Information and Software Technology 93 (2018) 163–185

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A systematic review of requirements change management

Shalinka Jayatilleke, Richard Lai*

Department of Computer Science and Information Technology, La Trobe University, Bundoora, Victoria 3086, Australia

ARTICLE INFO

Article history:
Received 4 May 2017
Revised 11 August 2017
Accepted 10 September 2017
Available online 12 September 2017

Keywords:
Requirements change management
Agile
Systematic review

ABSTRACT

Context: Software requirements are often not set in concrete at the start of a software development project; and requirements changes become necessary and sometimes inevitable due to changes in customer requirements and changes in business rules and operating environments; hence, requirements development, which includes requirements changes, is a part of a software process. Previous work has shown that failing to manage software requirements changes well is a main contributor to project failure. Given the importance of the subject, there's a plethora of research work that discuss the management of requirements change in various directions, ways and means. An examination of these works suggests that there's a room for improvement.

Objective: In this paper, we present a systematic review of research in Requirements Change Management (RCM) as reported in the literature.

Method: We use a systematic review method to answer four key research questions related to requirements change management. The questions are: (1) What are the causes of requirements changes? (2) What processes are used for requirements change management? (3) What techniques are used for requirements change management? and (4) How do organizations make decisions regarding requirements changes? These questions are aimed at studying the various directions in the field of requirements change management and at providing suggestions for future research work.

Results: The four questions were answered; and the strengths and weaknesses of existing techniques for RCM were identified.

Conclusions: This paper has provided information about the current state-of-the-art techniques and practices for RCM and the research gaps in existing work. Benefits, risks and difficulties associated with RCM are also made available to software practitioners who will be in a position of making better decisions on activities related to RCM. Better decisions will lead to better planning which will increase the chance of project success.

1. Introduction

Change is an intrinsic characteristic of the software engineering discipline compared to other engineering disciplines. In real-world scenarios, it is difficult to specify all the requirements for software as the need and the circumstance of the scenario is subject to change. Factors such as customer needs, market change, global competition, government policies, etc. contribute profoundly to the changing nature of requirements. The need for increasingly complex software is in high demand as organizations struggle to survive in a highly competitive market. Therefore, managing change in software development is not just important but crucial for the success of the final product.

Nurmuliani [1] defines requirements volatility as "the tendency of requirements to change over time in response to the evolving needs of customers, stakeholders, the organisation and the work environment". Requirements, in principle, are the needs and wants of the users and stakeholders of the system captured by an analyst through an elicitation process [2]. These requirements change throughout the system development and maintenance process, which includes the whole lifecycle of a system: requirement formation, analysis, design, evaluation and learning [1–15]. As this review progresses, we discuss in detail the factors that can cause these requirements changes. Therefore, requirements change management (RCM) can be defined as the management of such changing requirements during the requirements engineering process, system development and the maintenance process [2,5,16]. This definition of RCM is an adaptation of the definition provided by Sommerville [2] who states RCM is a process of "managing changing requirements during the requirements engineering process and system development".

* Corresponding author.
E-mail addresses: sejayatilleke@students.latrobe.edu.au (S. Jayatilleke), r.lai@latrobe.edu.au, lai@cs.latrobe.edu.au (R. Lai).

http://dx.doi.org/10.1016/j.infsof.2017.09.004
0950-5849/

Managing such evolving changes has proved to be a major challenge [12–15]. The consequences of unmanaged or improperly managed requirement changes can spell disaster for system development. These negative consequences can result in software cost and schedule overrun, unstable requirements, endless testing and can eventually cause project failure and business loss [1,17–23]. Therefore, the proper management of change can be both rewarding and challenging at the same time.

The research area of RCM is of importance to many parties as requirements change is a constant factor. Many research studies on have been conducted on improving RCM and many more have been conducted to look for answers in the knowledge gaps found in the current research. The main motivation of this research paper is to bring together the plethora of research work done in the area of RCM into one location. This will enable software practitioners and researchers alike a reference point in acquiring knowledge on the current practices, benefits, risks and difficulties associated with RCM. As a result, they can form realistic expectations before making decisions on activities related to RCM. Better decision making will lead to better planning which will increase the chance of project success. An equally important reason to conduct this research is to identify the knowledge gaps in the area of RCM. Given that a lot of research work has been done in this area, we felt it is important for us as well as other researchers to understand the future of RCM. Although this is a widely researched area, there are many gaps still remaining that once recognized and remedied could assist organizations immensely.

2. Research questions

To gain an understanding of current trends, practices, benefits and challenges in RCM, we formulated the following four questions;

RQ_1: What are the causes of requirement changes?

The motivation behind this question is to understand why requirement changes occur, which leads to the realization as to why this has been an evolving topic. To answer this question, we investigated various events and uncertainties that have been mentioned in literature. We also investigate whether there is any commonality between these events that would lead to a recognition pattern in predicting RCs.

RQ_2: What processes are used for requirements change management?

The motivation behind this question is to understand the various steps involved in managing RCs. To answer this question, we investigated the following: (1) recommendations for semi-formal methods of managing change; (2) formal process models available for RCM

RQ_3: What techniques are used for requirements change management?

The motivation for this question is to identify and understand the state-of-the-art techniques in managing major areas of the RCM process. To answer this question, we identify the main steps required to manage RC based on the answer to RQ_2 and then identify in the literature what techniques have been used in each of these steps.

RQ_4: How do organizations make decisions regarding requirements changes?

The motivation behind this question is to discover what factors are involved in making decisions regarding RCs at different organizational levels. To answer this question, we first identify the main levels of an organization and use the information available in the literature on RCM that can be mapped to each level.

3. Review approach

The systematic review was designed in accordance with the systematic review procedures and processes defined by Kitchenham [24,25]. According to Kitchenham [24], there are 10 sections in the structure of a systematic review: 1. Title; 2. Authorship; 3. Executive summary or abstract; 4. Background; 5. Review questions; 6. Review Method; 7. Inclusion and exclusion of studies; 8. Results; 9. Discussion; and 10. Conclusion. The first 5 sections have been covered so far. The review method comprises four sections: 1. Data search strategy; 2. Study selection; 3. Data extraction; and 4. Data synthesis. This section comprises the review method and the inclusion and exclusion of studies. The results, discussion and conclusion are presented in the next section.

3.1. Study objectives

As noted earlier, the objective of this literature review is to thoroughly study the background and existing methods in RCM and thereby provide a critical analysis of the relevant research work and identify future directions for improvement.

3.2. Selected sources

In order to carry out a comprehensive analysis, search strings were established by combining the keywords through the logical connectors “AND” and “OR”. The studies were obtained from the following search sources: IEEE, ACM, Science Direct (Elsevier), Springer, Wiley Inter Science, and Google Scholar. The quality of these sources guarantees the quality of the study.

3.3. Selected language

The English language is the most commonly used language in the world and most of the available research is written in English. Therefore, only papers which are written in English were selected for the literature review.

3.4. Data search

To answer the research question, we undertook the search using four steps;

Step 01 – Identify the fundamental areas to finalize the scope of the review.

Step 02 – Select key words/strings from the defined areas. Key words/strings were limited to seven (see Table 1).

Step 03 – Describe search expressions based on the first two steps i.e. [Expression = (A_1 OR A_2 OR A_3 OR A_4 OR A_5 OR A_6 OR A_7 OR A_8 OR A_9 OR A_{10} OR A_{11}) AND (B_1 OR B_2 OR B_3 OR B_4 OR B_5)].

Step 04 – Use the search expression in the libraries mentioned in the selected sources.

3.5. Study selection (Inclusion and exclusion of studies)

Once the research questions and the data search mechanism were defined, we started the process of selecting studies which fell under the defined scope and contained the keywords set out in the review process. As shown in category A of Table 1, the area of RCM has a lot of potential as change is a constant factor. As a result, our search yielded hundreds of research papers and studies. After screening these papers, we came to the conclusion that 28% (184) were relevant to the study.

Papers were excluded for a number of reasons related to format (editorial, seminar, tutorial or discussion), repetition, lack of peer

Table 1
Categories and keywords.

Category	Area	Keywords/strings
A	Requirement change management	A_1 – Requirement change/volatility/creep A_2 – Requirement change difficulties A_3 – Requirement change management A_4 – Requirement change management models/Processes A_5 – Requirement change identification/type A_6 – Requirement change analysis A_7 – Requirement change factors/causes A_8 – Requirement change decisions A_9 – Change impact analysis A_{10} – Agile requirement change management A_{11} – Requirement change cost estimation
B	Nature of study	B_1 – Case study B_2 – Experiment B_3 – Surveys B_4 – Industrial B_5 – Literature reviews

Table 2
Study selection process.

Analysis phase	Inclusion criteria	Number of papers
1. Initial search	• Papers written in English • Available online • Contain search keywords and strings	650
2. Scrutinizing titles	• Only published in journals, conferences, workshops and books • Not an editorial, seminar, tutorial or discussion	573
3. Scrutinizing abstract	• Experiments, case studies, literature reviews, industrial and surveys	340
4. Analysing introduction and conclusion	• Main contribution in the areas of search strings	230
5. Analysing main contribution	• Reported significant contribution • Originality of work • Sole focus related to the theme of this review study	184

Table 3
Classification of exclusion.

Exclusion criteria	No.
Paper format (editorial, seminar, tutorial or discussion)	95
Repetitions	43
Lack of peer review	75
Lack of a focus on RC and RCM	110
Redundancy	98
Lack of quality	45
Total	466

review, lack of a focus on RC and RCM, redundancy and lack of quality. Several papers appeared in more than one research repository. We eliminated the repetitions and only considered one instance of a paper. Details on repeated articles do not provide any significant information, except the names of the articles which have been published by more than one publishing authority (e.g. IEEE, ACM). As a result, we do not mention the names of the repeated articles which were found during the study selection process. In the initial phase, the extracted papers were independently reviewed by both authors based on the inclusion and exclusion criteria. In the secondary phase, both authors compared their outcome of their selection and through discussion, came to agreement on the inclusion and exclusion of papers. The overall inclusion process comprised five steps, as shown in Table 2. Table 3 provides details of the reasons for the exclusion of 466 papers.

Table 4
Data extraction process.

Aspects	Details
Study ID	Paper ID
Title	Title of paper
Authors	Names of authors
Publishers	Name of publishing authority
Publishing date	Date of publication
Causes of RCs	Factors that cause requirement changes
Study focus	Focus and perspective of paper
RCM processes/models	Processes/models listed for managing RC
RCM techniques	Techniques used for RCM (identification, impact analysis, cost estimation, etc.)
Reported challenges in RCM	Challenges and consequences associated with RCM
Decision making in RCM	Factors involved in decision making related to RCM
Study findings	Lessons learned from the paper
Knowledge gaps in RCM	Implications for future work

3.6. Data extraction

After completing the study selection process, we recorded basic information on each paper in data extraction form (refer to Table 4) to gather information on the causes of RCs, the study focus, RCM processes/models, RC identification, RCM techniques, reported challenges in RCM, decision making in RCM, study findings and knowledge gaps in RCM. The non-experimental models which presented a proposal without conducting experiments were also applied.

3.7. Data synthesis

Kitchenham [24,25] states that there are two main methods of data synthesis: descriptive (qualitative) and quantitative. The extracted data were analysed using a qualitative method to answer our research questions, which leads to a descriptive data synthesis. One of the co-authors of this paper has published qualitative systematic reviews [26,27] using similar techniques. The analysis used the constant comparison method [28] in comparing studies past and present in RCM. Using this method, we present the focus of the studies, the proposed methods, applicability to requirement change management, lessons learned from the studies and drawbacks and limitation of the studies.

4. Results for RQ_1: what are the causes of requirements changes?

It is anticipated that requirements will change during a project life cycle. Whilst this fact is a constant, delayed discovery of such changes poses a risk to the cost, schedule and quality of the software [3,29–31] and such volatility constitutes one of the top ten risks to successful project development [30–32]. Pfleeger [33] recommends that a method needs to be developed to understand and anticipate some of the inevitable changes during the development process in order to reduce these risks. The identification of factors that cause or influence requirements uncertainty is a necessity. The recognition of such factors will support requirements change risk visibility and also facilitate better recording of change data [30,31].

Change cause factors were collected using a key word search on academic papers, industry articles and books that deal with change management or requirement engineering. We used the search expressions A_1 OR A_2 OR A_3 OR A_5 OR A_7 (see Table 1).

Most literature extracted in this survey mentioned/indicated the reasons for requirement changes. However, it was deemed necessary to present these findings in a form that was meaningful rather than listing all the causes of RCs mentioned in the literature. Of the literature extracted, there were three studies that

formally classify the causes of RCs. Weiss and Basili [34] divide changes into two categories: error correction and modifications. This classification appears to be simplistic and categorising all the identified change causes may not create an in-depth understanding. Bano et al. [35] classifies change causes also under two categories; essential and accidental. They further classify the change causes based on their origin: within the project, from the client organization and from the business environment. McGee and Greer [30,31] use five areas/domains to classify change causes. For this survey, we use the classification presented by McGee and Greer as it has a more comprehensive categorization. The five change areas are: external market, customer organization, project vision, requirement specification and solution. Within the five change areas, they distinguish between two causes of change: trigger and uncertainty [30]. The difference between these two categories is that an event can cause a change without pre- or post-uncertainty. However, uncertainty cannot cause a change to occur without an event that is triggered to manage the risk of the uncertainty. The factors that were identified as causes of requirements change were sorted into five areas as follows:

(i) **Change area: External market**
In this category, the changes to the requirements are triggered by the events and uncertainties that occur in the external market which also include stakeholders. These stakeholders include parties such as customers, government bodies and competitors. Therefore, events such as changes in government policy regulations [36–38], fluctuations in market demands [1,37–39] and response to competitors [15,37,40,41] can be considered. Also, uncertainties such as the stability of the market [15,42] and the changing needs of the customers [15] are also part of this category.

(ii) **Change area: Customer organization**
In this category, changes to the requirements are triggered by the events and the uncertainties that arise from a single customer and their organizational changes. Although the changes occur within the customer's organization, such changes have a tendency to impact the needs of the customer and as a result, impact the design and requirements of the software project. Therefore, events such as strategic changes within the organization [4], restructuring of the organization [1,36,38,39,43], changes in organizational hierarchy [15,37,44] and changes in software/hardware in the organization should be considered. The stability of the customer's business environment can create uncertainties that may lead to changes and these are also part of this category.

(iii) **Change area: Project vision**
In this category, the changes to the requirements are triggered by changes in the vision of the project. These changes are in response to a better understanding of the problem space from a customer point-of-view and the emergence of new opportunities and challenges. Events such as improvements to business processes [2,37], changes to business cases due to return on investment [4], overrun in cost/schedule of the project [36,39], identification of new opportunities [36] and more participation from the stakeholder [38] should be considered. Uncertainties, such as the involvement of all stakeholders [37,43–45], novelty of application [37,46], clarity in product vision [37,38,45,47], improved knowledge development team in the business area [44,46], identification of all stakeholders [43,45], experience and skill of analyst [37,44,47,48], size of the project [2,44,49] can also cause changes under this category.

(iv) **Change area: Requirement specification**
In this category, changes in the requirements are triggered by events and uncertainties related to requirement specification. These trigger events are based on a developer's point-of-view and their improved understanding of the problem space and resolution of ambiguities related to requirements. Events such as increased understanding of the customer [2,36,37,49,50], resolution of misunderstandings and miscommunication [15,51,52] and resolution of incorrect identification of requirements [1] can be considered as change triggers. Uncertainties, such as the quality of communication within the development team [4], insufficient sample of user representatives [4], low staff morale [48], quality of communication between analyst/customer [37,42,44,49], logical complexity of problem [44,46,49], techniques used for analysis [2,36,37,39,48], development teams' knowledge of the business area [44,46], involved customers' experience of IT [46], quality of requirement specification [4], and the stability of the development team [4] can contribute towards change under this category.

(v) **Change area: Solution**
In this category, changes in the requirements are triggered by events and uncertainties related to the solution of the customer's requirements and the techniques used to resolve this. Events such as increased understanding of the technical solution [4], introduction of new tools/technology [5,15,36–38,41,43,53] and design improvement [15,36,51] should be are considered as change triggers. Technical uncertainty and complexity can also be considered under this category as a cause of change [4].

Table 5
Comparison between classifications.

Bano et al.'s Classification [35]	McGee and Greer's Classification [30]		
Essential	External market	Customer organization	
Accidental	Project vision	Requirement specification	Solution

The five change areas listed above can be mapped to the classification proposed by Bano et al. [35]. The terms *essential* and *accidental* were initially introduced by Brooks [54]. According to Bano et al. [35], change causes under the essential category are those that are inherent in nature and cannot be controlled i.e. "fluctuating market demand" cannot be controlled or avoided by the development team or the organization. In comparison, accidental causes can be controlled and avoided i.e. "overrun in cost/schedule of the project" can be avoided or at least controlled by putting better techniques and mechanisms in place. Being able to categorize change causes under these two categories has added benefits in managing RCs. With essential causes, the focus should be to deal with their impact and therefore use techniques that will reduce time and effort for their management. With the accidental causes, the focus should be to use techniques that avoid such occurrences. Table 5 shows how these five categories in McGee and Greer's classification [30] can be mapped to Bano et al.'s classification [35] of essential and accidental categories.

Key findings of RQ_1

Given that RC is an inevitable occurrence in any development project, it is beneficial to identify which factors can cause these changes. The knowledge gained through such findings will enable all stakeholders of a project to better manage the changes when they occur, develop systems based on the changes, and anticipate certain changes. Based on the discussion formulated for RQ_1, the following are the key findings:

1) The factors that cause RCs can be divided into two categories: change trigger events; and uncertainties.

2) In reality, it is difficult to determine whether change happens as a result of one or both. In a practical sense, it is not important that the causes of the changes are divided into these two categories, as long as they are identified.
3) These identified changes can be categorised into five areas: external market; customer organization; project vision; requirement specification; and solution.
4) These five areas were identified by observing the characteristics of the change events and the uncertainties discussed in the literature. For example, any change factor that was part of the external environment of the organization, such as competitors, government regulations, etc. was categorised as the external market.
5) These five areas can be divided into two categories: essential and accidental. Based on this division, development teams can be proactive in managing such changes.
6) Based on the location in the life cycle of the software project, the above information can be meaningful for anticipating what factors may cause change and as a result will lead to better planning that will ensure a better success rate for the project.

5. Results for RQ_2: what processes are used for requirements change management?

In order to answer RQ_2, the following sections discuss various processes suggested for managing RC and the process models that are dedicated for RCM. We used the search expressions A_3 OR A_4 OR A_6 OR A_9 OR A_{10} (see Table 1) to extract the relevant literature.

5.1. Semi-formal methods available for requirements change management

Change is considered to be an essential characteristic of software development and successful software has to be adapted to the requirements of its customers and users [5,55,56]. Thus RCM has become a significant activity, which is undertaken throughout the development of the software and also during the maintenance phase. Given the significance of this activity, it is unlikely that change management is undertaken in an ad-hoc manner. According to Sommerville [2], the process of RCM "is a workflow process whose stages can be defined and information flow between these stages partially automated". Having a proper process for RCM is linked with both improvement in the organizational processes and the success of software projects [5,6,57]. We have identified four (i–vii) academic works that refer to establishing semi-formal methods for managing change.

(i) *Proposal: Leffingwell and Widrig* [58]
This is a five-step process for managing change. The process is as follows:
1. Recognize that change is inevitable, and plan for it.
2. Baseline the requirements.
3. Establish a single channel to control change.
4. Use a change control system to capture changes.
5. Manage change hierarchically.

The process begins with a change management plan which recognizes that change is unavoidable. Requirements are therefore baselined for change control and any proposed RC is then compared with the baseline for any conflicts. In the third step, a change authority or change decision maker is established. For small projects, this would be a project manager while for larger systems, the responsibility would be handed to a change control board. In both cases, the decision is based on impact analysis. In the decision-making process, it is recommended that input from various stakeholders, such as customers, end-user, developers, testers, etc. should be taken into consideration. To be able to make an informed decision, the impact analysis should capture the effect of the change on cost, functionality, customers and external stakeholders. Also to be considered is the destabilization of the system, which can occur due to the implementation of the change. The decision which is taken should be communicated to all the concerned parties. The fourth step refers to establishing a system that can be used to capture the changes effectively. This could be either paper-based or electronic. The ripple effects of the change are to be managed in a top-down order.
Limitations of the proposal:
According to Leffingwell and Widrig [58], this process should enable software practitioners to identify changes that are "both necessary and acceptable". However, it is not mentioned in this work what steps are to be taken to decide if a particular change is both necessary and acceptable. Similarly, no specific details are given as to how to calculate the impact on cost, functionality, customers and external stakeholders. In this sense, these steps only form a basic understanding of what needs to occur in handling a change.

(ii) *Proposal: El Emam* et al. [57]
This process focuses on the preliminary analysis of change management. Two inputs are considered in order to conduct this process, the technical baseline and any comments made by stakeholders, such as customers, end-users, the development team, etc. The decision-making process involves a change control board as this change management process is prescribed for large systems. The technical baseline is essentially the system requirement specification document. The change management process has the following four phases:
1. Initial issue evaluation
2. Preliminary analysis
3. Detailed change analysis
4. Implementation

In the first step, the comments gathered from the stakeholders are validated and entered into a database as change requests. If a change request addresses a problem that is within the scope of the technical baseline, and has not been addressed before, a change proposal will be generated. In the second step, an analysis plan is formulated which describes the problem of the change proposal in detail. If this plan is approved by a change control board, then many potential solutions will be developed, from which one will be selected for implementation. This solution then needs to undergo further approval. In the third step, the solution approved by the preliminary analysis report is further analysed against the technical baseline to determine the impact on the system in detail and the changes required. In the last step, the technical baseline is modified according to the change proposal and the change request is closed.
Limitations in the proposal:
The use of these steps is limited to large projects. Furthermore, it is not clear on what basis the different alternative solutions are assessed and what exactly is the decision-making process in the second step. Given that this process is conducted at an initial stage of the development process, there is no access to the code. Therefore, a possibility exists that these changes may cause issues at a code level.

(iii) *Proposal: Kotonya and Sommerville* [59]
The authors emphasize the importance of having a formal process for change management to ensure the proposed changes continue to support the fundamental business goals. They [59] indicate that such a process ensures that similar information is collected for each proposed change and that overall judgements are made about the costs and benefits of

such changes. A three-step change management process is proposed in [59] as follows:

1. Problem analysis and change specification
2. Change analysis and costing
3. Change implementation

In the first step, a problem related to a requirement or a set of requirements is identified. These requirements are then analysed using the problem information and as a result, requirements changes are proposed. In the second step, the proposed changes are analysed to determine the impact on the requirements as well as a rough estimation of the cost in terms of money and time that is required to make the changes. Finally, once the change is implemented, the requirement document should be amended to reflect these changes and should be validated using a quality checking procedure.

Limitations in the proposal:

The cost estimation carried out in the second step has a component of seeking customer approval. The information which is lacking at this stage is the decision factors that are considered by the software practitioners and the customers in order to approve or disapprove a proposed change. The negotiation process with customers in relation to accepting or rejecting a proposed change as indicated in [59] is based on cost and there is no indication that the risks associated with implementing the change were considered.

(iv) *Proposal: Strens and Sugden* [7]

The change analysis process introduced in [7] is based on two analysis methods, namely sensitivity analysis and impact analysis. According to Strens and Sugden [7], sensitivity analysis is used to predict which requirements and design areas have the highest sensitivity to changes in requirements while impact analysis is used to predict the consequences of these changes on the system. The main outcome of this analysis is to reduce the associated risks in accepting and implementing RCs. The process is as follows:

1. Identify the factors which are the cause of change.
2. Identify those requirements which are highly affected by the change (this information is acquired from the previous history of requirements or intuition).
3. Identify the consequences of these changes - impact analysis
4. Undertake change analysis on other requirements, design, cost, schedule, safety, performance, reliability, maintainability, adoptability, size and human factors.
5. Decide on and manage changes.

Limitations in the proposal:

It is important to perform change analysis, however there is no clear explanation as to how the impact analysis is to be carried out for the elements mentioned in step four and how these factors will be "equated". It is also difficult to determine the ripple effect of the changes, given that there is no identification of the implementation part and the test documents to be modified.

(v) *Proposal: Pandey* et al. [60]

The authors propose a model for software development and requirements managements. There are four phases in this process model: requirement elicitation and development, documentation of requirements, validation and verification of requirements and requirements management and planning [60]. The management of RCs are controlled by the requirement management and planning phase. However, according to the full process model, the activities of this phase are interrelated with the other phases. The process is as follows:

1. Track the changes of the agreed requirements.
2. Identify the relationship between the changing requirements with respect to the rest of the systems.
3. Identify the dependencies between the requirements document and other documents of the system.
4. Decision on the acceptance of the change(s).
5. Validation of change request.
6. Maintain an audit trail of changes.

Limitations in the proposal:

Although a comprehensive set of steps is described, the paper does not discuss specific schematics in executing these steps. Dependencies are considered but there is no indication of further impact analysis. It is not clear how decisions will be made in terms of accepting or rejecting a change as the impact analysis phase is not clearly discussed. There is also no indication of consideration of the cost or risks associated with implementing the change.

(vi) *Proposal: Tomyim and Pohthong* [8]

The method introduced by Tomyim and Pohthong [8] for RCM usesUML for object-oriented development. The authors justify the use of UML due to the complexity of the many views and diagrams it produces, thereby adding more complexity in managing change. Therefore, a need arises for a process to manage the changes better using UML. The business model used in this method consists of two procedures: systems procedure (SP) and work instructions (WI). The SP explains the business operation from the beginning of a task until the end of the business process. The WI explain the way to operate any single task step by step. The method comprises the following steps:

1. Identify the change request.
2. Identify the related SP and WI.
3. Analyse the impact on the system and report on the impacted artefacts.
4. Make a decision based on the impact.

Limitations in the proposal:

The paper provides several sets of diagrams that represent the activities carried out but does not provide details of the execution of the steps. A decision on the implementation of the change is solely based on the impact analysis. This may be problematic if change priorities and costs/effort elements are not taken into consideration.

(vii) *Proposal: Hussain* et al. [61]

The method proposed by Hussain et al. [61] is based on the need to manage informal requirements changes. Such requirements are internally focused, potentially subversive to the development process and therefore harder to manage [61]. According to the authors, there are many reasons for informal changes, some of which are: prematurely ending requirement engineering activities [62]; attempting a requirements 'freeze' earlier than usual in a project [58]; as a consequence of work hidden by managers to get something developed by making ad hoc decisions and bypassing time consuming formalities [63]; additions made without the consideration of delay in the schedule and project cost [64]; and failure to create a practical process to help manage changes [58]. Therefore, the authors suggest that there is as much a need for a method for managing informal requirement changes as for formal requirement changes. The method comprises the following steps:

1. Identify informal requirement change.
2. Analyse the impact of change.
3. Negotiate the change with stakeholders.
4. If accepted, decide on whether to include in current phase or next.

Limitations in the proposal:
The process is not very different from formal change management techniques. The negotiation component after the impact analysis is a slight variation from the norm, however it does not explicitly explain how the negotiation is done. The main component considered for negotiation is the impact analysis. However, the proposed method does not disclose how the impact analysis is conducted and what is considered for the impact analysis i.e. affected components, cost, effort, etc.

5.2. Formal process models available for requirements change management

The processes introduced above are not formalized models for managing RC. This section introduces several RCM process models. These models facilitate communication, understanding, improvement and management of RCs. Typically, a process model includes activities, who is involved (roles) and what artefacts are to be used [9,65].

The activities of a change management process model are the actions performed during the RCM process that have a clearly defined objective, such as determining the change type which is a part of change identification [2,66,67]. The identification of the roles in these process models define the responsibilities attached to each role. For example, if the role of the customer is defined by the process model, this means the responsibilities need to be shared with the customer's organization and its representatives. The artefacts are documents and parts of the product created, used and/or modified during the process [2,66,67]. By identifying these artefacts as part of the RCM process, this makes the management of change more efficient due to the early detection of what documentation is going to be affected by the change.

Based on the information given in [16] and by individually studying several change management process models, ten such models [10,19,58,68–72] were selected from the literature. Table 6 compares these models based on their activities, roles and the artefacts used. There are certain limitations to these models, which are detailed in Table 7.

5.3. Agile methods available for requirements change management

One of the most important aspects of agile methods is that change is a built-in aspect of the process [73]. Since software development is done in small releases, agile methods tend to absorb RCM into these small iterations. The processes for managing change can neither be categorised as semi-formal nor formal. Because of the frequent face-to-face communication between the development team and the client, the main reported changes in requirements are to add or to drop features [74,75]. The clarity gained by clients helps development teams to refine their requirements, which results in less need for rework and fewer changes in subsequent stages [75]. There are several agile development models used, the most popular being Extreme Programming, Scrum, RUP, Lean, Plan-driven methods, Iterative and Incremental model and the General Agile model [76]. Regardless of the agile style of development used, the underlying processes have an inbuilt capacity to manage requirement change. We were able to extract 10 such processes that deal with RCM as follows:

1. *Face-to-face communication* [74,75,77–79]*:*
   This is a frequent characteristic activity between the client and the development team [74,77,78]. There is minimal documentation using user stories which does not require long and complex specification documents. The frequency of this activity helps clients to steer the project in their own direction as the understanding of needs tend to develop and requirements evolve [75,79]. Therefore, the possibility of dramatic and constant changes is reduced and the changes that do arise are easily communicated due to the frequent communication between all the stakeholders.
2. *Customer involvement and interaction* [73–75,78,80]*:*
   In relation to some of the change cause factors listed in RQ1, there are several elements to the involvement of the customer organization. In agile methods, there is a need to identify customers or representatives from the client organization for frequent collaboration to ensure that requirements are appropriately defined [78,80]. As discussed above, this leads to a better understanding of the system requirements and makes the inclusion of changes less complicated.
3. *Iterative requirements* [74,78,79]*:*
   Unlike traditional software development, requirements are identified over time through frequent interactions with the stakeholders (face-to-face communication) [78]. The frequent interactions make this an iterative process. This allows the requirements to evolve over time with less volatility [74]. This gradual growth of requirements leads to less requirement changes and far less time spent managing such changes.
4. *Requirement prioritisation* [75,78–80]*:*
   This is a part of each iteration in agile methods [75]. In each iteration, requirements are prioritised by customers who focus on business value or on risk [78,80]. In comparison, traditional requirements engineering is performed once before development commences. Iterative requirement prioritisation helps in RCM by comparing the need for the change with the existing requirements and then placing it an appropriate priority location for implementation. As this is done frequently, understanding the need for the change and its priority becomes a much easier process.
5. *Prototyping* [74,78,81]*:*
   This is a simple and straightforward way to review the requirements specification with clients, so that timely feedback is obtained before moving to subsequent iterations [81]. This assists in RCM by identifying what new additions are required and what existing requirements are to be changed or removed. This reduces complex and/or frequent RCs in subsequent iterations.
6. *Requirements modelling* [82,83]*:*
   One technique used in requirement modelling in agile methods is goal-sketching, which provides goal graphs that are easy to understand [83]. This activity is also iterative and the goals are refined during each iteration [82]. This helps in RCM by creating unambiguous requirements that have a clear purpose, reducing the need for change during subsequent iterations.
7. *Review meetings and acceptance tests* [78,84]*:*
   During review meetings, the developed requirements and product backlogs are reviewed to ensure user stories are completed. Acceptance tests are similar to a unit test, resulting in a "pass" or a "fail" for a user story. These tests increase the collaboration of all the stakeholders as well as reduce the severity of defects. One of the reasons for RC is defects in the end product. This practice effectively reduces the need for changes due to such defects.
8. *Code refactoring* [85]*:*
   This process is used for revisiting developed code structures and modifying them to improve structure and to accommodate change [86]. This practice deals with requirement volatility in subsequent stages of agile development [85]. Therefore, in terms of RCM, the method allows flexibility in handling dynamically changing requirements.
9. *Retrospective* [78,79,87]*:*
   This process comprises meetings which are held after the completion of an iteration [87]. These meetings often review the

Table 6
Comparison of RCM process models.

Areas of change management	Model elements	Process models									
	Activities	Leffingwell and Widrig [58]	Olsen [68]	V-Like [69]	Ince's [69]	Spiral [69]	NRM [10]	Bohner [72]	CHAM [70]	Ajila [71]	Lock and Kotonya [19]
Change identification	Plan of change	Y							Y		
	Problem understanding			Y		Y		Y			
	Determine type of change								Y		
Change analysis	Change impact on functionality	Y		Y						Y	Y
	Manage change hierarchy	Y									
	Solution analysis			Y		Y			Y		Y
Change effort estimation	Change impact on cost	Y							Y		Y
	Estimate effort								Y		
	Cost benefit analysis										Y
Other	Negotiation process	Y							Y		Y
	Update document	Y			Y				Y		
	Change implementation		Y	Y	Y		Y	Y		Y	Y
	Verification		Y				Y		Y		Y
	Validation			Y	Y		Y			Y	Y
	Document impact, cost and decisions										Y
Artefacts		Baseline, Vision document, Use case model, software requirement specification	N/A	Modification report, Problem statement	Problem statement, Change authorization note, Test record	Implementation plan, Release plan	N/A	N/A	N/A	N/A	Vision document, Use case model, software requirement specification, problem statement, change request form
Roles		Customer, developer, end user, change control board	N/A	Maintenance organization	Customer, Developer, Change control board	N/A	N/A	N/A	Customer, Developer, End user	N/A	Customer, Developer, End user

Table 7
Limitations of RCM process models.

Model	Limitations
Leffingwell and Widrig [58]	Implementation of change is missing. Verification is not available and therefore not able to ensure the stability of the system post-change. Documentation in the form of change requests and decisions are also missing which contributes to poor management and future decision making.
Olsen [68]	Does not explicitly mention if there is any update to documents to keep track of the changes and also, there is no indication of the artefacts used and who is involved in the management process.
V-Like [69]	Two key elements are missing, cost estimation and impact analysis.
Ince's [69]	The decision-making process is unclear. Verification is not done.
Spiral [69]	Similar to Ince's model, there is a lack of decision making and no verification. Does not mention who needs to be involved in this process.
NRM [10]	Activities are at a very abstract level. Given that no artefacts and roles are mentioned, it is difficult to make use of this model in practice.
Bohner [72]	A key element that is missing is the analysis of impact, which is a major part of the decision-making process.
CHAM [70]	Although cost and effort is estimated, there is no analysis of impact on functionality which is an important factor for decision making. The artefacts to be used are also not mentioned.
Ajila [71]	There is no estimation of cost or effort. artefacts and roles are also not mentioned.
Lock and Kotonya [19]	No aspect of change identification, which is critical in understanding the change.

Table 8
Challenges in traditional RCM resolved by Agile approaches.

Challenges in traditional RCM approaches	Solutions provided by Agile approaches
Communication gaps and lack of customer involvement causing ambiguous requirements	Frequent face-to-face communication, customer involvement, and iterative requirements
Changes that occur due to over scoping which is a result of communication gaps and changes after finalizing project scope	Continuous customer involvement, iterative requirements, and prototyping
Change validations	Requirement prioritisation through iterative processes, prototyping, and review meetings and acceptance tests

Table 9
Challenges in Agile RCM.

Agile technique	Challenges
Face-to-face communication	The frequency of the communication depends on the availability and willingness of the team members. Customers may not be familiar with this agile technique and could be wary of it.
Customer involvement	Failure to identify needed/correct customer representatives can lead to disagreements and changing viewpoints.
Requirement prioritisation	A focus only on business value when prioritising requirements/changes can be problematic as there can be other factors to consider.
Prototyping	Problems may occur if there a high influx in client requirements at a particular iteration.
Code refactoring	Can generate code wastage, which increases the project cost.
User stories and product backlog	This is the only documentation used in agile methods as minimal documentation is a characteristic. This becomes a problem when there is a communication lapse or project representatives are unavailable. It is also problematic when requirements must be communicated to stakeholders in distributed geographical locations.
Budget and schedule estimation	Due to the nature of incorporating RCs in subsequent iterations, it is not possible to make upfront estimations, which can result in budget and schedule overruns.

work completed so far and determine future steps and rework. In terms of RCM, this provides an opportunity to identify changes.

10. *Continuous planning* [79]
This is a routine task for agile teams where the team never adheres to fixed plans but rather adapts to upcoming changes from customers. In RCM, this facilitates changing requirements in the later stages of the project.

Agile development, different to traditional software development encourages change in every iteration. The iterative and dynamic nature of this development method promotes constant feedback and communication between the stakeholders. Therefore, the management of changes is continuous during the iterations. We have identified some of the challenges that are inherent in traditional methods of RCM that can be resolved by agile methods. This is discussed in Table 8. Whilst agile methods seem to have a very efficient way of managing change, we were able to identify some practical challenges in some of the techniques discussed above. The challenges are presented in Table 9.

Key findings of RQ_2

Similar to any other activity in the software development process, RCM has also been described in related work as an activity that needs to be carried out in defined steps. Based on the discussion that formulates the answer for RQ2, the following are the key findings:

1) Academic work has identified that it is important to establish a process for managing change where establishing and practicing a defined process for RCM is attached with benefits, such as the improvement of organizational processes and an increase in the predictability of projects.
2) In terms of traditional software development, two different approaches were investigated, namely: 1) recommendations for semi-formal methods of managing change; and 2) the formal process models available for RCM.
3) With semi-formal methods, it became evident that different academic work took different approaches and elements, and recommended different steps for managing change, which resulted in no consensus on the elements.
4) However, based on the activities on which the elements focused, we were able to identify three areas of management: change identification; change analysis; and change effort estimation.
5) These three areas were then applied to the ten formal process models of RCM found in the literature. Using this classification, we were able to identify certain commonalities between the process models, as illustrated in Table 6.

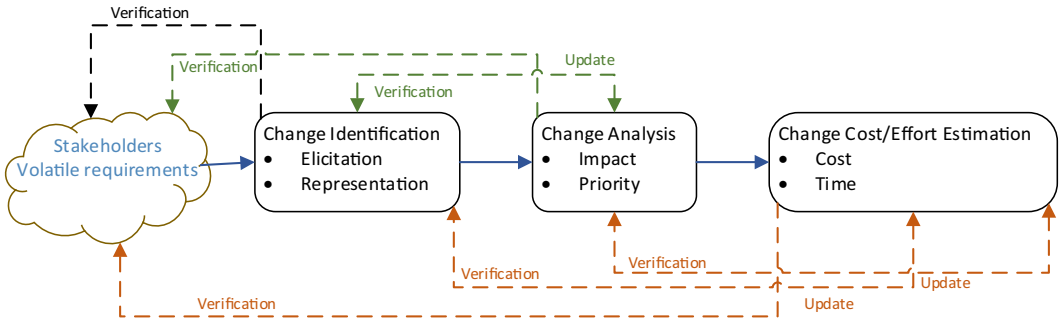

Fig. 1. Change management process.

6) The formal process models have three distinct sections: activities – the actions/steps taken in managing change; roles – the stakeholders involved in carrying out the activities; and artefacts – the documents needed in some of the activities (see Table 6).
7) We were also able to identify the limitations in both the semi-formal methods as well as the formal models.
8) Given the popularity of agile development in the recent past and present, several processes were identified that deal with RCM. Through this identification, we were able to discuss how agile methods can address some challenges in traditional RCM and also the challenges in agile RCM.

6. Results for RQ_3: what techniques are used for requirements change management?

The information gathered in RQ_2 will be used to formulate a framework to answer this question. Examining the processes introduced in RQ_2 as a whole, we have identified three key areas of a practical approach to managing change. Fig. 1 illustrates these areas i.e. change identification, change analysis and change cost estimation. It is important to understand how these areas can be practically implemented and what best practices are available in an organizational setting. As shown in Fig. 1, none of these areas are standalone. They need to communicate with each other in terms of updates and verifications. The reason for this is that each area has the ability to feed information to another area. For example, although change analysis can be undertaken once the change has been identified, the cost estimation may provide additional information for the analysis step that may not have been identified previously. A good RCM process does not have steps that are stand alone, rather they are interconnected with information following to and fro from the steps. We used the search expressions A_4 OR A_5 OR A_6 OR A_7 OR A_8 OR A_9 OR A_{10} OR A_{11} (see Table 1) to extract relevant literature.

6.1. Change identification

Change identification stems from several processes identified in RQ_2 [57–59]. This step is important for the rest of the management process as the steps to follow will be based on the correct identification of the problem space as well as the change requirement. According to Fig. 1, the change management process starts with change identification. Within this identification, there are two major activities, i.e. change elicitation and change representation. In order to ensure the correct elicitation of changes, the change requirements need to be identified.

The correct elicitation should then lead to identifying further details of the change and if possible, where in the system the change has to be made. This signifies the representation part of the identification step. In most situations, the personnel involved in this step will need to have continuous communication with the stakeholders in order to verify that identification is done correctly, as illustrated in Fig. 1. Through the literature, we identified two methods of change identification: taxonomies and classification. The following sections describe these two methods and several other methods that do not fall under these categories.

6.1.1. Through taxonomies

1) Research analysing change uses a plethora of techniques in order to build a taxonomy that can be used to identify changes as well as their impact. One such mechanism is the use of requirement engineering artefacts, such as use cases. The research done by Basirati et al. [88] establishes a taxonomy of common changes based on their observation of changing use cases that can then be used in other projects to predict and understand RCs. They also contribute to this research space by identifying which parts of use cases are prone to change as well as what changes would create difficulty in application, contributing also to the impact analysis of change.
2) The taxonomy developed by Buckley et al. [89] proposes a software change taxonomy based on characterizing the mechanisms of change and the factors that influence software change. This research emphasizes the underlying mechanism of change by focusing on the technical aspects (i.e. how, when, what and where) rather than the purpose of change (i.e. the why) or the stakeholders of change (i.e. who) as other taxonomies have done. This taxonomy provides assistance in selecting tools for change management that assist in identifying the changes correctly.
3) McGee and Greer [4] developed a taxonomy based on the source of RC and their classification according to the change source domain. The taxonomy allows software practitioners to make distinctions between factors that contribute to requirements uncertainty, leading to the better visibility of change identification. This taxonomy also facilitates better recording of change data which can be used in future projects or the maintenance phase of the existing project to anticipate the future volatility of requirements.
4) Gosh et al. [11] emphasize the importance of having the ability to proactively identify potentially volatile requirements and being able to estimate their impact at an early stage is useful in minimizing the risks and cost overruns. To this effect, they developed a taxonomy that is based on four RC attributes i.e. phases (design, development and testing), actions (add, modify and delete), sources (emergent, consequential, adaptive and organizational) and categories of requirements (functional, non-functional, user interface and deliverable).
5) The taxonomy established by Briand et al. [90] is the initial step in a full-scale change management process of UML

Table 10
Direction is change classification.

Direction	Parameters	Comment
Type [11,92–97]	Add, Delete, Modify	The most common way of classifying change.
Origin [11,38,98]	Mutable, Emergent, Consequential, Adaptive, Migration	Derived from the places where the changes originated from.
Reason [92,93,99]	Defect fixing, Missing requirements, Functionality enhancement, Product strategy, Design improvement, Scope reduction, Redundant functionality, Obsolete functionality, Erroneous requirements, Resolving conflicts, Clarifying requirements, Improve, Maintain, Cease, Extend, Introduce	Helps determine the causes of change and understand change process and related activities.
Drivers [100]	Environmental change, RC, Viewpoint change, Design change	Helps change estimation and reuse of requirements.

models. In their research, they establish that change identification is the first step in the better management of RCs. The classification of the change taxonomy is based on the types of changes that occur in UML models. They then use this taxonomy to identify changes between two different versions of UML models and finally to determine the impact of such changes.

6.1.2. Through classification

There are many benefits of using a classification, the main benefits being to manage change to enable change implementers to identify and understand the requirements of change without ambiguity [91,92]. The classification of RC has been studied in various directions. Table 10 lists the different directions that have been the subject of academic studies.

6.1.3. Other change identification methods

1) Kobayashi and Maekawa [10] proposed a model that defines the change requirements using the aspects where, who, why and what. This allows the system analyst to identify the change in more detail, resulting in better impact identification as well as risk and effort estimation. This method consists of verification and validation and can be used to observe the RCs throughout the whole lifecycle of the system.
2) The change identification method usually has a pre-established base upon which its semantics are built. Ecklund's [101] approach to change management is a good example of this. The approach utilizes use cases (change cases) to specify and predict future changes to a system. The methodology attempts to identify and incorporate the anticipated future changes into a system design in order to ensure the consistency of the design.

6.1.4. Change identification through agile methods

Unlike traditional requirement engineering methods, agile software development welcomes changes in various stages [75]. As discussed in RQ2, changes can be identified in several different phases of the development process. Table 11 presents the different phases of agile development that contribute to the identification of RCs, the challenges faced and solutions suggested by literature. The techniques given in the table have been described in detail in RQ2 (see Section 5.3).

6.2. Change analysis

Once a change has been identified, it needs to be further analysed to understand its impact on the software system so that informed decisions can be made. One of the key issues is that seemingly small changes can ripple throughout the system and cause substantial impact elsewhere [104]. As stated in the literature, the reason for such a significant impact is that the requirements of a system have very complex relationships [105–109]. Therefore, the way to realise this is to undertake change impact analysis, which according to Bohner [110] is defined as "the activity of identifying the potential consequences, including side effects and ripple effects, of a change, or estimating what needs to be modified to accomplish a change before it has been made". Change impact analysis provides visibility into the potential effects of the proposed changes before the actual changes are implemented [104,110]. The ability to identify the change impact or potential effect will help decision makers to determine the appropriate actions to take with respect to change decisions, schedule plans, cost and resource estimates.

6.2.1. Traceability issues and solutions

Given that the complex relationships between requirements are the key reason for impact analysis, most methods for impact analysis use requirement traceability as their focal point. Requirement traceability is defined as "the ability to describe and follow the life of a requirement in both a forward and backward direction (i.e. from its origins, through its development and specification to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases)" [111]. Although traceability has been defined by many scholarly articles, the above definition was selected as the most comprehensive because it describes both pre- and post-traceability and is used by many other scholarly articles [112–121] for the same purpose.

Although traceability is one of the best ways to track the impact of RCs, many scholarly works discuss the challenges in maintaining traceability. Tables 6 and 7 detail the issues in traceability and the solutions that have been provided. The solutions in Table 12 have not been verified by industry while the solutions in Table 13 have.

It is important to note that the solutions proposed might not be suitable for all types of organisations, however, some basic guidelines can be outlined.

i. The identified issues can act as a guideline to understand the challenges that might arise when creating and maintaining traceability and therefore improve the predictability of the traceability issues.
ii. The cost of traceability for a specific project will be concentrated on that project whilst its benefits (value) will span over and beyond the said project. The downside of this outcome is that it may hinder the motivation of a project team to work with traceability as the benefits are not realized immediately and therefore could be the cause of many of the challenges identified in Tables 6 and 7.

6.2.2. Use of traceability and other methods for impact analysis

According to Fig. 2, there are three sets of objects that can be impacted by a change: starting impact set (SIS), estimated impact set (EIS) and actual impact set (AIS).

- SIS is the set of objects that are thought to be initially impacted by the change
- EIS is the set of objects estimated to be impacted after further analysis
- AIS is the set of objects that are actually modified as a result of the change

Table 11
Change identification through agile methods.

Agile technique	Challenge(s)	Solutions
Face-to-face communication [74,75,77–79]	The success rate of the change identification at this stage is dependent on customer availability. However, this dependency is often unrealistic and a challenge as confirmed by studies [78,102]	In practice, teams have surrogates or proxy customers to play the role of real customers [80] or use the "onsite developer" by moving a developer representative to the customer site [103].
Iterative requirements [74,78,79]	Can create budget and schedule overruns as initial estimations will always change when requirements are added or removed during the iterations [78].	Inayat et al. [75] suggest frequent communication to identify as many requirements as possible at early iterations to keep these overruns to a minimum.
Prototyping [74,78,81]	Given that this is a review phase of development, the client may have a large number of changes to be included based on the prototype. This can create schedule overruns [75].	This can be mitigated somewhat, through frequent communication and high customer involvement and interaction in stages prior to prototyping [75].
Review meetings and acceptance tests [78,84]	Similar to the challenges of prototyping where there could be an influx of changes [84]. Also, if the product backlog is not maintained in detail, finding information related to changes made during the iterations will also be challenging.	Denva et al. [80] suggest maintaining a detailed artefact called delivery stories, in addition to user stories. These help developers make the right implementation choices in the coding stage of a sprint.
Retrospective [78,79,84]	If there are many changes identified in completed user story at this stage, there will be a considerable amount of rework to be done, causing budget and schedule overruns [75].	Increased customer involvement and interaction in the stages prior to completion of a user stories is essential [75].

Table 12
Traceability issues and their solutions (not verified).

Scholarly work	Issues in traceability	Solution (not verified by industry)
Arkley and Riddle [122]	Requirement traceability does not offer immediate benefit to the development process.	Traceable development contract.
Cleland-Huang et al. [123]	Informal development methods, insufficient resources, time and cost for traceability, lack of coordination between people and failure to follow standards.	Event-based traceability
Cleland-Huang et al. [124]	Lack of coordination between team members. Developers think that traceability costs more than it delivers. Excessive use of traceability generates more links which are not easy to manage.	Traceability for complex systems frameworks.
Cleland-Huang et al. [125]	Manual construction of a requirement traceability matrix is costly.	Dynamic retrieval methods are used to automate the generation of traceability links
Gotel and Morris [126]	Requirements change by user. Less appropriate information is available for making decision with requirements.	Media recording framework.
Ravichandar et al. [119]	Problems associated with tracing back to their sources.	Pre-requirements traceability technique.

Table 13
Traceability issues and their solutions (verified).

Scholarly work	Issues in traceability	Solution (verified by industry)
Blaauboer et al. [127]	Adopting requirement traceability into projects.	Increase awareness and adapt organizations to include requirement traceability.
Cleland-Huang [128]	Failure to trace non-functional requirements e.g. security, performance and usability	Goal centric traceability evaluated by an experiment
Gotel and Finkelstein [111]	Some problematic questions are identified as challenges: Who identifies a requirement and how? Who was responsible for the requirement to start with and who is currently responsible? Who is responsible for change(s) in requirements? What will be the effect on the project in terms of knowledge loss if key employees quit?	Framework of contribution structure.
Heindl and Biffl [116]	Cost related to requirement traceability.	Value-based requirements tracing tested through a case study.
Ramesh [129]	Organizational, environmental and technical factors.	Best practice given.
Verhanneman et al. [121]	Requirement management challenges in industry projects e.g. inadequate impact analysis and lack of information transfer.	Requirement management tools like DOORS and RequisitePro.

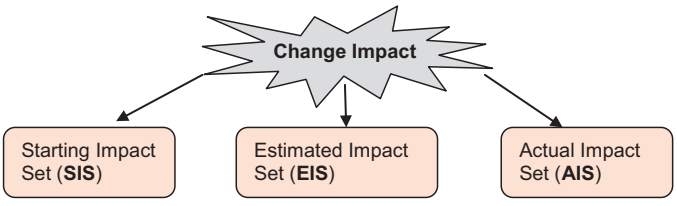

Fig. 2. Change impact object sets.

This is a concept introduced by Arnold and Bohner [130]. We identified in the literature several impact analysis techniques that use traceability and non-traceability methods. These methods were subject to the concept introduced by Arnold and Bohner [130] to identify which set of objects are analysed and are detailed in Tables 14 and 15. This finding benefits software practitioners in selecting a potential method for change analysis based on the set of objects on which they want to focus. Table 14 details solutions that use traceability techniques to analyse RC while Table 15 details solutions that use other techniques.

6.2.3. Predicting requirements changes

Another aspect of analysing change is to proceed beyond the existing change impact and to use historical data, design diagrams, codes, etc. to predict where change may occur and identify their impact. Based on this concept, we were able to extract literature that discusses the prediction of RCs, their possible impact on the systems and how the change may propagate through the system. These findings are important in order for development teams to

Table 14
Techniques used for impact analysis – traceability methods.

Scholarly work	Title of work	Solution (using traceability)	Impacted objects
Antoniol et al. [131]	Identifying the impact set of a maintenance request	The tracing is done at a coding level where the text in the maintenance request is mapped to development code components corresponding to the change request.	SIS
Li et al. [132]	Requirements-centric traceability for change impact analysis	The method uses an interdependency graph and traceability matrix to assess the impact at a requirement specification level.	SIS, EIS and AIS
Ibrahim et al. [133]	Integrating software traceability for change impact analysis	The method provides a holistic traceability solution that involves both high level and low level software models ranging from requirements to code.	AIS
Göknil et al. [134]	Change impact analysis based on formalization of trace relations for requirements	The method deals with a requirements metamodel with well-defined types of requirements relations, which are used to define change impact rules for requirements. These rules help identify the impacted requirements.	EIS and AIS
Von Knethen [135]	Change-oriented requirements traceability. Support for evolution of embedded systems	The approach consists of three parts, a conceptual trace model for embedded systems, rules to establish traces and analyse impact and a tool for semi-automatic impact analysis and consistency checking.	SIS and AIS

Table 15
Techniques used for impact analysis – non-traceability methods.

Scholarly work	Title of work	Solution (using non-traceability methods)	Impacted objects
Kobayashi and Maekawa [10]	Need-based requirements change management	The method captures RC using the 4Ws: where, who, why and what. The solution mainly consists of verification and validation activities.	SIS
Ali and Lai [136]	A method of requirements change management for global software development	The method consists of three stages: understanding change, analysing these changes and finally making decisions regarding the change based on the analysis.	SIS
Hassine et al. [137]	Change impact analysis for requirements evolution using use case maps	Method uses slicing and dependency analysis at the use case map specification level to identify the potential impact of RCs on the overall system.	SIS
Briand et al. [90]	Impact analysis and change management of UML models	The method uses a UML model-based approach where the UML diagrams are first checked for consistency. The impact analysis is carried out using a change taxonomy and model elements that are directly or indirectly impacted by the changes.	SIS and EIS
Hewitt and Rilling [138]	A light-weight proactive software change impact analysis using use case maps	The method seeks to predict impact of changes at a specification level. The method focus on extracting information from Use Case Maps (UMC) that can be used for proactive change impact analysis at the specification level.	SIS

foresee how to be prepared for RCs, make better decisions and better implement such changes. We present the prediction methods and their limitations in Table 16.

6.2.4. Change analysis using agile techniques

In agile development, requirement engineering activities are not explicit. Partially, this is due to the fact that there are less distinct boundaries in agile development than in traditional software development [145]. Therefore, similar to change identification, the analysis of RCs in agile development is not restricted to a particular phase of the development but a mixture of techniques is used that occur iteratively. The agile techniques discussed in RQ2 (Section 5.3) are detailed in Table 17 to show how change analysis is carried out in agile development.

Two of the documents used in agile development that are worth mentioning are user stories and product backlog, which form a critical part of the change analysis process. User stories are created as the specification of the customer requirements. They facilitate better communication and unambiguous understanding between all stakeholders [80]. User stories are made up of three components: a written description, conversations, and tests [148]. They are meant to reduce the need for constant requirement change and also act as a reference point to check if changes are implemented to satisfy the client requirements. Product backlog keeps track of the details of all the developed requirements. This is one of the documents that can be used to keep track of all the requirements changes [78].

6.3. Change cost/effort estimation

Software cost/effort estimation is referred to as the process of predicting the effort required to develop a software system [149,150]. It is noteworthy that although effort and cost are closely related, they are not a simple transformation of each other [149]. Effort is often measured in person-months of the development team whilst cost (dollars) can be estimated by calculating payment per unit time for the required staff and then multiplying this by the estimated effort [149]. Cost estimation is usually carried out at the beginning of a project but as we have demonstrated, changes to the system can occur at any stage of the project. Therefore, there is a need to estimate the additional cost for implementation of the change.

There are some basic factors to be considered when estimating, regardless as to whether it is for the entire project or just for a change. The first step in cost/effort calculation is the calculation of the size of the software, which is considered to be the most important factor affecting estimation [149]. Therefore, it is essential to understand the popular software sizing methods used and their suitability for estimating the cost/effort of implementing requirements changes, as shown in Table 18.

Table 16
Methods of predicting requirements changes.

Title	Solution	Limitations
1. Learning from Evolution History to Predict Future Requirement Changes [139]	Method uses historic information to develop a metrics that measures the evolution history of a requirement. Based on the metrics, the method proposes to reduce the impact of requirements evolution by attempting to predict requirements that are prone to change in the future.	Can only be applied to projects that have historic data. Some important requirements changes may be neglected by the prediction method.
2. Managing Changing Compliance Requirements by Predicting Regulatory Evolution [140]	Method uses an adaptability framework which helps requirements engineers to identify: why requirements change (rationale); how requirements change (classifications); and which portions of a proposed rule are most likely to change when the final rule is issued (heuristics). The framework allows engineers to focus primarily on analysing and specifying compliance requirements from the more stable areas of the laws, while the less stable areas of the laws are clarified during the final rulemaking.	The study uses two case studies from the healthcare industry and therefore the findings and applicability remain limited to the healthcare industry.
3. Mining the Impact of Object-Oriented Metrics for Change Prediction using Machine Learning (ML) and Search-based Techniques (SBT) [141]	This method is used to identify the probability of classes that would change (change proneness of a class) in the subsequent release of software. The study develops a relationship between Object- Oriented metrics and the change proneness of a class. The method evaluates the effectiveness of six SBT, four ML techniques and the statistical technique - Logistic Regression (LR) on change proneness prediction data and compares their results.	Findings and applicability limited to object-oriented environments.
4. Using Early Stage Project Data to Predict Change-Proneness [142]	This paper presents a feasibility study undertaken to test the validity of a hypothesis that data from requirements and design activities may also prove to be useful in predicting change proneness. A metrics is developed for quantifying requirements and design activities. Next, values are generated for these metrics from a real-world case study and finally a comparison is made with the actual number of changes detected.	Method can only be applied if the project has requirements and/or design information available. Clearly, this creates a limitation for approaches such as agile methods that have limited documentation.
5. Predicting the Probability of Change in Object-Oriented Systems [143]	This is a probabilistic approach to estimate the change proneness of an object-oriented design by evaluating the probability that each class of the system will be affected when new functionality is added or when existing functionality is modified. The goal is to assess the probability of how each class will change in a future generation.	Previous versions of a system must be analysed to acquire internal probability values creating scalability problems for large systems. Cannot be applied in the initial stages of the development process (e.g. at the design level).
6. Using Bayesian Belief Networks to Predict Change Propagation in Software Systems [144]	The approach seeks to predict the possible affected system modules, given a change in the system. The method is composed of two steps: extracting information and predicting changes. In the first step, the authors extract the system elements' dependencies and change history. In the second step, the Bayesian Belief Networks are built using the extracted information and then predictions are produced using probabilistic inference.	Can only be applied to methods that have historic data and documentation.

Table 17
Change analysis using agile methods.

Agile technique	How change analysis is done
Iterative requirements [74,78,79]	The requirements related to a user story are not identified at the beginning of a project. Requirements are built on iterations which allow stakeholders to gain a better understanding of what is required and therefore analyse and understand the need for changes.
Requirement prioritisation [75,78–80]	In each of the iteration, the identified requirements are prioritised. This means that any changes that occur during the iterations will be compared to existing requirements and will be assigned a place in the hierarchy of implementation. The iterative nature of this activity ensures the priority of requirements remain current.
Prototyping [74,78,81]	This allows the agile team to review the requirement specifications with clients to obtain feedback. The process will highlight issues with the changes identified so far and will prompt the development team to find better solutions.
Testing before coding [74,78,79,146]	The development team writes tests prior to writing functional codes for requirements. This promotes identification test failure which can be a form of validation of the changes that have been applied during the iterations.
Requirement modelling [82,83]	A technique used in modelling in agile approaches is goal-sketching [83]. The outcome is an easy-to-read goal graph which allows all stakeholders to refine the goals, making them well defined. Changes that are introduced in the iterations can be mapped to goals and this can help with decision making in the implementation of changes.
Review meetings and acceptance tests [78,84]	The developed requirements and product backlogs are reviewed to identify if user stories have been completed. In terms of change analysis, this evaluates if changes have been implemented correctly and satisfy the end goal.
Regression testing [147]	Regression testing is done in agile methods to make sure that the newly incorporated changes do not have side effects on the existing functionalities and thereby finds the other related bugs. This is a form of change validation in terms of change analysis.

There are many methods described in the literature that are popular techniques for estimating cost/effort. As presented in Fig. 3, we considered the more frequently used estimation methods in traditional software development and they can be classified into two categories: algorithmic and non-algorithmic [149,157]. Algorithmic models can be quite diverse in the mathematical expressions used. It is important to remember that these algorithmic models need to be adjusted to suit the local environment. Regardless of the technique used, none of the methods discussed in this section can be used off-the-shelf.

One of the key findings in this section is to identify the appropriateness of these methods for estimating the cost/effort of implementing RCs. Tables 19 and 20 describe several popular estimation techniques that belong to these two categories and their suitability for change cost estimation.

Table 18
Popular software sizing techniques.

Sizing technique	Feature	Suitability for change cost/effort calculation
Line of Code (LOC) [149,151]	Based on the number of lines of the delivered source code of software. Programming language dependent. Widely used sizing method.	Exact LOC can only be obtained after the completion of the project and is therefore not suitable for changes at the early stage of the design. Also depends on expert judgement and can compromise reliability. Can be used for changes that occur towards the latter part of the development process.
Software science [152]	Based on code length and volume metrics. Code length is the measurement of the source code program length and volume is the amount of storage space required.	There have been disagreements over the underlying theory and therefore reliability is questionable [153,154]. Not suitable for changes in the early phase (reason as above). Possibility of using this in the latter stages, yet the measure has received decreasing support [149].
Function points [155]	Working from the specification, systems functions are counted (inputs, outputs, files, inquiries, interfaces) These points are then multiplied by their degree of complexity.	Use of the specification makes it suitable to analyse changes in the early phase of development. Equally suitable for changes in the latter stages.
Feature point [156]	Extension of function points to include algorithms as a new class.	Similar usability as function points and suitable systems with little input/output and high algorithmic complexity.

Table 19
Popular estimating techniques – non-algorithmic.

Category	Non-algorithmic		
Technique	Features	Challenges	Suitability for change cost/effort estimation
Expert judgment	Based on one or more experts using their experience and techniques such as PERT or Delphi for estimation.	Dependency on experts, where human error is a major risk and there can be bias.	Can be suitable since the method is fast and can easily adapt to diverse circumstances. But the limitation carries a lot of risk.
Parkinson	Cost is determined (not estimated) by the available resources rather than an assessment of the entire situation.	Can provide unrealistic estimations and does not promote good software engineering practice.	Given the limitations far exceed its functionality, it cannot be recommended.
Price to win	Estimated to be the best price to win a project. Estimate is based on customer budget.	Not good software practice as software functionality is not considered. Can produce large overruns.	Software functionality is a key factor in change cost estimation and therefore is not suitable.
Bottom-up	Each component of the system is estimated separately and the result is combined to produce the overall estimate. Based on initial design.	Requires more effort and can be time consuming.	Can be suitable for changes in the latter phase. Not suitable for changes in the early phases as it requires detailed system information.
Top-down	The opposite of the bottom-up approach. This is an overall estimation based on global properties. Total cost can be split among the various components.	Less stable as the estimation does not consider different components.	Useful for changes in the early stages. Changes in the latter phases require more detailed costing and therefore it is not suitable.

Table 20
Popular estimating techniques – algorithmic.

Category	Algorithmic		
Technique	Features	Challenges	Suitability for change cost/effort estimation
COCOMO	Uses power function models where $\boldsymbol{Effort} = \boldsymbol{a} \times \boldsymbol{S^b}$ S is the code size and a, b are functions of other cost factors.	Not suitable for small systems.	Exact code size can only be obtained at the completion of a project and therefore may not be suitable for changes at early stages.
Putnam's model and SLIM	Equation used $\boldsymbol{S} = \boldsymbol{E} \times (\boldsymbol{Effort})^{1/3}\boldsymbol{t_d}^{4/3}$ where S is LOC, t_d is delivery time, E is environment factor (based on historical data)	Based on information from past projects and may not be suitable for the current environment.	Although generally suitable for changes in cost estimation, dependency on historical data can make the accuracy questionable.
Price-S	This is a proprietary estimation model. Uses an estimate of project size, type and difficulty and computes cost and schedule.	Because it is company specific, it may not suitable for all environments.	Not suitable for change cost estimations due to limitations.

Effort estimation is more challenging in the agile context as requirement changes are embraced through multiple iterations of development. In line with the previous two sections, we consider the techniques used in agile development for effort estimation. Table 21 details the techniques, the challenges and the suitability for change cost/effort estimation.

Key findings of RQ_3

The majority of the academic work on RC is focused on devising solutions for the different areas of RCM. Based on the discussion that formulated the answer for RQ_3, the following are the key findings:

1) Change identification methods do not seem to have much consensus on how the identification should be done nor are many of the methods formal.
2) Most change identification methods found are based on two techniques: through taxonomies and through classifications.
3) The change taxonomies tend to be based on larger concepts such as use cases and UML models whilst change classifications use more simplified mechanisms such as change directions and parameters.
4) Change identification usually leads to understanding of the need for the change, which also relates to further analysis of the change.
5) Traceability techniques have been the more popular choice when analysing change as requirement traceability facilitates the identification of the impact of change more efficiently. However, this seems to be a theoretical concept as requirement traceability has many limitations.

Table 21
Popular estimating techniques – agile.

Category	Agile		
Technique	Features	Challenges	Suitability for change cost/effort estimation
Expert judgment [158,159]	Developers look to past projects or iterations, and draw on their own experiences to produce estimates for the user stories.	Dependency on experts, where human error is a major risk and there can be bias.	Can be suitable since the method is fast and can easily adapt to diverse circumstances. But the limitation carries a lot of risk.
Planning poker [160,161]	Once the user stories have been understood, all the team members of the agile team make independent estimates and reveal their estimates simultaneously. The lowest and highest estimates need to be justified by their estimator. The group continues the discussion in order to decide on a collective estimate, possibly by conducting one or more additional rounds of individual estimating.	If the estimation process is unstructured, factors such as company politics, group pressure, anchoring, and dominant personalities, may reduce estimation performance.	Similar suitability as expert judgment but is still dependent on the skill and experience of the team members.
Use Case Points (UCP) [162,163]	Once the use cases are identified based on the user stories, UCPs are calculated based on the number and complexity of use cases and actors of the system, non-functional requirements and characteristics of the development environment. The UCP for a project can then be used to calculate the estimated effort for a project.	UCP method can be used only when the design is done using UML or RUP.	Can be suitable for an early stage change estimation of the development process. Changes in the latter phases require more detailed costing and therefore it is not suitable.
Story points [164–166]	Story point is a measure for relatively expressing the overall size of a user story or a feature. A point is assigned to each user story. The value of the story point is dependent on development complexity, the effort involved, the inherent risk and so on.	Story points create lots of vagueness to the agile process. For every team, story size could mean different things, depending on what baseline they chose. If two teams are given the same stories, one team can say their velocity is 46 and the other can say 14, depending on what numbers they chose. Story points do not relate to hours.	May only be suitable for teams that are collocated, based on the challenges of the method. Also, it may not be suitable for effort calculation in hours as it will take additional calculations to convert story points to hours.

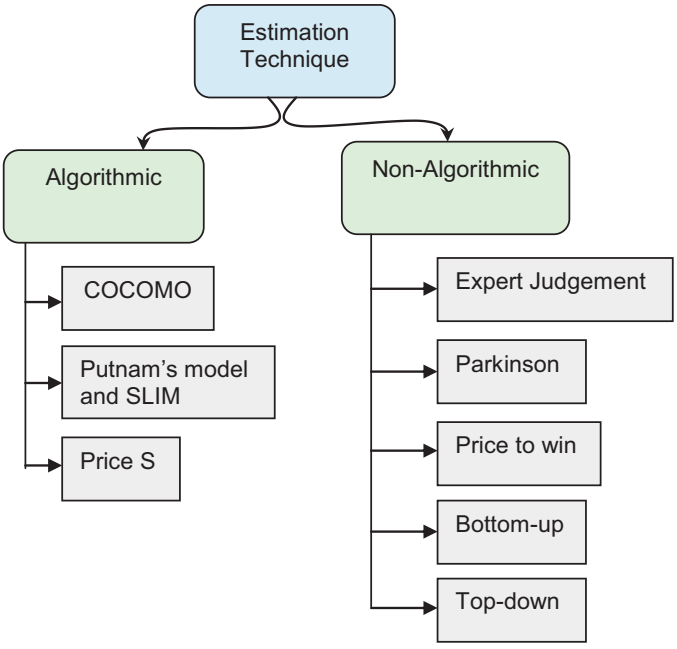

Fig. 3. Costing techniques.

6) The main idea of change analysis is to identify how the requested change impacts the existing design or system. To this effect, methods of change impact analysis found in literature can be grouped based on objects that are impacted: starting impact set, estimated impact set and actual impact set.
7) In terms of the agile context, changes in requirements are expected and welcome aspects of development. As we discovered in the literature, change identification and analysis tend to happen at almost all parts of the iterative process in development.
8) Due to the change-susceptive nature of agile development, unlike traditional development, in most cases change identification and analysis does not require special processes but are embedded into the processes that are part of the development cycle.
9) Costing techniques dedicated for estimating the cost of RC seem to be rare. In most cases, existing costing techniques such as COCOMO, expert judgement, etc. are used for this purpose.
10) It is possible to divide existing costing techniques into two categories: algorithmic and non-algorithmic.
11) Depending on which point of the lifecycle the software project is and what artefacts are used for the cost estimation, each estimation can be judged for suitability to be used for cost estimation of RCs.
12) Some methods can be used but with many risks (i.e. expert judgement), some methods can be used for changes introduced in the latter phase of the project life cycle (i.e. bottom-up, COCOMO, etc.), some methods can be used for changes introduced in the early phase of the project life cycle (i.e. top-down) and some other methods are not suitable for change cost estimation (i.e. price to win, Price-S, etc.).
13) Unlike change identification and analysis, cost/effort estimation in agile development requires special attention. The nature of agile development tends to discover requirements through several iterations and therefore, any estimations at the beginning of a project change significantly along the development cycle. Given this criterion, special techniques are

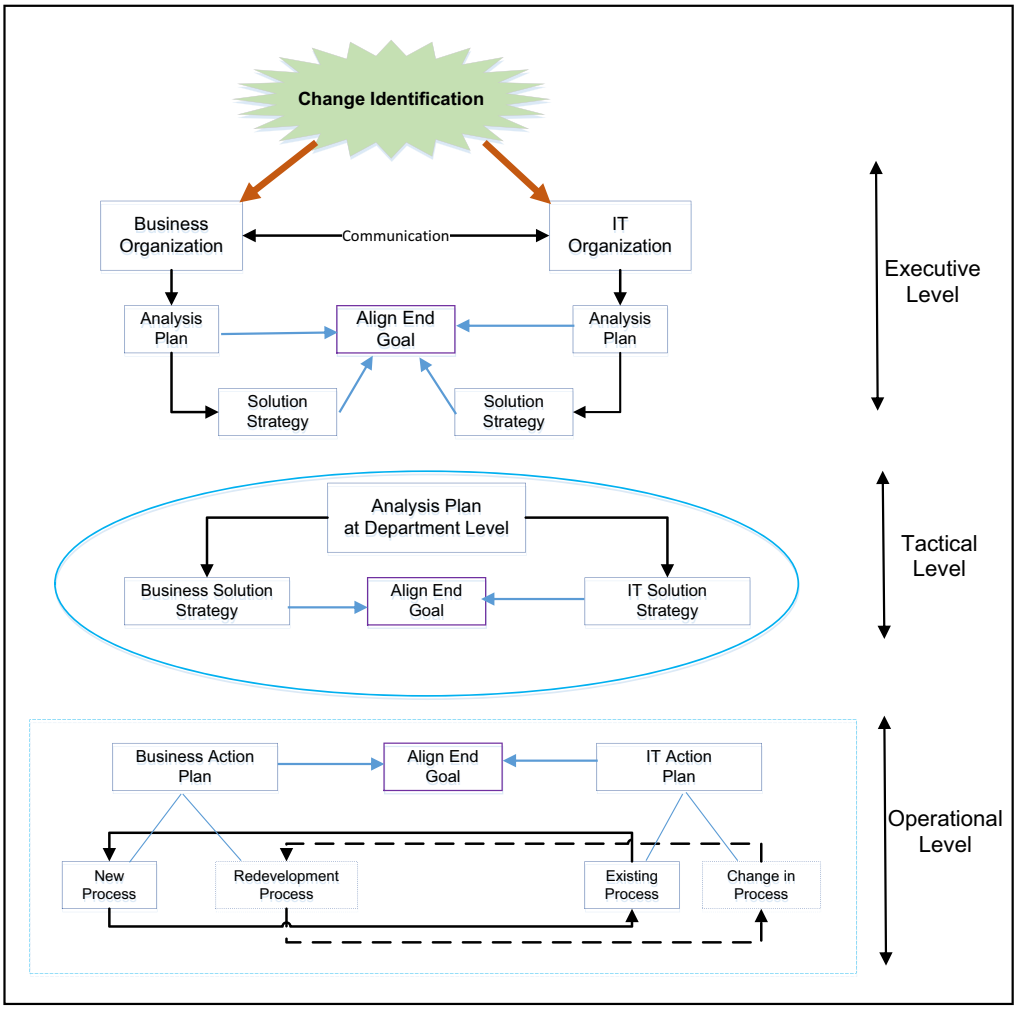

Fig. 4. RCM with respect to organization level.

required for the estimation of cost and effort, which, we discovered in the literature, are mostly dependent on expert judgement and team collaboration.

7. Results for RQ_4: how do organizations make decisions regarding requirements changes?

An organization has a harmonious existence when coordination and integration between business objectives and IT services and infrastructure in realizing the common business goals are in alignment [167–169]. However, when managing RCs of system software or software projects, stakeholders may perceive different end goals at different levels of the organization [170]. In other words, change management and analysis plans and strategies vary with organisational level, where each strategy tends to have different goals and objectives. An organization can be categorized into two parts: business organization and IT organization and each of these two categories can be split into three levels, as illustrated in Fig. 4. We used the search expressions A_3 OR A_6 OR A_8 (see Table 1) to extract the relevant literature.

7.1. Executive level

Once the need for a change in a software process or requirement arises, the top level management (CEO, CIO, etc.), which is the executive level, formulates very broad strategies for managing the said change. The tendency to create broad plans is usually due to the responsibilities of the top level executives in terms of what the organization as a whole stands to gain by implementing these changes [170]. In some instances, business and IT tend to have a contradictory understanding of the need for change. Decisions by the IT side for obtaining new technology that is required for implementation of the change may not always be agreed upon by the business counterparts of an executive level [59,170]. Research has demonstrated that when business and IT top management fail to understand the need for the change and the IT capabilities that are required for its realization, these software projects tend to have unsatisfactory outcomes in the form of cost overruns and failure [18,168,170,171].

7.2. Tactical level

The tactical level in Fig. 2 corresponds to the change management plans and strategies formulated by the middle management of an organization. These strategies can be referred to as functional strategies. The main concern at this level is to assess the change with respect to cost and benefits and find ways to introduce the change without adversely affecting the project [2,20,59,170]. The broad strategies at an executive level may not always match with the strategies formulated at a tactical level. For example, the end goal of a change at an executive level could be to improve quality while at a tactical level, the goal would be to complete the project successfully and therefore, may consider the change intrusive [59,170]. It is also noteworthy that the notion of business

vs. IT mindset exists at this level too. One of the key barriers in creating a cohesive change strategy between business and IT at this level is due to interpretation and communication barriers that stem from the lack of a common change specification technique [38,99,172,173].

7.3. Operational level

As the strategies flow down the organizational structure, they tend to become less complicated and less abstract. At this stage, it becomes a process of understanding the strategies laid down by the tactical level and formulate plans as to how to best implement them. The goals at this level are more short-term due to the fact that development teams are dealing with simpler strategies. Provided that business and IT change strategies at this level are aligned, the combination of such short term strategies could be linked back to the business objectives set at the executive level [174]. Moreover, it is essential at this level that development teams are able to cope with the changes in the business strategies originating at a higher level. Therefore, strategies formulated at an operational level should incorporate a mechanism to deal with such changes that will ensure the final product is what is expected by the executive level.

7.4. Different viewpoints based on structure

Change analysis can be observed from two main viewpoints: one from a developer point of view at a code level and the second from a decision-maker's point of view at a higher abstraction level. The executive and the tactical levels can be considered as the decision-maker point of view while the operational level represents the developer point of view. There has been debate over which of these levels is more important in change management. Some of the literature emphasizes the importance of managing change at a program modification level where such analysis would be helpful to a programmer to effectively implement the change [175–177]. In support of a higher level of decision making to effectively manage change, many studies argue that it is inaccurate to realize change at the code level, where in fact the source of the change is at a requirement level and therefore should be managed at a higher abstraction level [132,136,137].

7.5. Decision making and organizational culture in agile development

The primary goal of all agile methods is to deliver software products quickly, and to adapt to changes in the process, product, environment, or other project contingencies [178]. While evidence suggests that agile methods have been adopted in a wide variety of organizational settings [179–181], such methods are assumed to be more suited to certain organizational environments than others. According to Nerur et al. [179], Karesma [180], Reifer et al. [181] and Pino et al. [182], agile development is more suited to smaller organizations as development is carried out in small teams. There are scalability issues when it comes to large organizations or large projects [180,181]. In smaller organizations, there is a strong positive correlation in some aspects of organizational culture with that of agile development; the organization values feedback and learning; social interaction in the organization is trustful, collaborative, and competent; the project manager acts as a facilitator; the management style is that of leadership and collaboration; the organization values teamwork, is flexible and participative and encourages social interaction; the organization enables the empowerment of people; the organization is results-oriented; leadership in the organization is entrepreneurial, innovative, and risk taking; and the organization is based on loyalty and mutual trust and commitment [183].

There are certain characteristics of agile development, such as cross-functional teams and customer involvement that create harmonious interaction between various levels of the organization in decision making. Cross-functional teams include members from different functional groups who have similar goals [75,184]. Such a practice combined with customer involvement helps reduce challenges such as over scoping of requirements and communication gaps, which are some of the key causes of requirement change. According to these studies, agile development has the ability to create harmony within the organizational culture and within the structure of the organization that will positively contribute to the reduction of the number of changes required and will be able to gain better clarity in decision making and the development of software projects.

Key findings of RQ_4

Not many studies in the literature used for this survey discuss how decision making at various levels of the organization may differ. We feel that this is an important concept to investigate as such differences in decisions can create difficulties in coming to a consensus on accepting the change and also moving forward by executing the change. Based on the discussion that formulated the answer for RQ_4, the key findings are as follows:

1) It is important to realize that based on the level of the organizational structure, decision-making concepts differ and this can be detrimental to the success of a project when dealing with RCs.
2) An organization can be divided into two parts i.e. the business organization and the IT organization.
3) Each of these two parts can then be divided into three levels of structure: Executive, Tactical and Operational. The differing levels of decision making between these structural levels have been identified to be a challenging factor in RCM.
4) Not only can decision making be contradictory at each level, it can also cause a contradictory understanding of the change between the business and IT counterparts.
5) There are also two viewpoints to consider: the developer and the decision maker. The literature seems to be divided on which viewpoint is more important, providing cause and effect for merit for both viewpoints.
6) Agile techniques tend to be a better way of development when it comes to creating better harmony within the organizational culture and decision making. However, this comes with the constraints of scalability and therefore is better recommended for development using smaller teams or for smaller organizations.

8. Comparison with related work

There is a plethora of work which has been evaluated in various areas of RCM, such as change impact analysis, change complexity analysis, change decision support, change identification, etc. A number of literature reviews related to change management have been conducted on research topics such as identifying change causes [35], change taxonomies [31] and requirement change process models [16]. These reviews deal with only one aspect of RCM, as detailed in Table 22.

In comparison, the work presented in our systematic review investigates the causes of requirement change and the processes/models used for RCM, it explores in-depth the techniques used in RCM and the decision making in managing change and provides a critical analysis of the methods extracted by identifying research gaps. The methods extracted comprise both traditional and agile techniques in RCM. In summary, this review provides information related to many aspects of RCM in more detail, giving a more holistic view for its readers.

Table 22
Comparison with related work.

Research work	Findings and contributions
Towards an understanding of the causes and effects of software requirements change: two case studies [31]	The study identifies various causes of requirement change and uses a simple taxonomy to group these causes for better understanding and future identification.
Causes of requirement change-a systematic literature review [35]	Similar to the previous study, identifies the causes of requirement change and groups these cause into two categories; essential and accidental. The main difference from [31] is that the study is done as a systematic review.
Requirement change management process models: Activities, artefacts and roles [16]	The study brings together various requirement management models, identifying their key features.

9. Threats to validity

The findings presented in this review study have the following threats to validity.

(i) Construct validity: this is primarily related to obtaining the right information by defining the right scope. At this stage, the biggest challenge is to decide what should be included in the review. To address this issue, we considered all the studies which provided empirical, case study, experimental, industrial and survey-related information about RCM.
(ii) External validity: the findings of this review cannot be generalized because the results are based on a specific set of keywords and the research repositories that have been used for the data collection. Therefore, our results could be limited and cannot be applied to every organizational setup.
(iii) Results validity: the concept of RCM has a very long history dating back to the early 1980s. The area is still evolving and a large set of keywords are available which can be used to represent the concept of RCM. In this review, we considered 12 different keywords which are mostly used in the context of RCM in software development, and used six research repositories to conduct an initial search in the study selection process. Thus, our findings are only based on the selected set of keywords and from six research repositories.
(iv) Internal validity: this is mainly related to the capability of replicating similar findings. We addressed this aspect by defining and later following the systematic review procedure, described in Section 3. Two researchers were involved in the review process, who, over a period of time, worked together to avoid duplications and achieved consensus in the acceptance of the identified studies. However, it could be possible that if this study is replicated by other researchers, minor variations in the identified studies will be observed due to differences in personal aptitude and thinking. Regardless of this fact, the findings presented in this review will enable readers to obtain a clear picture of RCM.
(v) Conclusion validity: The number of research articles presented in this study does not indicate the actual number of RCM practices being undertaken in reality. Thus, the number could only be used to make inferences as to how practical and applicable RCM methods are.

10. Conclusions and future work

It is evident that changes in requirements occur for many reasons and can be caused by multiple stakeholders. Regardless of who or what cause these changes, the need for appropriate management is great due to the undesirable consequences if left unattended. However, through this review, it was discovered that change management is an elusive target to achieve and that there are many ways to tackle it. The main objective of this review was to collate information and techniques related to RCM and critically analyse the functionality of such techniques in managing change. This also led to identifying strengths and limitations of these techniques, which signifies the need to enhance the existing change management approaches. This review is also a guide for future researchers on change management in terms of what major work has been undertaken thus far.

In the review, the section on factors that cause change in requirements provides an understanding on how vast and constant these changes can be. There is no one root cause for changes which makes change management a challenging task. Therefore, even with an abundance of research on change management, there is still room for improvement. Given the complexity of changes, it is important to identify the processes in place to manage them. It is clear from the available literature that there is no consensus on how to manage change. In some instances, it is based on the type of organization and the environment and in many cases, it is based on the type of changes. Through the available process steps, three common processes were identified; identification, analysis and cost estimation of change. Significant work has been done in each of these areas and several models that encompass these steps have been developed in an effort to provide a full-scale solution for change management. It is also important to understand that the approaches vary depending on the level of the organisation managing the change.

When identifying future work in RCM, we deemed it useful to focus on the three areas of RQ_3 where the majority of the techniques have been discussed. We do not directly suggest future work but identify the research gaps in the areas of change identification, analysis and cost estimation where the possibility for new research lies.

10.1. Research gaps in change identification

Accurate change identification not only leads to a better understanding of the required change but also the impact it can cause on the entire system and project. The techniques discussed in change identification can be divided into two categories: change taxonomies and change classification as discussed in the previous section. Given the existence of these methods, their still remains several major gaps that need to be addressed:

1) The parties involved in the elicitation and identification process of changes are from a variety of backgrounds and experience levels. Common knowledge for one group may be completely foreign for another. This is especially true in the case of communication between the analyst and the stakeholder(s).
2) The language and terminology used to communicate the changes to and from the stakeholder to the analyst and then to software practitioners (designers, developers, testers, etc.) may be either too formal or informal to meet the needs of each party involved.
3) There will be a large amount of information gathered that is part of one single change. Not having a common structure to categorize this information may lead to misinterpretation of the need for the change and the change itself.

4) Information gathered at one level of the organization could be biased based on the parties involved if one form of structure is not used to capture the changes at all levels.
5) The methods already in existence provide minimal guidance in terms of applying them to identify changes.

10.2. Research gaps in change analysis

As seen in the previous section on change analysis, it is clear that traceability is one of the most popular techniques to analyse the impact of changes on a system, either in existence or in the design phase. Several other non-conventional methods were also identified that contribute to change analysis. Through these methods and the existing knowledge on the volatility of requirements, several gaps in the research are identified:

1) Although traceability is a common method of identifying impact, it can be costly and time consuming, and in most cases, the benefits (of traceability) are realized immediately. This gives rise to a need for another method that addresses these limitations.
2) In most existing methods of change impact analysis, the priority of changes is not established. Understanding priority benefits the decision-making process by allowing software practitioners to establish which change to implement first and also how critical the change is to the existing system and hence, resources can be allocated accordingly.
3) The existing literature is unclear on ways to identify the difficulty of implementing a change in an early phase of the change request process. Understanding the difficulty associated with a change leads to better decision making in two ways: firstly, if the difficulty of implementing the change is too high and the delivery of the product is time sensitive, the change could be held back for a consecutive version; secondly, the difficulty can be used as a gauge of the effort required to implement the change.

10.3. Research gaps in change cost estimation

The cost estimation methods discussed in the previous section were not explicit for the estimation of implementing changes. In practice, these methods can still be applied for this purpose yet there is still much room for improvement. Based on the information discussed earlier and in the other related literature, several gaps in the research were identified:

1) No significant work in the existing literature caters explicitly for estimating the cost of implementing RCs. As demonstrated in the previous sections, changes occur for a plethora of reasons and can occur during any phase of the software development life cycle. Therefore, it would be beneficial if there was a dedicated method by which to estimate the cost of such changes as the implication of these changes based on the project's timeline results in different outcomes.
2) Estimation done at an early stage of the development process is usually based on expert judgement with less precise input and less detailed design specification. In some cases, this may result in effort estimation which is too low which leads to issues such as delayed delivery, budget overrun and poor quality while high estimates may lead to loss of business opportunities and the inefficient use of resources.
3) Estimating the cost in the early stages of development depends on expert judgment and historical data which can be biased and inconsistent. There needs to be ways to eliminate these ambiguities in change cost estimation.

The research gaps identified indicate the importance of having a full- scale model that increases the efficiency of managing change with better accuracy. The review highlights that although the concept of change management has been in existence for many years, the applicability of the available methods has many limitations and has room for improvement. With challenges such as poor communication, impact identification issues and no dedicated method for change cost calculation, the avenues for future research is promising.

References

[1] N. Nurmuliani, D. Zowghi, S. Powell, Analysis of requirements volatility during software development life cycle, in: Software Engineering Conference, 2004. Proceedings. 2004 Australian, IEEE, 2004, pp. 28–37.
[2] I. Sommerville, Software Engineering. International Computer Science Series, Addison Wesley, 2004.
[3] N. Nurmuliani, D. Zowghi, S.P. Williams, Requirements volatility and its impact on change effort: evidence-based research in software development projects, in: Proceedings of the Eleventh Australian Workshop on Requirements Engineering, 2006.
[4] S. McGee, D. Greer, A software requirements change source taxonomy, in: Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on, IEEE, 2009, pp. 51–58.
[5] S. Ramzan and N. Ikram, "Making decision in requirement change management," in 2005 International Conference on Information and Communication Technologies, 2005, pp. 309–312: IEEE.
[6] W. Lam, V. Shankararaman, Requirements change: a dissection of management issues, in: EUROMICRO Conference, 1999. Proceedings. 25th, vol. 2, IEEE, 1999, pp. 244–251.
[7] M. Strens, R. Sugden, Change analysis: a step towards meeting the challenge of changing requirements, in: Engineering of Computer-Based Systems, 1996. Proceedings., IEEE Symposium and Workshop on, IEEE, 1996, pp. 278–283.
[8] J. Tomyim, A. Pohthong, Requirements change management based on object-oriented software engineering with unified modeling language, in: Software Engineering and Service Science (ICSESS), 2016 7th IEEE International Conference on, IEEE, 2016, pp. 7–10.
[9] L. Lavazza, G. Valetto, Enhancing requirements and change management through process modelling and measurement, in: Requirements Engineering, 2000. Proceedings. 4th International Conference on, IEEE, 2000, pp. 106–115.
[10] A. Kobayashi, M. Maekawa, Need-based requirements change management, in: Engineering of Computer Based Systems, 2001. ECBS 2001. Proceedings. Eighth Annual IEEE International Conference and Workshop on the, IEEE, 2001, pp. 171–178.
[11] S. Ghosh, S. Ramaswamy, R.P. Jetley, Towards requirements change decision support, in: 2013 20th Asia-Pacific Software Engineering Conference (APSEC), vol. 1, IEEE, 2013, pp. 148–155.
[12] B. Nuseibeh, S. Easterbrook, Requirements engineering: a roadmap, in: Proceedings of the Conference on the Future of Software Engineering, ACM, 2000, pp. 35–46.
[13] N. Ikram, The Management of Risk in Information Systems Development, 2000.
[14] B.R. Butler, et al., The Challenges of Complex IT Projects, Royal Academy of Engineering. e British Computer Society, 2004 *Relatório Técnico*.
[15] B. Curtis, H. Krasner, N. Iscoe, A field study of the software design process for large systems, Commun. ACM 31 (11) (1988) 1268–1287.
[16] S. Ramzan, N. Ikram, Requirement change management process models: activities, artifacts and roles, in: 2006 IEEE International Multitopic Conference, IEEE, 2006, pp. 219–223.
[17] B.W. Boehm, Understanding and controlling software costs, J. Parametrics 8 (1) (1988) 32–68.
[18] D. Firesmith, Common requirements problems, their negative consequences, and the industry best practices to help solve them, J. Object Technol. 6 (1) (2007) 17–33.
[19] S. Lock, G. Kotonya, An Integrated Framework for Requirement Change Impact Analysis, 1999.
[20] I. Sommerville, P. Sawyer, Requirements Engineering: A Good Practice Guide, John Wiley & Sons, Inc., 1997.
[21] A. Taylor, IT projects: sink or swim, Comput. Bull. 42 (1) (2000) 24–26.
[22] E. Oz, When professional standards are lax: the CONFIRM failure and its lessons, Commun. ACM 37 (10) (1994) 29–43.
[23] S. Lock and G. Kotonya, "Requirement level change management and impact analysis," 1998.
[24] B. Kitchenham, Procedures for Performing Systematic Reviews, in: Procedures for Performing Systematic Reviews, vol. 33 (2004), Keele University, Keele, UK, 2004, pp. 1–26.
[25] B. Kitchenham, O.P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering–a systematic literature review, Inf. Softw. Technol. 51 (1) (2009) 7–15.
[26] A. Ullah, R. Lai, A systematic review of business and information technology alignment, ACM Trans. Manage. Inf. Syst. 4 (1) (2013) 4.
[27] D. Wickramaarachchi, R. Lai, Effort estimation in global software development - a systematic review, Comput. Sci. Inf. Syst. 14 (2) (2017) 393–421.
[28] M.B. Miles, A.M. Huberman, Qualitative Data Analysis: An expanded Sourcebook, SAGE, 1994.

[29] B.J. Williams, J. Carver, R.B. Vaughn, Change risk assessment: understanding risks involved in changing software requirements, in: Software Engineering Research and Practice, Citeseer, 2006, pp. 966–971.
[30] S. McGee, D. Greer, Software requirements change taxonomy: evaluation by case study, in: Requirements Engineering Conference (RE), 2011 19th IEEE International, IEEE, 2011, pp. 25–34.
[31] S. McGee, D. Greer, Towards an understanding of the causes and effects of software requirements change: two case studies, Requirements Eng. 17 (2) (2012) 133–155.
[32] B. Boehm, Industrial software metrics top 10 list, IEEE Softw. 4 (5) (1987).
[33] S.L. Pfleeger, Software metrics: progress after 25 years? IEEE Softw. 25 (6) (2008) 32.
[34] D.M. Weiss, V.R. Basili, Evaluating software development by analysis of changes: some data from the software engineering laboratory, IEEE Trans. Softw. Eng. (2) (1985) 157–168.
[35] M. Bano, S. Imtiaz, N. Ikram, M. Niazi, M. Usman, Causes of requirement change-a systematic literature review, in: Evaluation & Assessment in Software Engineering (EASE 2012), 16th International Conference on, IET, 2012, pp. 22–31.
[36] A. Van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, Wiley Publishing, 2009.
[37] K. Wiegers, J. Beatty, Software Requirements, Pearson Education, 2013.
[38] S.D. Harker, K.D. Eason, J.E. Dobson, The change and evolution of requirements as a challenge to the practice of software engineering, in: Requirements Engineering, 1993., Proceedings of IEEE International Symposium on, IEEE, 1993, pp. 266–272.
[39] R.S. Pressman, Software Engineering: A Practitioner's Approach, Palgrave Macmillan, 2005.
[40] C. Rolland, C. Salinesi, A. Etien, Eliciting gaps in requirements change, Requirements Eng. 9 (1) (2004) 1–15.
[41] E. Fricke, B. Gebhard, H. Negele, E. Igenbergs, Coping with changes: causes, findings, and strategies, Syst. Eng. 3 (4) (2000) 169–179.
[42] A.M. Davis, K.V. Nori, Requirements, Plato's cave, and perceptions of reality, in: Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International, vol. 2, IEEE, 2007, pp. 487–492.
[43] B. Boehm, Requirements that handle IKIWISI, COTS, and rapid change, Computer 33 (7) (2000) 99–102.
[44] M.G. Christel and K.C. Kang, "Issues in Requirements Elicitation," DTIC Document1992.
[45] C. Ebert, J. De Man, Requirements uncertainty: influencing factors and concrete improvements, in: Proceedings of the 27th International Conference on Software Engineering, ACM, 2005, pp. 553–560.
[46] T. Moynihan, How experienced project managers assess risk, IEEE Softw. 14 (3) (1997) 35–41.
[47] T. Moynihan, Requirements-uncertainty': should it be a latent, aggregate or profile construct? in: Software Engineering Conference, 2000. Proceedings. 2000 Australian, IEEE, 2000, pp. 181–188.
[48] S. Ferreira, J. Collofello, D. Shunk, G. Mackulak, P. Wolfe, Utilization of process modeling and simulation in understanding the effects of requirements volatility in software development, International Workshop on Software Process Simulation and Modeling, 2003.
[49] L. Mathiassen, T. Saarinen, T. Tuunanen, M. Rossi, in: Managing Requirements Engineering Risks: An Analysis and Synthesis of the Literature, Helsinki School of Economics, 2004, p. 63.
[50] C. Jones, Strategies for managing requirements creep, Computer 29 (6) (1996) 92–94.
[51] N. Nurmuliani, D. Zowghi, S.P. Williams, Using card sorting technique to classify requirements change, in: Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International, IEEE, 2004, pp. 240–248.
[52] S. Jayatilleke, R. Lai, A method of specifying and classifying requirements change, in: Software Engineering Conference (ASWEC), 2013 22nd Australian, 2013, pp. 175–180.
[53] J. Kramer, J. Magee, The evolving philosophers problem: dynamic change management, IEEE Trans. Softw. Eng. 16 (11) (1990) 1293–1306.
[54] F.P. Brooks Jr., No silver bullet essence and accidents of software engineering, Computer (4) (1987) 10–19.
[55] J.S. O'Neal, Analyzing the Impact of Changing Software Requirements: A Traceability-Based Methodology, Clemson University, 2003.
[56] S. Lock, G. Kotonya, Tool support for requirement level change management and impact analysis, in: Doctoral Symposium Proceedings, Citeseer, 1998.
[57] K. El Emam, D. Holtje, N.H. Madhavji, Causal analysis of the requirements change process for a large system, in: Software Maintenance, 1997. Proceedings., International Conference on, IEEE, 1997, pp. 214–221.
[58] D. Leffingwell, D. Widrig, Managing Software Requirements: A Unified Approach, Addison-Wesley Professional, 2000.
[59] G. Kotonya, I. Sommerville, Requirements Engineering: Processes and Techniques, Wiley Publishing, 1998.
[60] D. Pandey, U. Suman, A.K. Ramani, An effective requirement engineering process model for software development and requirements management, in: 2010 International Conference on Advances in Recent Technologies in Communication and Computing, 2010, pp. 287–291.
[61] W. Hussain, D. Zowghi, T. Clear, S. MacDonell, K. Blincoe, Managing requirements change the informal way: when saying 'No'is not an option, in: Requirements Engineering Conference (RE), 2016 IEEE 24th International, IEEE, 2016, pp. 126–135.
[62] D.M. Berry, K. Czarnecki, M. Antkiewicz, M. AbdElRazik, Requirements determination is unstoppable: an experience report, in: Requirements Engineering Conference (RE), 2010 18th IEEE International, IEEE, 2010, pp. 311–316.
[63] M. Bommer, R. DeLaPorte, J. Higgins, Skunkworks approach to project management, J. Manage. Eng. 18 (1) (2002) 21–28.
[64] K. Skytte, Engineering a small system, IEEE Spectr. 31 (3) (1994) 63–65.
[65] B. Curtis, M.I. Kellner, J. Over, Process modeling, Commun. ACM 35 (9) (1992) 75–90.
[66] S.T. Acuña, X. Ferré, Software process modelling, in: ISAS-SCI (1), 2001, pp. 237–242.
[67] J. Lonchamp, A structured conceptual and terminological framework for software process engineering, in: Software Process, 1993. Continuous Software Process Improvement, Second International Conference on the, IEEE, 1993, pp. 41–53.
[68] N.C. Olsen, The software rush hour (software engineering), IEEE Softw. 10 (5) (1993) 29–37.
[69] M. Makarainen, in: Software Change Management Processes in the Development of Embedded Software, vol. 4 (1), VTT Publications, 2000, p. 6.
[70] W. Lam, V. Shankararaman, S. Jones, J. Hewitt, C. Britton, Change analysis and management: a process model and its application within a commercial setting, in: Application-Specific Software Engineering Technology, 1998. ASSET-98. Proceedings. 1998 IEEE Workshop on, IEEE, 1998, pp. 34–39.
[71] S.A. Ajila, Change management: modeling software product lines evolution, in: Proc. of the 6th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, 2002, pp. 492–497.
[72] S.A. Bohner, Impact analysis in the software change process: a year 2000 perspective, in: ICSM, vol. 96, 1996, pp. 42–51.
[73] A. Eberlein, J. Leite, Agile requirements definition: a view from requirements engineering, in: Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE'02), 2002, pp. 4–8.
[74] L. Cao, B. Ramesh, Agile requirements engineering practices: an empirical study, IEEE Softw. 25 (1) (2008).
[75] I. Inayat, S.S. Salim, S. Marczak, M. Daneva, S. Shamshirband, A systematic literature review on agile requirements engineering practices and challenges, Comput. Hum. Behav. 51 (2015) 915–929.
[76] S. Bilgaiyan, S. Mishra, M. Das, A review of software cost estimation in agile software development using soft computing techniques, in: Computational Intelligence and Networks (CINE), 2016 2nd International Conference on, IEEE, 2016, pp. 112–117.
[77] Y. Zhu, Requirements engineering in an agile environment. Uppsala University J. Inayat et al., Comput. Hum. Behav. 30 (2014) (2009).
[78] B. Ramesh, L. Cao, R. Baskerville, Agile requirements engineering practices and challenges: an empirical study, Inf. Syst. J. 20 (5) (2010) 449–480.
[79] L. Jun, W. Qiuzhen, G. Lin, Application of agile requirement engineering in modest-sized information systems development, in: Software Engineering (WCSE), 2010 Second World Congress on, vol. 2, IEEE, 2010, pp. 207–210.
[80] M. Daneva, et al., Agile requirements prioritization in large-scale outsourced system projects: an empirical study, J. Syst. Softw. 86 (5) (2013) 1333–1353.
[81] A. De Lucia, A. Qusef, Requirements engineering in agile software development, J. Emerging Technol. Web Intell. 2 (3) (2010) 212–220.
[82] N.A. Ernst, A. Borgida, I.J. Jureta, J. Mylopoulos, Agile requirements engineering via paraconsistent reasoning, Inf. Syst. 43 (2014) 100–116.
[83] K. Boness, R. Harrison, Goal sketching: towards agile requirements engineering, in: Software Engineering Advances, 2007. ICSEA 2007. International Conference on, IEEE, 2007, p. 71.
[84] D. Carlson, P. Matuzic, Practical agile requirements engineering, in: Proceedings of the 13th Annual Systems Engineering Conference, 2010.
[85] D.M. Berry, "The inevitable pain of software development, including of extreme programming, caused by requirements volatility," Eberlein and Leite, 2002.
[86] M. Fowler, "Refactoring: Improving the Design of Existing Code. 2000," http://www.martinfowler.com/books.html/refactoring, 2003.
[87] R. Carlson, P. Matuzic, R. Simons, Applying Scrum to Stabilize Systems Engineering Execution (2012) 1–6.
[88] M.R. Basirati, H. Femmer, S. Eder, M. Fritzsche, A. Widera, Understanding changes in use cases: a case study, in: Requirements Engineering, 2015., Proceedings of IEEE International Symposium on, 2015.
[89] J. Buckley, T. Mens, M. Zenger, A. Rashid, G. Kniesel, Towards a taxonomy of software change, J. Softw. Maint. Evol. 17 (5) (2005) 309–332.
[90] L.C. Briand, Y. Labiche, L. Sullivan, Impact analysis and change management of UML models, in: Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on, IEEE, 2003, pp. 256–265.
[91] S.D.P. Harker, K.D. Eason, J.E. Dobson, The change and evolution of requirements as a challenge to the practice of software engineering, IEEE International Symposium on Requirements Engineering, 1993.
[92] N. Nurmuliani, D. Zowghi, S. Fowell, Analysis of requirements volatility during software development life cycle, in: Australian Software Engineering Conference, 2004, p. 28.
[93] N. Nurmuliani, D. Zowghi, S.P. Williams, Using card sorting technique to classify requirements change, in: Requirements Engineering Conference, 2004, pp. 240–248.
[94] X. Hua, Q. Jin, Z. Ying, Supporting change impact analysis for service oriented business applications, in: Systems Development in SOA Environments, 2007. SDSOA '07: ICSE Workshops 2007, 2007, p. 6.
[95] C. Gupta, Y. Singh, D. Chauhan, A dynamic approach to estimate change impact using type of change propagation, J. Inf. Process. 6 (4) (2010).

[96] G.E. Stark, P. Oman, A. Skillicorn, A. Ameele, An examination of the effects of requirements changes on software maintenance releases, J. Softw. Maint. 11 (5) (1999) 293–309.
[97] C. Gupta, Y. Singh, D.S. Chauhan, A dynamic approach to estimate change impact using type of change propagation, JIPS 6 (4) (2010) 597–608.
[98] B. Nuseibeh, S. Easterbrook, Requirements engineering: a roadmap, in: presented at the Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, 2000.
[99] S. Nurcan, J. Barrios, G. Grosz, C. Rolland, Change process modelling using the EKD-change management method, in: European Conference on Information Systems, 1999, pp. 513–529.
[100] W. Lam, M. Loomes, Requirements evolution in the midst of environmental change: a managed approach, in: Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering, 1998, pp. 121–127.
[101] E.F. Ecklund Jr., L.M. Delcambre, M.J. Freiling, Change cases: use cases that identify future requirements, ACM SIGPLAN Not. 31 (10) (1996) 342–358 ACM.
[102] M. Pichler, H. Rumetshofer, W. Wahler, Agile requirements engineering for a social insurance for occupational risks organization: a case study, in: Requirements Engineering, 14th IEEE International Conference, IEEE, 2006, pp. 251–256.
[103] Z. Racheva, M. Daneva, A. Herrmann, A conceptual model of client-driven agile requirements prioritization: results of a case study, in: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, 2010, p. 39.
[104] S. Ibrahim, N.B. Idris, M. Munro, A. Deraman, A requirements traceability to support change impact analysis, Asian J. Inf. Technol. 4 (4) (2005) 345–355.
[105] Å. Dahlstedt, A. Persson, Requirements interdependencies: state of the art and future challenges, in: A. Aurum, C. Wohlin (Eds.), Engineering and Managing Software Requirements, Springer Berlin Heidelberg, 2005, pp. 95–116.
[106] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, J. Natt och Dag, An industrial survey of requirements interdependencies in software product release planning, in: Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on, 2001, pp. 84–91.
[107] B. Regnell, B. Paech, A. Aurum, C. Wohlin, A. Dutoit, and J. Natt och Dag, "Requirements Mean Decisions! - Research issues for understanding and supporting decision-making in Requirements Engineering," 2001.
[108] I. Sommerville, G. Kotonya, in: Requirements Engineering: Processes and Techniques, John Wiley & Sons, Inc., 1998, p. 282.
[109] K. Pohl, in: Process-Centered Requirements Engineering, John Wiley & Sons, Inc., 1996, p. 342.
[110] S.A. Bohner, Software Change Impact Analysis, 1996.
[111] O. Gotel, A. Finkelstein, Extended requirements traceability: results of an industrial case study, in: Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on, IEEE, 1997, pp. 169–178.
[112] M.F. Bashir, M.A. Qadir, Traceability techniques: a critical study, in: 2006 IEEE International Multitopic Conference, IEEE, 2006, pp. 265–268.
[113] F. Bouquet, E. Jaffuel, B. Legeard, F. Peureux, M. Utting, Requirements traceability in automated test generation: application to smart card software validation, ACM SIGSOFT Softw. Eng. Notes 30 (4) (2005) 1–7 ACM.
[114] J. Dick, Design traceability, IEEE Softw. 22 (6) (2005) 14–16.
[115] A. Egyed, P. Grunbacher, Automating requirements traceability: beyond the record & replay paradigm, in: Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on, IEEE, 2002, pp. 163–171.
[116] M. Heindl, S. Biffl, A case study on value-based requirements tracing, in: Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2005, pp. 60–69.
[117] M. Jarke, Requirements tracing, Commun. ACM 41 (12) (1998) 32–36.
[118] B. Ramesh, M. Jarke, Toward reference models for requirements traceability, IEEE Trans. Softw. Eng. 27 (1) (2001) 58–93.
[119] R. Ravichandar, J.D. Arthur, and M. Pérez-Quiñones, "Pre-requirement specification traceability: bridging the complexity gap through capabilities," *arXiv preprint* cs/0703012, 2007.
[120] S. Rochimah, W.M. Wan-Kadir, A.H. Abdullah, An evaluation of traceability approaches to support software evolution, in: ICSEA, 2007, p. 19.
[121] T. Verhanneman, F. Piessens, B. De Win, W. Joosen, Requirements traceability to support evolution of access control, ACM SIGSOFT Softw. Eng. Notes 30 (4) (2005) 1–7 ACM.
[122] P. Arkley, S. Riddle, Overcoming the traceability benefit problem, in: 13th IEEE International Conference on Requirements Engineering (RE'05), IEEE, 2005, pp. 385–389.
[123] J. Cleland-Huang, C.K. Chang, M. Christensen, Event-based traceability for managing evolutionary change, Softw. Eng. IEEE Trans. 29 (9) (2003) 796–810.
[124] J. Cleland-Huang, G. Zemont, W. Lukasik, A heterogeneous solution for improving the return on investment of requirements traceability, in: Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International, IEEE, 2004, pp. 230–239.
[125] J. Cleland-Huang, R. Settimi, C. Duan, X. Zou, Utilizing supporting evidence to improve dynamic requirements traceability, in: Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on, IEEE, 2005, pp. 135–144.
[126] O. Gotel, S. Morris, Crafting the requirements record with the informed use of media, in: Proceedings of the First International Workshop on Multimedia Requirements Engineering (MeRE'06), Citeseer, 2006.
[127] F. Blaauboer, K. Sikkel, M.N. Aydin, Deciding to adopt requirements traceability in practice, in: International Conference on Advanced Information Systems Engineering, Springer, 2007, pp. 294–308.
[128] J. Cleland-Huang, Toward improved traceability of non-functional requirements, in: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, ACM, 2005, pp. 14–19.
[129] B. Ramesh, Factors influencing requirements traceability practice, Commun. ACM 41 (12) (1998) 37–44.
[130] R.S. Arnold, S.A. Bohner, Impact analysis-towards a framework for comparison, in: ICSM, vol. 93, 1993, pp. 292–301.
[131] G. Antoniol, G. Canfora, G. Casazza, A.D. Lucia, Identifying the starting impact set of a maintenance request: a case study, in: Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European, IEEE, 2000, pp. 227–230.
[132] Y. Li, J. Li, Y. Yang, M. Li, Requirement-centric traceability for change impact analysis: a case study, in: Making Globally Distributed Software Development a Success Story, Springer, 2008, pp. 100–111.
[133] S. Ibrahim, N.B. Idris, M. Munro, A. Deraman, Integrating software traceability for change impact analysis, Int. Arab J. Inf. Technol. 2 (4) (2005) 301–308.
[134] A. Göknil, I. Kurtev, K. van den Berg, Change impact analysis based on formalization of trace relations for requirements, presented at the ECMDA Traceability Workshop (ECMDA-TW), Berlin, Germany, 12 June 2008, 2008.
[135] A. Von Knethen, Change-oriented requirements traceability. Support for evolution of embedded systems, in: Software Maintenance, 2002. Proceedings. International Conference on, IEEE, 2002, pp. 482–485.
[136] N. Ali, R. Lai, A method of requirements change management for global software development, Inf. Softw. Technol. 70 (2016) 49–67.
[137] J. Hassine, J. Rilling, J. Hewitt, R. Dssouli, Change impact analysis for requirement evolution using use case maps, in: Principles of Software Evolution, Eighth International Workshop on, IEEE, 2005, pp. 81–90.
[138] J. Hewitt, J. Rilling, A light-weight proactive software change impact analysis using use case maps, in: Software Evolvability, 2005. IEEE International Workshop on, IEEE, 2005, pp. 41–46.
[139] L. Shi, Q. Wang, M. Li, Learning from evolution history to predict future requirement changes, in: Requirements Engineering Conference (RE), 2013 21st IEEE International, IEEE, 2013, pp. 135–144.
[140] J.C. Maxwell, A.I. Antón, P. Swire, Managing changing compliance requirements by predicting regulatory evolution, in: Requirements Engineering Conference (RE), 2012 20th IEEE International, IEEE, 2012, pp. 101–110.
[141] R. Malhotra, M. Khanna, Mining the impact of object oriented metrics for change prediction using machine learning and search-based techniques, in: Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on, IEEE, 2015, pp. 228–234.
[142] C. Ingram, S. Riddle, Using early stage project data to predict change-proneness, in: Proceedings of the 3rd International Workshop on Emerging Trends in Software Metrics, IEEE Press, 2012, pp. 42–48.
[143] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, Predicting the probability of change in object-oriented systems, IEEE Trans. Softw. Eng. 31 (7) (2005) 601–614.
[144] S. Mirarab, A. Hassouna, L. Tahvildari, Using Bayesian belief networks to predict change propagation in software systems, in: Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on, IEEE, 2007, pp. 177–188.
[145] N.N.B. Abdullah, S. Honiden, H. Sharp, B. Nuseibeh, D. Notkin, Communication patterns of agile requirements engineering, in: Proceedings of the 1st Workshop on Agile Requirements Engineering, ACM, 2011, p. 1.
[146] B. Haugset, T. Stalhane, Automated acceptance testing as an agile requirements engineering practice, in: System Science (HICSS), 2012 45th Hawaii International Conference on, IEEE, 2012, pp. 5289–5298.
[147] R. Popli, P. Malhotra, N. Chauhan, Estimating regression effort in agile environment, Int. J. Comput. Sci. Commun. 5 (2014) 23–28.
[148] M. Cohn, User Stories Applied: For Agile Software Development, Addison-Wesley Professional, 2004.
[149] H. Leung, Z. Fan, Software cost estimation, in: Handbook of Software Engineering, Hong Kong Polytechnic University, 2002, pp. 1–14.
[150] S. Rajper, Z.A. Shaikh, Software development cost estimation: a survey, Indian J. Sci. Technol. 9 (31) (2016).
[151] N. Fenton, J. Bieman, Software Metrics: A Rigorous and Practical Approach, CRC Press, 2014.
[152] M.H. Halstead, Elements of Software Science, Elsevier, New York, 1977.
[153] P.G. Hamer, G.D. Frewin, MH Halstead's software science-a critical examination, in: Proceedings of the 6th International Conference on Software Engineering, IEEE Computer Society Press, 1982, pp. 197–206.
[154] V.Y. Shen, S.D. Conte, H.E. Dunsmore, Software science revisited: a critical analysis of the theory and its empirical support, IEEE Trans. Softw. Eng. (2) (1983) 155–165.
[155] A.J. Albrecht, J.E. Gaffney, Software function, source lines of code, and development effort prediction: a software science validation, IEEE Trans. Softw. Eng. (6) (1983) 639–648.
[156] C. Jones, Applied Software Measurement: Assuring *Productivity and Quality*, 1997.
[157] S. Kumari, S. Pushkar, Performance analysis of the software cost estimation methods: a review, Int. J. Adv. Res. Comput. Sci. Softw. Eng. 3 (7) (2013).
[158] P. Abrahamsson, I. Fronza, R. Moser, J. Vlasenko, W. Pedrycz, Predicting development effort from user stories, in: Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on, IEEE, 2011, pp. 400–403.

[159] M. Ceschi, A. Sillitti, G. Succi, S. De Panfilis, Project management in plan-based and agile companies, IEEE Softw. 22 (3) (2005) 21–27.
[160] N.C. Haugen, An empirical study of using planning poker for user story estimation, in: Agile Conference, 2006, IEEE, 2006, pp. 9–34, doi:10.1109/AGILE.2006.16.
[161] V. Mahnič, T. Hovelja, On using planning poker for estimating user stories, J. Syst. Softw. 85 (9) (2012) 2086–2095.
[162] S.K. Khatri, S. Malhotra, P. Johri, Use case point estimation technique in software development, in: Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO), 2016 5th International Conference on, IEEE, 2016, pp. 123–128.
[163] N. Nunes, L. Constantine, R. Kazman, IUCP: estimating interactive-software project size with enhanced use-case points, IEEE Softw. 28 (4) (2011) 64–73.
[164] E. Coelho, A. Basu, Effort estimation in agile software development using story points, Int. J. Appl. Inf. Syst. 3 (7) (2012).
[165] P.R. Hill, Practical Project Estimation: A Toolkit for Estimating Software Development Effort and Duration, International Software Benchmarking Standards Group, 2010.
[166] A. Panda, S.M. Satapathy, S.K. Rath, Empirical validation of neural network models for agile software effort estimation based on story points, Procedia Comput. Sci. 57 (2015) 772–781.
[167] A.G. Silvius, Business & IT alignment in theory and practice, System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on, IEEE, 2007 211b-211b.
[168] B. Campbell, Alignment: resolving ambiguity within bounded choices, in: PACIS 2005 Proceedings, 2005, p. 54.
[169] P. Tallon, K.L. Kraemer, A Process-Oriented Assessment of the Alignment of Information Systems and Business Strategy: Implications for IT Business Value, Center for Research on Information Technology and Organizations, 1999.
[170] A. Fuggetta, A.L. Wolf, Software Process, John Wiley & Sons, Inc., 1996.
[171] E.J. Barry, T. Mukhopadhyay, S.A. Slaughter, Software project duration and effort: an empirical study, Inf. Technol. Manage. 3 (1-2) (2002) 113–136.
[172] S. Bohner, Impact analysis in the software change process: a year 2000 perspective, in: Software Maintenance 1996, Proceedings., International Conference on, IEEE, 1996, pp. 42–51.
[173] R. Chitchyan, A. Rashid, P. Rayson, R. Waters, Semantics-based composition for aspect-oriented requirements engineering, in: Proceedings of the 6th International Conference on Aspect-Oriented Software Development, ACM, 2007, pp. 36–48.
[174] V. Basili, et al., Bridging the gap between business strategy and software development, in: ICIS 2007 Proceedings, 2007, p. 25.
[175] T. Goradia, Dynamic impact analysis: a cost-effective technique to enforce error-propagation, ACM SIGSOFT Softw. Eng. Notes 18 (3) (1993) 171–181 ACM.
[176] J. Law, G. Rothermel, Whole program path-based dynamic impact analysis, in: Software Engineering, 2003. Proceedings. 25th International Conference on, IEEE, 2003, pp. 308–318.
[177] P. Tonella, Using a concept lattice of decomposition slices for program understanding and impact analysis, Softw. Eng. IEEE Trans. 29 (6) (2003) 495–509.
[178] M. Aoyama, Agile software process and its experience, in: Software Engineering, 1998. Proceedings of the 1998 International Conference on, IEEE, 1998, pp. 3–12.
[179] S. Nerur, R. Mahapatra, G. Mangalaraj, Challenges of migrating to agile methodologies, Commun. ACM 48 (5) (2005) 72–78.
[180] P. Karesma, Scaling Agile Methods, 2016.
[181] D.J. Reifer, F. Maurer, H. Erdogmus, Scaling agile methods, IEEE Softw. 20 (4) (2003) 12–14.
[182] F.J. Pino, O. Pedreira, F. García, M.R. Luaces, M. Piattini, Using scrum to guide the execution of software process improvement in small organizations, J. Syst. Softw. 83 (10) (2010) 1662–1677.
[183] D.E. Strode, S.L. Huff, A. Tretiakov, The impact of organizational culture on agile method use, in: System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on, IEEE, 2009, pp. 1–9.
[184] E. Bjarnason, K. Wnuk, B. Regnell, A case study on benefits and side-effects of agile practices in large-scale requirements engineering, in: Proceedings of the 1st Workshop on Agile Requirements Engineering, ACM, 2011, p. 3.