is newly introduced into the project. Also assume D₁ and D₂ carry
out relatively equivalent number of interactions on artefacts to achieve a use case $C_x$ at time $t$. In this scenario, SOI provides a basis to rank the extent of relevance of traces from $D_1$ and $D_2$ to the common use case they have worked on ($C_x$). Within this scenario, it is also possible to assume that the developer that has a longer history of interaction events with project entities is more likely to generate more strategic events compared to a relatively new developer with a shorter history of interaction with project entities. Based on developer’s influence on the project, $D_1$ is expected to be higher in the traceability relevance ranking relative to $D_2$ for $C_x$ irrespective of their both having equal number of interactions.

Based on the example scenario described in section 2, it is possible to calculate the SOI of each entity represented in the collaboration space. Figure 11 is a SOI representation of developers for the TickX project scenario. Similar representations can be created for tasks and artefacts. As shown in Figure 11, the SOI of Amy is defined as $6/9$ (entities within Amy’s work context / total number of entities - 2 tasks and 7 classes). Similarly SOI ratios are calculated for other developers, artefacts and tasks.

Figure 11: Sphere of influence representation for developer entities in the TickX collaboration project

Entities that compose a defined SOI can be characterised with overlapping properties. For instance as shown in Figure 11, Customer.java constitutes
only the SOI of Amy, while MovieCatalog.java constitutes the SOI of Amy, Bill and Ruben respectively.

The maximum SOI that E can achieve is 1. This is for a case where E is associated with every other entity that is not of its type in the collaboration space. This is typical for scenarios where the collaboration space consists of a single artefact, use case or developer. A minimum value of 0 is achieved if the work context of E is an empty set; this is typical for scenarios where for example a developer in a collaboration space has not interacted with any use case or artefact or a use case has not been associated with any developer and artefact. On a whole, as the number of entities that consists an entity’s work context relative to the number of entities in the collaboration space increases, the SOI ratio of the entity also increases.

The concepts of development work context, interaction events associated with an entity, and the variation of its SOI ratio forms the basis of the linear accumulation of relevance discussed in this paper. This model is intended to provide accurate, real-time traceability information on the overall work effort of individual developers; an indication of which use case and artefacts have consumed most effort over all developers; and hence an indirect indication of the relevance of entities that constitute a traceability link. In this approach, the linear model cumulatively builds the relevance values of entity instances as they are associated with interaction events and as their SOI ratios vary. This paper discusses how these cumulative relevance values are derived for the history mode and how they are further used to derive a relevance list of developers, code artefacts or use cases associated with a particular trace link. A more general application of this approach [30] also derives relevance values for the recent mode and can provide a perception of the relevance of the most recent trace links that have been formed.

3.1. History mode

The history mode aims to provide a relevance perception of trace links based on the overall dissipation of work effort across entities. This is computed by linearly combining the relevance values associated with an entity in a selected work context before an interaction event with the relevance gained as a result of the interaction event. The relevance gained as a result of an interaction event is dependent on the type of interaction event and the SOI ratio of the selected entity work context. More formally, the cumulative relevance value $x$ gained by an entity instance $e$ in response to an interaction event at time $n$ is represented by equation 15. The type of interaction is
Table 2: History mode relevance values of entity instances constituting the 'Purchase ticket' work context for every event influencing the context.

<table>
<thead>
<tr>
<th>Interaction type</th>
<th>SOI Ratio</th>
<th>Relevance value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Create</td>
<td>1.0000</td>
<td>0.0100</td>
</tr>
<tr>
<td>2 Update delta = 300</td>
<td>1.0000</td>
<td>0.0400</td>
</tr>
<tr>
<td>3 Update delta = 150</td>
<td>0.5000</td>
<td>0.0475</td>
</tr>
<tr>
<td>4 View</td>
<td>0.5000</td>
<td>0.0480</td>
</tr>
<tr>
<td>5 View</td>
<td>0.6667</td>
<td>0.0480</td>
</tr>
<tr>
<td>6 Create</td>
<td>0.6250</td>
<td>0.0480</td>
</tr>
<tr>
<td>7 Update delta = 175</td>
<td>0.6250</td>
<td>0.0480</td>
</tr>
<tr>
<td>8 View</td>
<td>0.6250</td>
<td>0.0480</td>
</tr>
<tr>
<td>9 Create</td>
<td>0.6667</td>
<td>0.0480</td>
</tr>
<tr>
<td>10 View</td>
<td>0.7778</td>
<td>0.0480</td>
</tr>
<tr>
<td>11 Create</td>
<td>0.8000</td>
<td>0.0480</td>
</tr>
<tr>
<td>12 Update delta = 84</td>
<td>0.8000</td>
<td>0.0480</td>
</tr>
<tr>
<td>13 Update delta = 25</td>
<td>0.8000</td>
<td>0.0480</td>
</tr>
<tr>
<td>14 View</td>
<td>0.8000</td>
<td>0.0480</td>
</tr>
<tr>
<td>15 Update delta = 5</td>
<td>0.8000</td>
<td>0.0480</td>
</tr>
<tr>
<td>16 View</td>
<td>0.8000</td>
<td>0.0480</td>
</tr>
<tr>
<td>17 View</td>
<td>0.8000</td>
<td>0.0480</td>
</tr>
<tr>
<td>18 Create</td>
<td>0.8000</td>
<td>0.0480</td>
</tr>
<tr>
<td>19 Update delta = 60</td>
<td>0.8000</td>
<td>0.0480</td>
</tr>
<tr>
<td>20 Update delta = 90</td>
<td>0.8000</td>
<td>0.0480</td>
</tr>
</tbody>
</table>

represented by \( t \) and the different values it can assume are shown in Table 1. The SOI ratio of entity \( e \) is represented by \( s \).

In other words, the relevance value for entity \( e \) (considering all the interactions associated with \( e \) after time \( n \)) is based on its previous value plus the value of the interaction multiplied by the SOI ratio of the entity. A ranking of entities based on their relevance values within the work context forms the relevance list for that context. Entities with the highest relevance values are positioned at the top of the list. Using the relevance lists, mined traceability links based on developer interaction events can then be associated with information on how linked entities impact on each other.

\[
x_{(n)}e = (x_{(n-1)} + t_{(n)} * s_{(n)})e
\]

(15)
3.2. Illustration

To illustrate how our linear representation can be used to obtain trace links with weighted relevance, we reuse the assumed interaction event trails shown in figure 2 for the TickX project. Also, any selected time point corresponds to at least one event associated with a developer, a use case and an artefact as described in section 2.1. Table 2 demonstrates the accumulation of relevance values of entity instances constituting the achievement of the Purchase Ticket use case across the timelines shown in figure 2.

Table 2 is used to demonstrate that an entity instance accrues an additional relevance value each time it is used to achieve the Purchase Ticket use case. Based on figure 2, the project started with the creation of Account.java code artefact by Amy while accomplishing the Purchase ticket use case. Using equation 15, the first recorded event associates relevance values of 0.01 to both Amy and Account.java respectively. This is subsequently followed by her updating Account.java that generated an update delta size of 300. Again, this second event further increases the relevance of Amy and Account.java to 0.04 respectively. Over the lifetime of TickX, relevance values updates are constantly recalculated for each entity instance that is used to achieve Purchase Tickets. Finally, the relative relevance position of entities are generated based on the accumulated relevance values for each entity instance that is associated with Purchase Tickets. The final relevance values are sequenced in a descending order.

The relative relevance positions of entity instances on the trace links representing a selected work context can provide insight into overall work effort dissipated over entity instances constituting the context. Figure 12 shows that within the Purchase Tickets work context, Account.java has had greater overall influence on the state of the use case compared to other code artefacts. Also, Amy is attributed with greater overall coding effort on this use case compared to other developers. Furthermore, entity instances achieve different levels of relevance across the range of traceability links they are associated with. Although MovieCatalog.java has influenced the state of both the Purchase Tickets and Browse Movies use cases, its relative relevance to the two use cases is different. As demonstrated in figure 12, MovieCatalog.java is at the top of the artefact relevance ranking for trace links representing Browse Movies, but lower in the artefact relevance ranking of trace links representing Purchase Tickets.
4. Implementation

We have implemented a system called CRI (Continuum of Relevance Index)\cite{30, 31} that can be used to monitor interaction events generated by developers that are working in a distributed manner. A client-server architecture was chosen to implement the CRI where each developer’s Eclipse IDE is a client and the model processing logic and storage of interaction sequence data is performed on the server. The client monitors sequences of view, update and create interaction events that are executed within Eclipse, which was chosen as the basis for the implementation because of its open plug-in architecture. When a network connection exists, this event data is offloaded to the server and synchronised with that of other developers. While there is no connection (or a slow connection) the client can temporarily store event data locally and perform local model processing logic to give the developer a partial view of current trace links and relative relevance - offline mode. The CRI implementation architecture is as shown in Figure 13. The architecture is distributed across client and server ends, and consists of four core layers: the model, event, messaging and Rich Client Platform (RCP). The client end of each layer is plugged into the Eclipse platform while the server end resides on an Apache Tomcat web application server.

The model layer is the main event processing unit in the architecture
Figure 13: CRI architecture

shown in figure 13. This layer implements the Bayesian and linear approaches described in this paper. The event layer is responsible for capturing and archiving interaction event sequences generated within a collaboration space. The \textit{log.event} component is the clearing centre and data warehouse of all events generated by collaborators. The messaging layer carries out asynchronous processing of request/response messages from the server. The \textit{offline.emulator} component emulates the server end functions of the model and event layers while a developer is generating interaction events in offline mode. Finally, the RCP layer resides only on the client end, and provides the minimal set of components required to build a rich client application in Eclipse.

Figure 14 is a snapshot of an Eclipse view of the \textit{visualisation.rpc} component. System developers can open, activate and deactivate their use cases of interest by using the popup menu labelled 3 in figure 14. All interaction events carried out by the developer are traced to the work context of an activated use case. The RCP layer is also responsible for generating visualisations of the outcome of monitored and processed trace links and the relative relevance of associated entities. A relevance based visualisation of use case trace links is as shown in figure 14 labelled 2.

The visualisation of entities involved in a trace link is structured such that entity instances with greater relevance values are positioned at the top of the trace relevance list. The relative difference in the relevance values of entities is depicted using varying colour intensities. Entities at the top of the relevance
list are represented with greater colour intensity. Entities with closely related relevance values show the same relative colour intensity. Figure 14 label 2 is a visualisation of code artefacts that constitute the trace links for the Purchase Tickets and Browse Movies use cases previously presented in the example. For instance, although MovieCatalog.java has been used to realise both use cases, its relative impact on the state of Browse Movies is greater than on the state of Purchase Tickets. Similar visualisations are provided for the relative impact of developers on the state of the two use cases.

The visualisation of historical trace links associated with a software cycle, presents the capability to replay the evolving state of each trace link’s attributes. Such attributes include the relative time the trace link is formed and the changing relevance of associated entities over the life time of a software process. As an example, a playback of the evolution of trace links of the code artefacts associated with the lifetime of Purchase Tickets, can be obtained by sliding through the slider bar labelled 1 in figure 14 (see also figure 15). Such historical visualisation of traceability links formation can potentially enhance the understanding of how the implementation of a software project has evolved from inception to its current state. For instance, as shown in figure 15 (which was generated from event data captured during preliminary study described in section 5), the developer Tracy was respon-
Figure 15: History playback of evolving trace link with associated relevance of code artefacts and developers that have been involved in achieving the User Interface use case.

5. Evaluation

The main objective of this evaluation is to obtain insight on the accuracy of defined traceability models in identifying the artefacts that are consuming more effort to be developed. Such information need is essential for project managers, testers and developers to perform the analysis of the quality of associated code artefacts. In this case, the amount of work effort put in by a developer is used for estimating the relevance of a traceability link. The conjecture is that the higher the effort the higher the complexity of the produced code artefact, and hence a more relevant artefact to analyse for the quality of the generated software. Furthermore, a number of studies have
shown that complexity correlates with the fault proneness of code artefacts [23, 34, 35]. Our evaluation approach is twofold. Firstly, we provide practical insight into the challenges and viability of capturing the interaction events required for relevance analysis by CRI. This is achieved via a usability study of our proposed CRI implementation prototype. Secondly, we investigate the accuracy of the traceability model techniques. This is achieved by comparing developer-reported perceived code artefacts they had put in the most work effort with the posterior probability of rankings generated for both single and multiple variable evidence. Similar comparison is also carried out for the linear model.

5.1. Methodology

The study involved ten advanced software engineering students in the third year of their Integrated Masters/Honours programme in Computer Science, all of whom volunteered to participate. All participants had at least 2.5 years of object-oriented development experience using Java. They were all participating in the group project class developing Gizmoball - an editor and simulator for a pinball table first proposed by MIT- and working in groups of three. Of the ten participants two groups of three were the best two performing groups in the class (G1 and G2), another group of three participants (G3) was also above average, and a single student came from a group that was of average performance (G4). The groups had been designed to consist of individuals of similar academic ability so as to encourage equal participation.

Participants were not restricted to time or place of work. Groups were required to have at least one face-to-face meeting every week; during this time they also discussed their progress with the teaching assistant coordinating the group. Feedback from participants suggested that, besides the mandatory meeting, they also held occasional collocated meetings. All the groups used a version control system. Feedback from group G1 suggests occasional pair programming practice, while group G2 also used a wiki system.

CRI data was gathered over a 6 week development period - 2 weeks of prototype development and 4 weeks of full-scale development. The model was used during development (rather than maintenance) and was used in both a distributed and collocated setting- all participants recorded instances of working from home and within the university campus, the gathered data suggested that participants spent more time working at different times or places
than they spent working together. During the study, use cases or system features were modelled and tagged with meaningful short form descriptions or acronyms that was easy to understand by the collaborators. Furthermore, to minimise intrusion and closely mimic real collaboration scenarios, use cases and system features were defined by developers and subsequently used as a basis for task assignment. These use cases and features were not previously validated by the involved teaching assistant but were rather defined by participants as needed and agreed upon by the associated group.

At the end of the six weeks, structured interviews were conducted with eight of the participants independently and at different times. The two remaining participants (Greg and Smith), were unavoidably absent, as shown in figure 16(a) they also used CRI least during the study. All data presented in this paper has been anonymised for the purpose of presentation to preserve the privacy of the participants. During interview sessions, to determine the extent of use of CRI during the study period, each participant was also asked to state how frequently they remembered to log into CRI on a range of 0-100%, with the lowest frequency being zero and the highest 100. To check if CRI had an impact on the way that participants will normally carry out a programming task, participants were also asked on a scale of 1-7, the difficulty experienced in always working within the context of a use case. 1 is the least impact and 7 the most. A number of other questions were also asked in sequence. These included how frequently a new use case was created or activated as the participant’s work context changed, how difficult it was to create a new use case and to activate an existing use case in CRI.

In the second phase of each interview session, the participant was presented with a list of artefacts and use cases s/he had been working on. The participant was then asked to rank the top four artefacts and use cases based on the overall coding effort s/he needed to achieve the Gizmoball project. Discussed in this paper is the ranking of artefacts produced by participants in groups G1 and G2. These are the two groups for which all the members participated in the interview session. The selected groups also generated a greater proportion of events monitored by CRI.

To obtain insight into the effectiveness of the different traceability models, there is need to understand the level of performance of each model in the retrieval of relevant trace links. In this study, this is achieved by analysing the precision and recall measure for each traceability model. The measure is in relation to the top four rankings of artefacts and use cases as provided by each participant during the interview and based on their overall coding
effort. The main focus of the evaluation is on the accuracy of selected code artefacts relative to the ranking provided by CRI approaches. This was because an uppermost of 6 use cases (as shown in figure 16b) associated with a participant, is rather small to provide useful insight into the accuracy of CRI. Precision and recall are well known information retrieval (IR) metrics used to measure the performance of retrieval systems [22, 2].

\[
Recall = \frac{\sum_{i=1}^{n} (Relevant_i \land Retrieved_i)}{\sum_{i=1}^{n} Relevant_i} \quad (16)
\]

\[
Precision = \frac{\sum_{i=1}^{n} (Relevant_i \land Retrieved_i)}{\sum_{i=1}^{n} Retrieved_i} \quad (17)
\]

\[
AveragePrecision_{Participant} = \frac{\sum (Precision_{Relevant\text{artefact}})}{Total\text{numberofrelevantartefacts}} \quad (18)
\]

\[
MeanAveragePrecision = \frac{\sum (AveragePrecision_{Participant})}{Total\text{numberofparticipants}} \quad (19)
\]

Recall is the ratio of relevant artefacts retrieved for a given participant over the total relevant artefacts for that participant. Precision is the ratio of relevant artefacts retrieved over the total artefacts retrieved (equations 16 and 19). Since for each participant the model retrieves a ranked list of artefacts, a cut-off level \(i\) is used to select the first \(i\) artefacts in the ranking. The precision and recall behaviour of the top \(ith\) artefacts is then analysed, where \(i\) ranges from 1 to \(n\) - the total number of artefacts a participant has been associated with. The average precision for each participant, and the mean average precision that combines the average precision for all participants (equations 18 and 19) is then used to provide insight into the overall performance of each traceability model.

5.2. Results

5.2.1. Viability of capturing the interaction events during project development

Interview data suggested that CRI may have only captured 60-90% of developers’ work effort - see Figure 17(a), and developers changed use case within CRI 25-50% of the time they were actually working on the use case - see Figures 17(b) - 17(e).

Luke and Tony were the two participants that recorded the lower number of interaction events and less frequency in the activation or creation of use
cases with changing work context (60% and 25% respectively as shown in Figure 17(a) and 17(b)). Of the eight participants that were interviewed and took part in the debriefing sessions, two (Boris and Smith) did not switch use cases and worked within the context of only one use case through the development period. The remaining six switched between two and six use cases. In practical scenarios we did expect that it was impossible to capture all developer interaction events. This is because adhering to CRI workflow would not always be a primary concern for a developer amidst other concerns such as meeting project schedules. Thus, while the results of the traceability modelling approaches are likely to be subject to an element of inaccuracy, our aim is to also understand the performance of the models within the bounds of such inaccuracies and where they might exist. This has been achieved with a second interview phase described below, and by comparing the precision and recall outcome of CRI for each of the participants in the study.

5.2.2. Accuracy of traceability techniques compared to participants opinion

Table 3 shows the average and mean average precision for each traceability model. To get an average precision of 1.0, the traceability model must
retrieve all relevant artefacts appropriately ranked in accordance with the participants selection (i.e., the participants top four selections are also the top four on the list of the traceability model rankings. In such a case recall = 1.0 and precision = 1.0). Table 3 shows that the single and multiple variable evidence models achieved relatively low precision (0.04 to 0.22 and 0.06 to 0.19 respectively) compared to the linear model (0.57 to 1).

At a high-level, the outcome of this analysis suggests that a mean average precision of 0.77 can be obtained when using the linear traceability modelling approach for the information need initially presented in section 1. The scenario as captured in this analysis involves six developers who worked on an average of 70 code artefacts to achieve an average of 3 use cases each. A preliminary low-level inference for this scenario suggests a mean average precision of 0.77 in finding the right traceability information need from the top four of a traceability relevance ranking of associated entities.

Finally, the aim of this analysis is also to provide insight into the con-
sequence of low activity and formal use case activation data. As mentioned above, Luke and Tony were the two participants that recorded fewer interaction events and less frequency in the activation or creation of use cases with changing work context. Average precision for the linear model representing Luke and Tony shown in table 3 were lower than for the other participants. Table 4 shows the number of relevant artefacts retrieved for the linear approach at selected cut-off values for $i$. Unlike Luke and Tony, the other participants retrieved all relevant artefacts at an earlier cut off mark. Furthermore, given that use cases and code artefacts exist in a shared repository and that traceability relevance values are also centrally estimated, low activity data for a member of a group has the potential of also affecting the precision of the traceability relevance outcome of other collaborators in the group. This outcome, which we aim to further investigate, suggests the reason why Alex retrieved the top four artefacts at a relatively lower cut-off value as shown in table 3, compared to collaborators of group G2.

The preliminary inference from this analysis suggests that although from a pragmatic viewpoint it is difficult to always capture all events generated by a developer or consistent activation/creation of use cases with changing work context, a threshold is required for which more accurate traceability rankings can be obtained. Establishing such a threshold and its specified attributes is the focus of future evaluation. The outcome of the amount of data captured related to formal use case activation/creation (Figure 17(c)) also suggest that we need to investigate other approaches as an organising concept for the work of developers.

5.3. Threats to validity

A standard criticism of this kind of university-based research project is the use of students. The best that can be done is to use experienced students working on realistic development projects. The project only lasted ten weeks, and was only monitored for six weeks. Therefore, these findings must be treated with caution. However, we argue that they provide a reasonable indication of the potential strengths and weaknesses of each of our model approaches for enhanced requirements traceability between use cases, system developers and associated artefacts using interaction event trails. A related threat is that the participants had limited experience of collaborating in groups and this may have impacted their working practices compared to more experienced participants.
Table 3: Average precision and mean average precision for different relevance model

<table>
<thead>
<tr>
<th>Participant</th>
<th>Linear</th>
<th>Single variable</th>
<th>Multiple variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tony (G1)</td>
<td>0.57</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>Alex (G1)</td>
<td>0.64</td>
<td>0.06</td>
<td>0.19</td>
</tr>
<tr>
<td>Luke (G1)</td>
<td>0.63</td>
<td>0.08</td>
<td>0.13</td>
</tr>
<tr>
<td>James (G2)</td>
<td>0.78</td>
<td>0.04</td>
<td>0.07</td>
</tr>
<tr>
<td>Paul (G2)</td>
<td>1.00</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>Tracy (G2)</td>
<td>1.00</td>
<td>0.22</td>
<td>0.13</td>
</tr>
</tbody>
</table>

| Mean Average Precision | 0.77 | 0.08 | 0.10 |

Another threat is that CRI did not accurately capture all development data. It is clear that the participants did not record all use cases that they worked on and did not always change use cases as they changed work context. This is a challenge for any use case based model harvesting traceability links from developer interactions. An alternative is to explore a similar model based only on artefacts and developers (without use cases).

The studies were part of an assessed university course. Participation was voluntary and the lecturer associated with the course was not involved in any interviews or data analysis. He only saw anonymised data. The assignment of participants to groups was not carried out within the context of the experimental study but rather as part of the assessed university course. The groups were then approached subsequently to request their participation in the experiment. This explains the reason for having only one participant from group 4. Also, groups had been designed to consist of individuals with similar academic abilities. Thus, results obtained in this study might differ from groups where there is a substantial variation in abilities.

The results may have been impacted by participants’ lack of experience with CRI. Again for pragmatic reasons, participants were only provided with a CRI user guide and a 30 minute tutorial. Some participants may not have developed a sufficient understanding to fully utilise CRI and gain deeper insights into its strengths and weaknesses. This study was carried out in the context of a forward engineering project and no deduction can be made about its use in reverse engineering or maintenance contexts.
Table 4: Number of relevant artefacts retrieved at cut-off(i) based on the linear traceability model

<table>
<thead>
<tr>
<th></th>
<th>Paul</th>
<th>Tony</th>
<th>James</th>
<th>Luke</th>
<th>Alex</th>
<th>Tracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Rel</td>
<td>i</td>
<td>Rel</td>
<td>i</td>
<td>Rel</td>
<td>i</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td></td>
<td>9</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td></td>
<td>10</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td></td>
<td>11</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td></td>
<td>12</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td></td>
<td>13</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td></td>
<td>14</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td></td>
<td>15</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Participants remembered to log into the tool 60-90% of the time, this represents some loss of data during work sessions, and we do not know the amount or type of work they performed in the unlogged sessions. While the participants pointed out that this included coding and non-coding activities, the precise amount of time spent on each could not be established. Thus, the interactions of participants during the period that was not monitored could have potential effect on the results. Again, the evaluation primarily tests how effectively the defined models can identify which artefacts are most relevant to a use case. The outcome is primarily dependent on the participant’s ability to recall and assess what is most relevant. It is also worth mentioning that the choice of top four artefacts was made based on experience gained during weekly formal meetings that the teaching assistant (first author) for the course had with three different collaboration groups that took the course. During the meeting, one of the aims was for each member of the group to identify and discuss the code artefacts and the use cases they had
individually worked on since the previous meeting. The number of artefacts that participants discussed ranged between two and six. Our main concern was choosing a number that would not put too much memory strain on the participant on the ability to recall/assess what is relevant and at the same time get feedback that is accurate and not based on assumptions.

On the whole, the outcome of the evaluation can be applicable for software projects with moderate complexity developed on short term basis. This is as a result of the number of students that participated in the study and given that only one system (Gizmoball) was considered during the evaluation of our approach. More empirical studies will be required to understand how these results are can be extended to longer term software projects.

6. Discussion

The main aim of this paper has been to explore approaches that can provide a perception of the relevance and impact of entities associated with requirement traceability link information. This has been achieved by using monitored developer interaction event trails within a software development tool environment. Two main model approaches have been investigated. The first approach is based on Bayesian inference technique. This technique explores the use of single and multiple variables as evidence nodes to determine the posterior probability of the relevance of use case, developers and code artefact entity instances. The second approach is based on a linear model, which is used to accumulatively determine the relevance of use cases, system developers and code artefacts to a requirement traceability link.

One of the lessons learnt in this research is that monitoring developers’ tool interaction events can help rank and categorise traceability link information for particular needs. While relevance ranking based on overall work effort, a number of other ranking possibilities can also be investigated. These include relevance rankings of use cases, developers and artefacts based on recent work effort, difficulty, and associated bug triage respectively. It is expected that each of these rankings will provide different traceability insights. These insights can also be used to further augment requirements traceability information derived from static and dynamic analysis of a code base and other resources where the harvesting of contextual relations is difficult to obtain.

The preliminary study revealed some shortcomings in the use of BBN to suggest the relevance of traceability links. As demonstrated in table 3,
the single and multiple variable BBN model had inconsistent trends with low precision values for all the participants in the study. While this study has not postulated a solution to reducing such inconsistency and improving on the precision of BBN approach, sparse interaction event data, especially at the early phase of software project development, needs to be further investigated.

The need for sparse data investigation is based on the insight that there will always be scenarios where for instance a developer works on code artefacts or use cases that he/she has never worked on before. In such cases, there is no evidence on which to base the estimated probabilities on such entity’s PDT. The extent of this effect, also referred to as the zero frequency problem on the outcome of BBN traceability relevance ranking is unknown. The rationale for our approach builds on initial insight from previous but unrelated work by Dahll [7] on probability estimation based on observations entered into BBN. The outcome of Dahll’s work suggests that one wrong (negative) observation, as well as a set of few negative observations, is not enough to change the overall results of estimated probabilities. A closely related outcome is also demonstrated by Montironi et al. [28]. Furthermore, if for instance, the developer had previously not worked on the use case or code artefact before, then such use case or code artefact does not form the work context of the developer and hence not of any trace relevance to the associated developer. Thus, entity instances that had not been associated with the developer interactions under consideration prior to probability estimations (either because they have not been worked on or that their associated events have not been recorded and hence can give rise to zero frequency problem), are not considered during the calculations of the joint probabilities for the associated developer’s traceability work context.

This study also reveals the potential of the linear model in the relevance ranking of entities associated with a traceability link based on overall work effort. While it is assumed here that work effort has a direct correspondence to the amount of interaction events generated from a collaboration workspace, this might not always be the case. For instance, undocumented work effort in the design phase might not be noticed in the coding phase. On the whole, this research points to a need for more studies to validate the usefulness of our linear approach in providing requirements traceability information for development processes where use cases are used as a structure for the definition and assignment of tasks.

Another interesting point learnt from the linear modelling approach was that the SOI ratio can be central in revealing a number of latent requirements.
traceability properties during collaborative software development process. For instance, a high SOI ratio for a developer suggests that s/he is working with many parts of the system and hence central to the development process (for example a chief architect that is also doing some programming). Furthermore, assuming every developer tends to be associated with high SOI ratio, this might imply a shared code ownership development model like in XP. If a use case has a high SOI ratio, more than one deduction can be made. Firstly, on conditions of best practice in the definition of use cases and pragmatic architectural design; this can indicate the importance of the use case to the development process given that it has been associated with a higher number of developers and/or artefacts. Secondly, this can also indicate bad use case definition and allocation practice - for instance, the use case has not been broken down enough or the development process has not been well segmented. Also, if the use case coincides with part of a system being connected with everything else, maybe the design is not good enough, or the architectural design and the use case split up does not fit. The use of SOI for a forensics analysis of the latent properties of development processes aimed at enhancing requirements traceability is the focus of another paper. In this work, best practice in the definition of use cases and pragmatic architectural design was assumed.

Although the purpose of this evaluation has been focused on the significance of the Bayesian and linear approach to requirements traceability, the participants also demonstrated the usefulness of the tool during their development activities. For instance, requirements traceability rankings were used to provide insight on architectural styles. Typical example of revealed style is the n-tier architecture revealed while achieving gizmoball feature by Tony in group G1 shown in figure 18. The size of each node in figure 18 depict their relative relevance in achieving the associated feature. Furthermore, the study also revealed that the traceability outcome can also be used to understand the architectural implications of the different interaction events carried out while achieving a project endeavour. Such architectural implications included impact of executed events on initial system decision and also identifying critical pointers to bottlenecks and information centres in the software project. A drawback observed from the study was that the traceability network became increasingly cluttered as the number of entities associated with a project increased. Thus, while a selected entity from a traceability network could be moved around within the implementation interface for visual clarity, this was a difficult process for complex networks.
From a more general viewpoint, this research suggests a novel approach to representing and understanding requirements traceability. Traditionally, traceability has been understood as a Boolean relationship - either there is a link between two entities or there is not - which is often documented in a traceability matrix or using some form of hyperlink. A typical traceability matrix is shown in figure 19. Here we see for instance that requirement $R_2$ is implemented through components $C_2$, $C_3$, and $C_m$. This research suggests that it is possible to obtain a shift from a Boolean to a fuzzy understanding of traceability, where the extent of relevance of one entity to another is seen as a position in a continuum. Some of the possible interpretations of such fuzzy understanding are highlighted below:

- **Incomplete knowledge**: A fuzzy approach to traceability can be used as an expression of incomplete knowledge and provides information on how certain it is that a component participates in the implementation of the requirement. High relevance values for components associated with a requirement suggest that the components are more likely to have participated in the implementation of the requirement.

- **Relationship frequency**: A second possible interpretation of a fuzzy approach to requirements traceability is that of relationship frequency. This is because, even with complete knowledge, there are some com-
ponents and requirements that are closely connected, while others are only marginally connected, i.e., the component will only need to be changed under some conditions. A related interpretation could be that a relevance value of 0.30 means that this component implements 30% of the functionality of the use case. The relationship frequency approach would also be applicable to non-functional requirements. A component that is accessed several times during development will most likely be a component that has a lot of influence on the non-functional requirement in question.

- **Cohesion measure:** An additional possibility could be to interpret the relevance values as a measure of code cohesion. If a function has many traces with low relevance values, this could be an indicator of low cohesion - and maybe high coupling.

![Traditional requirements traceability matrix](image)

Figure 19: Traditional requirements traceability matrix

There are a number of issues we still seek to address in this research. The interpretations of fuzzy understanding from relevance values of traceability links require further empirical studies. There is also need for some insight on how such fuzzy knowledge differs from traceability links that are generated using information retrieval techniques. The approaches described in this paper do not measure the time duration developers spend viewing code artefacts. Developers may be spending more time on entities that they consider more important. Furthermore, developers might put more hours of work in fewer lines of code which are complex than in more lines of code which are easy to implement. Thus, it can be argued that time would be a much better indicator for effort than the lines of code as represented in this paper. It is expected that the measure of such viewing time and perhaps the sizing of view events based on scrolling and mouse movement can increase the accuracy of any of the approaches in its relevance estimation based on
different criteria. The main challenge here is that it is difficult to determine the time actually spent viewing the artefact. This is because the duration of time an artefact is left open may not directly suggest that it is being viewed to gain understanding. Also, distinguishing between local updates that are never committed and updates made visible to others through commits to the source repository can arguably increase the accurate measure of work effort. The granularity of interactions for the current implementation and evaluation of presented models has been carried out at the artefact file level. It is expected that any of these approaches can easily be scaled to a lower level of granularity, for instance by monitoring the creating, viewing and updating of artefacts at the method level. We also aim to carry out subsequent evaluations, especially in establishing a required threshold in the amount of interaction activity captured and the frequency of use case activation data.

7. Review of related work

A simplistic approach to the traceability problem arising from complex dependencies is the enforcement of strict partitioning. In strict partitioning, system developers are assigned to use cases and also own the code used in achieving such use case. Related research [14] has cited project longevity as one of the underlying problems of requirements traceability. Such longevity comes with the evolution and reassignment of use cases and associated artefacts. Also, in software development settings such as open source projects, artefact ownership is not always obtainable [17] [26].

Research related to traceability links between requirements, documentation and source code has been presented in Antoniol et al. [2, 3] and Penta et al. [32]. The core viewpoint is that information retrieval techniques can provide ways to semi-automatically recover traceability links between documentation of a system and its source code. Also, LeanArt by Grechanik et al. [15] is a semi-automated approach to recovering and using use case diagram to source code traceability links. Poshyvanyk and Marcus [33] proposed an approach for using traceability links to assess and maintain the quality of software documentation such as use case descriptions and user manuals. Their work explored the use of information retrieval techniques to compute similarities among sections of software documentation. The work of Antoniol et al. and other related research is based on the premise that programmers use meaningful names such as functions, variables, types, classes and methods. This assumption implies that application domain knowledge
processed while writing code can be captured by mnemonics for identifiers. The analysis of these mnemonics is then used to trace high level concepts with program code concepts and vice-versa. The vector space model and probabilistic models based on Bayesian inference or latent semantic indexing are then used to provide a ranked list of documents that can be mapped to source code artefacts. Related analysis of support given by information retrieval techniques during requirements traceability have also been presented by De Lucia et al.[8]. The main outcome of the study by Lucia et al. was that traceability recovery tool significantly reduces the time spent by the software engineer with respect to manual tracing.

The evaluation of Antoniol et al. and other related approaches also present significant precision and recall results. The main distinction between document based IR as exemplified by Antoniol et al. and the approach presented in this paper lies in the analytical means used to instantiate traceability links between entities. The document based traceability retrieval technique is highly reliant on the accurate use of mnemonics in the code base that can be mapped to higher level software document. Also, one advantage of such techniques is that they allow for the ranking of results (source code artefacts) based on their relevance to the query (a use case for example) amidst many ambiguities in the process. This paper rather, presents a traceability instantiating approach that is based on the core events that affect the state of involving entities (in this case documents and source codes). Our approach is particularly useful in scenarios where existing higher level documentation of the system is not sufficient for the mapping of the mnemonics that exist in the code artefacts. A typical scenario will be agile software engineering approaches where less documentation describing the system is envisaged. In an agile environment, user stories are short and simple description of the expectations for the system from a user point of view. Such user stories might not be appropriate for applying IR techniques for the purpose of mapping source code mnemonics. Furthermore, our approach has been extended to provide a categorisation of the expertise of different system developers and analysts depending on the aspect of the system they have worked on.

There exist a number of research approaches to identifying traceability links that can enable expertise recommendation. These approaches can also offer means to determine the expert developer to contact in case questions regarding an artefact or use case. These approaches are mainly focused on the isolated mining of historical, runtime and code repositories. For instance, Anvik et al. [4] have applied machine learning techniques to open source
bug repositories to learn a small number of developers suitable for resolving a report. Kagdi et al. [18] presents an approach recommending a ranked list of developers to assist in performing changes to a given source code file. Kagdi et al. have achieved this by combining IR technique that uses Latent Semantic Indexing (LSI) for textual analysis of source code files that exist in repositories. The meta-data held by the code repository on associated source code is then further mined to recommend a ranked list of candidate developers for source code change. Again, German [13] recommended an approach that can enable developers to be aware of who are the people who tend to work in the same code as they do. German’s approach was achieved by analysing CVS code repositories. Mockus and Herbsleb [27] introduced Expertise Browser that use experience atoms to measure expertise of developers. Other approaches include Emergent Expertise Locator by Shawn and Murphy [25] and Expertise Recommender by McDonald and Ackerman [24].

Our approach to identifying expertise based on core events that can impact the state of entities builds on this existing work. This is because a subset of events we have analysed, such as updates, can be extracted from software repositories. In addition, we consider developer view events that are normally not captured by software repository. Our extended approach can enable the capturing of additional traceability relations that might not be syntactically inferred by analysing repositories. For instance, the traceability information on potential code artefacts a developer viewed to gain understanding is revealed. We anticipate that such understanding can influence the further update of the same or other code artefacts. For example, a change to the code that writes data to a file may require understanding changes to the code which reads data from the same file, although there exists no traditional (example, data and control flow) dependencies. Furthermore, the metrics for our measure of the relevance of traceability relations is not just based on some form of IR similarity measure that is highly dependent on string matching or the size of updates carried out by developers. Our SOI concept enables the representation of the relevance of traceability links that is based on the relative influence of a trace entity on the state of other entities that exist in the software project. On the whole, the basis of this research is hinged on the conjecture that mining historical, run-time and code repositories, in combination with more social and unstructured data source can uncover useful and important traceability patterns and information.

Briand et al. [5] investigated the problem of traceability in model-driven development practices by monitoring users’ modifications. Trace links are
identified by first determining the intent of the designer that lay behind changes when models are refined. The intent is modelled by a taxonomy of refinements. Rules associated with different types of refinements are then used to determine traceability links. Refinements are based on atomic change types including add, delete, move and change. While in Briand et al.'s work, impact analysis between models at different levels of abstraction is simply defined as a subset of the target elements of a traceability link, our approach extends the expressiveness of trace links by presenting the relative relevance of the elements. Mader et al. [21] proposed an approach for the automated update of existing traceability relations during the evolution and refinement of UML analysis and design models. The approach observes elementary changes applied to UML models, recognises the broader development activities and triggers the automated update of impacted traceability relations. Similar to Briand et al., the elementary change events on model elements include add, delete and modify. The broader development activity is also recognised using a set of rules that helps in associating an elementary change as constituent parts of intentional development activity. The key similarity between the work presented in this paper and Mader et al.'s approach is the focus on maintaining up-to-date post requirement traceability relations. However, our approach also provides a perception of the relevance of the links based on the amount of development activity required to achieve and maintain the links between the different entities. Furthermore, this paper also aimed to provide not only traceability to model representations but also to the system developers.

Finally, a number of other general techniques have been used for elucidating requirement traceability between different entities during software development processes. These include cross referencing schemes, key phrase dependencies, use of templates, different forms of requirements traceability matrix, constraint networks, and hypertext [14]. Each of these techniques differs in the quality and diversity of traceability information they present. Their use, however, gets increasingly complex for non-trivial software projects with longer lifecycle and projects with high developer turnover. The approach introduced in Mylyn [19] is closely related to ours in creation and maintenance of trace links by monitoring users’ modifications. The degree of interest model presented in Mylyn can provide traceability insight, but only from the viewpoint of artefacts that are related to a task. Also, Gull et al. [11] has presented a related approach based on data mining technique. Gull et al. used relation analysis to reveal insights on the logical coupling of
modules. Gull et al.’s work compared artefacts (classes) based on dates and authors of changes. With this information, parts of the system that were changed together can be identified.

8. Conclusion and further work

This paper has presented a Bayesian and a linear inferencing technique to model the relevance of use cases, developers and artefacts associated with requirements traceability links. Both techniques depend on the interaction events trails left behind by collaborating developers while working within a development tool environment such as Eclipse. The Bayesian inference technique explores the use of single and multiple variables as evidence nodes to determine the posterior probability of relevance of use case, developers and code artefact entity instances. The linear technique accumulatively determines the relevance of use cases, system developers and code artefacts to a requirement traceability link.

The main advantage of our approach over related work is that it can be used to provide insight into traceability information needs that are contextual such as the appropriate developer to seek for help, and the artefact that has most affected the state of a use case. Such information would be difficult to obtain from structured document resources. An initial evaluation using advance level masters/Honours software engineering students has been presented. The outcome of the study suggests an advantage for the linear approach in the relevance ranking of use cases, developers and code artefacts associated with a selected traceability link. We also explored the challenges of the two techniques.

In the short term, our further work focuses on getting more insight into the consequence of the so called zero frequency problem on our BBN traceability modeling approach. We also plan to validate the different traceability relevance approaches using existing open source systems. Using open source systems, the aim is to capture create and update events by developers and also leverage the number of artefacts and developers normally collaborating over such systems. Longer term further work will investigate the feasibility of developing a more sophisticated and robust approach that can adapt to differing dynamic properties such as the type, lifecycle and nature of development use cases; the programming domain; and developer profiling. Only source code artefacts have been considered in CRI trace analysis, due to the complexity of tracking other forms of artefacts within Eclipse IDE. This pro-
vides a focus for further work in investigating an appropriate mechanism that will enable the capturing of non-code based artefacts. Finally, a comparison of Gull et al.’s data mining approach and our linear mechanism is highly promising for further studies.

Our approach excluded relations and hence relevance of a set of artefacts given a specific artefact. Similarly, the relevance of a set of use cases given a specific use case and a set of developers given a specific developer was not considered. It is expected that considering these relations would provide more answers to the traceability information needs during collaborative software development setting. Finally, major ethical considerations exist in the real world use of our suggested techniques as it could be abused as the basis of capability judgment and reward structuring. This can be partially addressed by appropriate management attitude and also CRI’s facility to allow developers to switch monitors on and off at any stage.

References


Notes