IET Software

Review Article

Dynamic software updating: a systematic mapping study

ISSN 1751-8806
Received on 30th July 2019
Revised 21st December 2019
Accepted on 16th July 2020
E-First on 2nd September 2020
doi: 10.1049/iet-sen.2019.0201
www.ietdl.org

Babiker Hussien Ahmed[1]*, Sai Peck Lee*[1] ✉*, Moon Ting Su*[1]*, Abubakar Zakari*[1]

[1]*Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia*

✉ *E-mail: saipeck@um.edu.my*

Abstract: Dynamic software updating (DSU) is shifting gears to modify software systems without a halt. Even though extensive research has been conducted on DSU, it is necessary to synthesise and map the results of recent studies on DSU for prospective research highlights. This study aims to highlight the current state-of-the-art, to recognise trends, and to identify existing open issues in DSU. A systematic mapping study (SMS) was conducted with a set of six research questions. A total of 1066 papers published from 2005 to 2019 were recorded. After a filtering process, 112 primary studies were selected and inspected. This study highlights the current state-of-the-art of DSU including approaches, tools, models, and techniques. Also, this study outlines application domains, research type, and contributions type facets in DSU. In addition, this study demonstrates benchmarks, datasets, evaluation metrics, and existing open issues for future research in DSU. The results of this investigation can be used to highlight current state-of-the-art of DSU, to show trends of DSU, and to demonstrate existing open issues in DSU.

1 Introduction

Dynamic software updating (DSU) plays an important role in the maintenance and evolution of software systems [1–3]. Generally, DSU is a process of modifying running software systems without a termination [3–7]. Currently, there are growing appeals for DSU to support the continuous availability of the deployed software systems [8, 9].

Over time, extensive literature has developed on DSU [1, 10–13]. In short, DSU has been discussed by a great number of authors for decades [14, 15]. Furthermore, recent evidence highlights that DSU is one of the top-most features desired by developers and users [16]. Despite this interest, no one as far as we know has investigated DSU employing a systematic mapping method.

For this study, it was of interest to investigate DSU using a systematic mapping study (SMS). Previous studies [17–20] have emphasised that SMS is a secondary study that aims to outline a research domain, to identify current trends, and to provide lights for future research in a research domain [17, 20].

In this investigation, an SMS was conducted with a set of six research questions (RQs), a total of 1066 studies published from 2005 to 2019 were identified, after a filtering process, a total of 112 primary studies were selected and inspected.

It is interesting to note that, the results of this study cast a new light on aspects of DSU research topics involved in the design and execution of DSU. In short, this paper has the following contributions:

- Highlight the current state-of-the-art of DSU including approaches, tools, models, and techniques.
- Outline application domains, research type facets, and contributions type facets in DSU.
- Demonstrate benchmarks, datasets, evaluation metrics, algorithms, programming languages, and existing open issues for future research in DSU.

This paper is organised as follows. Section 2 highlighted related work and motivation. Section 3 describes the research method. Section 4 presents the results and discussion. Section 5 presents the principal findings. Section 6 discusses the threats to the validity of this study. Section 7 presents the concluding remarks and future work.

2 Related work and motivation

This section presents the related work and justifies the current need for a systematic mapping study in dynamic software updating.

2.1 Related work

In this section, it was of interest to highlight the studies that have been conducted to review DSU [1, 10–13] as follows:

DSU is an active research area [10], but its use in the industry is lacking, this motivated Ilvonen *et al.* [10] to review SWEBOK (IEEE's Guide to the Software Engineering Body of Knowledge) and investigated (Software Engineering Education Knowledge reference curricula) to find out how these guides acknowledge DSU techniques. Ilvonen *et al.* [10] found that although DSU is not mentioned explicitly in the reviewed guides, the need is well motivated and many DSU concepts are implicitly supported [10].

Seifzadeh *et al.* [11] developed a framework to review DSU studies. Seifzadeh *et al.* [11] discussed DSU techniques and evaluation metrics. Also, Gregersen *et al.* [12] compared three DSU systems developed for Java and investigated state-of-the-art of DSU in Java using selected target programs as benchmarks and discussed the implications [12].

Mugarza *et al.* [13] presented an analysis of existing DSU techniques and focused on DSU for increasing safety, security, and availability of the Industrial Internet of Things (IIoT) systems. Moreover, Miedes and Muñoz-Escoí [1] reviewed a set of DSU studies in a technical report. The study in [1] was focused on DSU for software maintenance [1].

2.2 Need for an SMS

Although several studies have reviewed DSU [1, 10–13], our research group has found none of the SMS published in the area. Conspicuously, a direct search in the web search engines for (‘systematic mapping study’ OR ‘systematic literature review’) AND ‘dynamic software updating’ has shown that no study has tried to investigate DSU in a systematic review (SR).

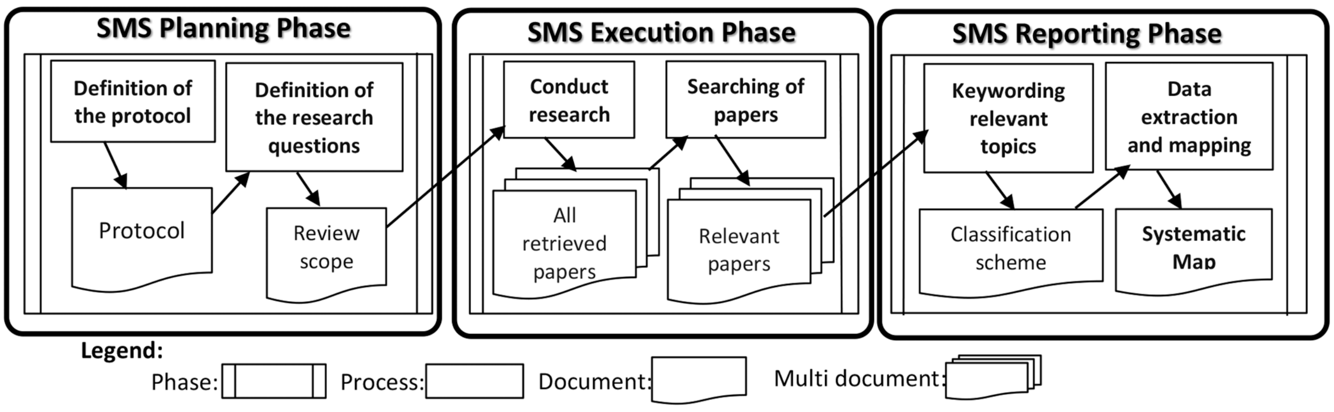


Fig. 1 *Systematic mapping study process [17, 19, 20]*

Table 1 Research questions and objectives

#	Research question	Objective
RQ 1	What approaches and tools existing in dynamic software updating?	To identify existing dynamic software updating approaches and tools.
RQ 2	What techniques and models existing in dynamic software updating ?	To recognise existing dynamic software updating techniques and models.
RQ 3	What programming languages and algorithms employed in dynamic software updating?	To identify programming languages and algorithms employed in dynamic software updating.
RQ 4	Which application domains have dynamic software updating been applied?	To recognise current dynamic software updating application domains.
RQ 5	What target programs, evaluation metrics, datasets, and open issues existing in dynamic software updating ?	To identify target programs, evaluation metrics, datasets and open issues existing in dynamic software updating.
RQ 6	What research type facets and contribution type facets focused on dynamic software updating studies?	To identify research type facets being focused on DSU studies and contribution type facets they provided.

("Dynamic software updating") OR (DSU) OR
("Dynamic software updates") OR ("Dynamic software update")

Fig. 2 *Defined search string*

An SR is 'a secondary study that reviews primary studies with the aim of synthesising evidence related to a specific research' [21]. In general, there are several forms of SR [18, 21] as follows:

- *Systematic literature review (SLR):* uses a well-defined methodology to identify, analyse and interpret available evidence related to a specific RQ in a way that is unbiased and to a degree repeatable [18, 21, 22].
- *Systematic mapping study (SMS):* is a broad review of primary studies in a specific research area to identify available evidence in the domain [18, 21].
- *Tertiary review:* is a systematic review of systematic reviews [18, 21].

The majority of prior research has applied SMS to gather and aggregate current evidence in a research area [23]. Also, 'A very broad topic' is one of the reasons for appointing an SMS over an SLR [18, 21]. Following that, and the inherent broadness of DSU, we selected to conduct an SMS on DSU instead of SLR. Furthermore, this research provides an exciting opportunity to advance our knowledge of DSU using an SMS in the domain.

3 Research method

In this study, the well-known SMS method for software engineering established by Petersen *et al.* [17, 19], Kitchenham and Charters [18], and Budgen *et al.* [24] was utilised to conduct this investigation. As shown in Fig. 1, this study was conducted in three phases (i.e Planning, Execution, and Reporting) [17, 19, 20]. The Planning and Execution phases are highlighted in the following sub-sections, and the Reporting Phase is presented in Section 4.

3.1 SMS planning phase

Previous studies have recommended using an SMS planning protocol [19, 25] for conducting SMS. Therefore, our research group has employed an SMS planning protocol. In fact, the SMS planning protocol useful to:

- Organise how the selected studies to answer the RQs [19, 25].
- Guide the research objective and mark out how it should be fulfilled [19, 25].

In this investigation, the SMS planning protocol includes: (i) the research scope, objective and questions; (ii) the structure of the search string; (iii) primary studies selection process; (iv) data extraction and management strategy; and (v) synthesis methods.

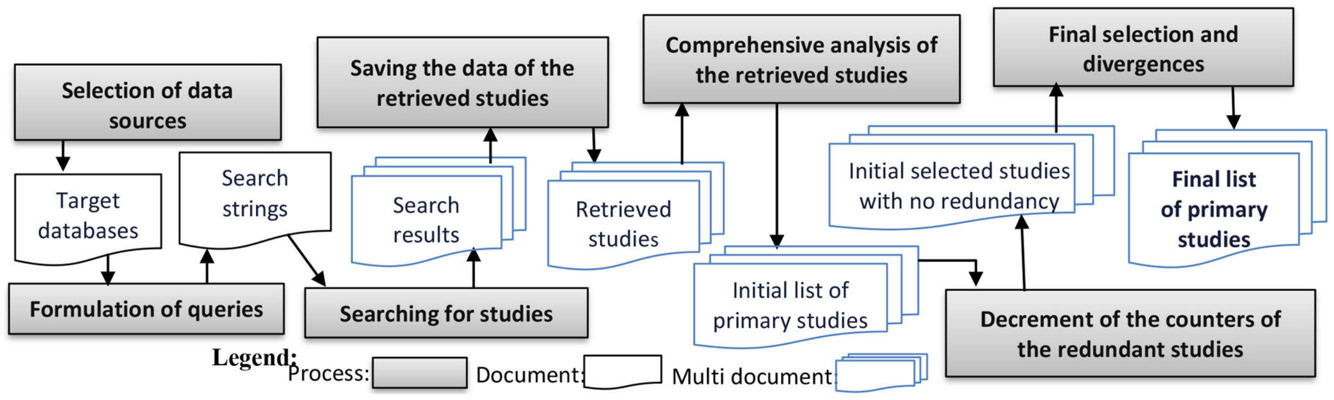
3.1.1 Research scope, objective and questions: This study aims to explore current state-of-the-art, to recognise trends, and to identify existing open issues in DSU. Our RQs and objectives are presented in Table 1.

3.1.2 Structure of the search string: As has been previously reported in the literature, a search string is required to be defined in an SMS to fetch the studies from the research inventory databases [18]. To define a search string, some authors have recommended structuring the SMS RQs based on PICO (Population, Intervention, Comparison, and Outcome) criteria [18, 19].

Conspicuously, PICO criteria are useful to find out the search keywords [18, 19], to formulate the search query strings [18, 19], and to sustain the consistency of search along with various data sources [18, 19]. For this study, PICO criteria were used to define the search string in Fig. 2 as follows:

- *Population:* Published studies on software.
- *Intervention:* Dynamic software updating.
- *Comparison:* Not applied.
- *Outcome:* Published studies on DSU.

3.1.3 Primary studies selection process: For this research, Fig. 3 shows the utilised primary studies selection process [18–20, 26–28]. For the primary studies selection process, several studies suggest utilising Inclusion and Exclusion Criteria (IC, EC) [17, 19] to support the choice of primary studies [17, 19, 20] and to sustain the final choice of primary studies [17, 19, 20]. As a result, Table 2 demonstrates the IC, EC for the current investigation.



Selection of data sources
Saving the data of the retrieved studies
Comprehensive analysis of the retrieved studies
Final selection and divergences
Target databases
Search strings
Search results
Retrieved studies
Initial selected studies with no redundancy
Final list of primary studies
Formulation of queries
Searching for studies
Initial list of primary studies
Decrement of the counters of the redundant studies
Legend:
Process:
Document:
Multi document

Fig. 3 *Primary studies selection process [18–20, 26–28]*

Table 2 Inclusion and exclusion criteria

Exclusion criteria	
EC 1	The study does not address any part of dynamic software updating contribution.
EC 2	The study does not include presentation analysis of results.
EC 3	The study is not related to dynamic software updating.
EC 4	The primary study is not published from 2005 to 2019.
EC 5	The study is not written in the English language
Inclusion criteria	
IC 1	The primary study must propose theory/guideline/lessons learnt/ advice for dynamic software updating.
IC 2	The primary study must address the use of approach/ technique/ method/ algorithm/ methodology/ metric/model/ language/ strategy to enhance or support dynamic software updating.
IC 3	The primary study must propose a tool/ framework/prototype/ architecture/ platform/ process to enhance or support dynamic software updating.

As shown in Table 2, a study will be excluded if it meets one of the defined execution criteria. If it is not excluded, and it meets one of the defined inclusion criteria, it will be included in the analysis.

3.1.4 Data extraction and management strategy: In this analysis, data collection and management are organised as follows:

(i) *Data collection method:*A data extraction form is introduced and utilised for data collection. The data extraction form is designed based on the RQs. It includes the following:

- *Approaches and tools for RQ 1:*

  ○ *Approach*: it explains how DSU would be carried out.
  ○ *Tool*: is any tool (DSU system) that is used or proposed to be used for DSU.
- *Techniques and models for RQ 2:*

  ○ *Model*: is abstracted to specify DSU [29].
  ○ *Technique:* is a way of carrying out a DSU task, especially the execution or performance of a DSU procedure.
- *Programming languages and algorithms for RQ 3:*

  ○ *Programming languages:* is any programming language utilised or proposed to utilise DSU.
  ○ *Algorithm*: is an algorithm used in DSU or proposed to be utilised for DSU.
- *Application domains for RQ 3:* is any field utilising or proposed to utilise DSU.
- *Evaluation metrics, target programs, datasets, and Open issues for RQ 5:*

  ○ *Evaluation metrics:* is a measure used to assess implementing DSU [1].
  ○ *Target program:* is a benchmark program that used to implement and assess DSU [6].
  ○ *Dataset*: is a benchmark that includes target programs used to evaluate DSU [30].
  ○ *Open issue:* is existing open research gap or problem discussed in DSU.
- *Research type facets and contribution type facets for RQ 6:*

  ○ *Research type facet:* is general and independent of a specific research area [17, 19, 31]. Examples of research type facets categories in Table 3.
  ○ *Contribution type facet:* is referred to as the type of intervention being studied [17, 19, 20, 32].

(ii) *Data management strategy:* A number of authors recommended using software tools such as spreadsheets and reference management to manage SMS data [18, 19, 28]. Therefore, spreadsheets programs and EndNote [33] software tool were nominated to manage the data of this study.

3.1.5 Synthesis methods: For this study, quantitative methods were nominated to synthesise the data through the following:

(i) Counting the number of studies per year in each answer.
(ii) Utilising a bubble plot chart [34] to report the frequencies of research trends in RQs [17, 20].

3.2 SMS execution phase

In this phase, our planning protocol was implemented, a classification scheme was utilised, and the data was extracted and managed as follows.

3.2.1 Primary studies selection process: In this phase, our primary studies selection process was executed and we obtained the following results:

(i) *Selection of the data sources:* A number of authors have recommended that the use of IEEE and ACM and two indexing databases are sufficient for SMS [19, 26, 27]. Accordingly, five databases were selected including ACM, IEEE Explore, Wiley, Springer link and Science directs.
(ii) *Formulation of queries:* For each of the selected databases, a search string was formulated using the defined string in Fig. 2. The search strings of this study are presented in Table 4.
(iii) *Searching for studies:* A total of 1066 studies published from 2005 to 2019 were recorded. The details of the recorded studies per database are displayed in Table 5.
(iv) *Saving the data of the retrieved studies:* The obtained 1066 studies were downloaded and adapted in EndNote software [33]. The records of the retrieved studies contain the year, source, title, and the author's fields.

(v) *Comprehensive analysis of the retrieved studies:* In this investigation, we implemented the following:

Table 3 Research type facets [17]

Type	General description
validation research	'Techniques investigated are novel and have not yet been implemented in practice. Techniques used are for example experiments, i.e. work done in the lab' [17].
evaluation research	'Techniques are implemented in practice, and an evaluation of the technique is conducted. That means, it is shown how the technique is implemented in practice (solution implementation) and what are the consequences of the implementation in terms of benefits and drawbacks (implementation evaluation). This also includes identifying problems in the industry' [17].
solution proposal	'A solution to a problem is proposed, the solution can be either novel or a significant extension of an existing technique. The potential benefits and the applicability of the solution are shown by a small example or a good line of argumentation' [17].
philosophical papers	'These papers sketch a new way of looking at existing things by structuring the field in the form of taxonomy or conceptual framework' [17].
opinion papers	'These papers express the personal opinion of somebody whether a certain technique is good or bad, or how things should have been done. They do not rely on related work and research methodologies' [17].
experience papers	'Experience papers explain on what and how something has been done in practice. It has to be the personal experience of the author' [17].

Table 4 Search strings and selected databases

Database	Search string
ACM	acmdlTitle:(+ Dynamic + Software + update) AND (Dynamic Software update).
IEEE Explore	('Document Title': Dynamic Software Update OR 'Abstract': Dynamic Software Update).
Science Direct	TITLE (Dynamic Software update) or ABSTRACT (Dynamic Software Update).
Springer link	'"Dynamic Software Update OR Dynamic Software Updating OR Dynamic Software Updates"'.
Wiley	Dynamic Software Update in Article Titles OR Dynamic Software Update in Abstract between years 2005 and 2019.

- *Pre-intermediate selection:* As demonstrated in Table 2, EC 4 and EC 5 were used to exclude studies. Also, we read the title, abstract, introduction, and conclusion of the retrieved studies under EC1, EC 2 and EC 3.
- *Intermediate selection:* For each study, we read the title, abstract, introduction, and conclusion, and then select or reject following IC 3, IC 2, IC1, EC 2 and EC 1.

(vi) *Decrement of the counters of the redundant studies:* In this activity, we implemented the following:

- The redundant titles were hunted using the sort feature in EndNote software [33].
- For studies with redundant titles, we read the abstract, the introduction, and finally, one version was kept.

(vii) *Final selection and divergences:* In this study, we read a total of 163 studies. Finally, 112 primary studies were selected for an SMS.

In summary, we present the results of the implemented primary studies selection process in Table 5.

3.2.2 Classification scheme: For this study, a classification scheme was utilised to classify and analyse the selected primary studies [17]. Most early studies, as well as current work, focus on implementing the SMS classification scheme using Keywording [17]. Keywording is reading the abstract, introduction, and conclusion of a paper to search for keywords. Fig. 4 shows the SMS classification scheme [17].

Previous research suggests classifying the selected primary studies in facets during classification scheme to understand them at an abstract level [17]. Therefore, for this study, as a result, a set of 13 main facets were produced for DSU including approaches, tools, techniques, models, programming languages, algorithms, application domains, open issues, evaluation metrics, datasets, target programs, research type facets, and contribution type facets.

Furthermore, as demonstrated in Fig. 5, for this analysis, we classified DSU techniques into four facets including update points [35], state transformation [36, 37], runtime [38], and safety [39].

3.2.3 Extraction and management of the SMS data: For this study, the data of this SMS were collected from the selected primary studies through the data extraction forms as demonstrated in Section 3.1.4. Also, prior research suggests reviewing the data to ensure the extraction of accurate data [19, 31]. Therefore, in this analysis, more than one researcher was involved in the data collection and checking.

In this study, a total of 12 data tables were created in a Spreadsheet to organise and analyse the data. The data tables are presented in Table 6.

4 Results and discussion

This section presents the results and discusses the answers to the research questions.

Table 5 Output of the process of the primary studies selection

Database	Activity								
	Searching for studies	Comprehensive analysis				Decrement of the counters of the redundant studies		Final selection and divergences	
		Pre –intermediate selection		Intermediate selection					
		Excluded	Selected	Excluded	Selected	Excluded	Selected	Excluded	Selected
ACM	253	146	107	65	42	3	39	5	34
IEEE explore	450	297	153	96	57	3	54	9	45
Springer link	94	45	49	18	31	0	31	13	18
Science direct	204	171	33	7	26	0	26	16	10
Wiley	65	39	26	13	13	0	13	8	5
Total	**1066**	**698**	**368**	**199**	**169**	**6**	**163**	**51**	**<u>112</u>**

The bold values are indicates the total number of studies.

The 'underlined 112' indicates the total number of the selected primary studies.

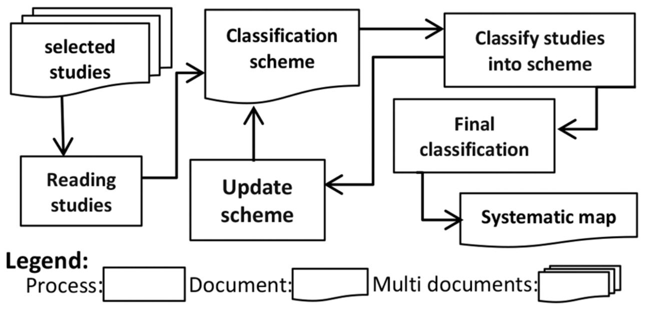


Fig. 4 *Classification scheme for systematic mapping study [17]*

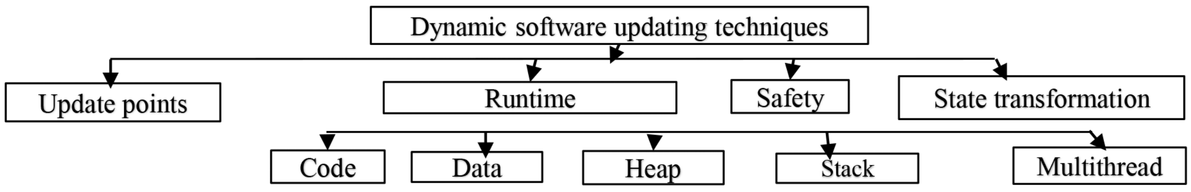


Fig. 5 *Classification of dynamic software updating techniques for a systematic mapping study*

Table 6 Data tables and research questions

RQ#	Data table	Target data
RQ 1	approaches	approaches
	tools	tools (DSU systems)
RQ 2	techniques	techniques
	models	models
RQ 3	algorithms	algorithms
	programming languages	programming languages
RQ 4	applications	application domains
RQ 5	open issues	open issues
	evaluation metrics	evaluation metrics
	datasets	datasets
	target programs	target programs
RQ 6	research type and contribution type	research type facets and contribution type facets

Table 7 Top six most cited DSU approaches

Approach	Primary studies
customised java virtual machine	[4, 16, 35, 37, 40–57]
dynamic reconfiguration	[5, 40, 43, 56, 58–72]
dynamic patch	[73–82]
hot-swapping	[30, 64, 69, 72, 83–86]
multi version programming	[3, 48, 60, 62, 83, 87–90]
customised class loaders	[38, 45, 46, 48, 52, 63, 89, 91]

RQ 1: What approaches and tools existing in dynamic software updating?

In this study, we have identified 23 DSU approaches. Based on the number of primary studies for each approach, we list the top six most cited DSU approaches in Table 7.

Also, number of authors have highlighted additional DSU approaches including dynamic replacement [22, 54, 66, 75, 86, 92, 93], dynamic instrumentation [52, 59, 60, 74, 84, 94], dynamic linking [37, 39, 53, 75, 83, 85], lust-in-time compilation [30, 44, 55, 58, 76, 95], wrapper [4, 40, 42, 44, 52], dynamic aspect-oriented [35, 41, 50, 96, 97], extension of features [98, 99], dynamic adaptation [43, 59], rolling update [84], object transformer [49], data-driven programming [92], distributed DSU [38], dynamic core library update [56], using run time to support DSU [92], procedure-oriented [30], micro-language-based [100], and dynamic rebinding [5].

To demonstrate an existing trend of DSU approaches, Fig. 6 is a bubble plot chart of DSU approaches by the publication years. In Fig. 6, the *Y*-axis represents the top six most cited approaches, the *X*-axis represents the publication years, and the intersection bubble is the reference numbers of the primary studies. The most interesting result is that multi-version programming, dynamic patch, and dynamic reconfiguration are continuing the most popular approaches in the domain.

In this exploration, we have identified a list of 55 tools (DSU systems). Based on the number of primary studies for each tool, we list the top ten most cited tools in Table 8.

In addition, number of studies have mentioned additional tools including Jrebel [42, 46, 48, 53, 91], LUCOS [22, 51, 76, 79, 94], Javelus [35, 40, 42, 44], JavAdaptor [16, 38, 46, 52], DVM [4, 42, 44, 52], HotSwap [16, 42, 50, 57], DynaMOS [51, 76, 78, 105], Dlpop [37, 39, 106], Mx [83, 88, 90], EcoDSU [106, 108], PROTEOS [51, 95], Ekiden [6, 105], lusagent [59, 84], DYMOS [22, 47], EmbedDSU [106, 112], UpgradeJ [4, 47], Prose [41, 57], REPLUS [76, 113], FASA [65, 106], and Adapt.net [59, 68].

Moreover, several investigations have also cited extra tools including Jikes RVM [4], ADSU [87], Jooflux [41], AspectJ [41], BSDiff [75], FastSwap [52], Podus [29], CURE [73], FiGA [46], DAS [22], Ksplice [37], Lasagne [52], Iguana/J [43], DUT [52], ISLUS [51], and JAC [52].

Furthermore, some authors have also mentioned other tools including LyRT [38], strong Update [74], MQTT [84], MUC [88], Tedsuto [111], Multics [22], uFlow [92], changeboxes [56], and KUPC [82].

RQ 2: What techniques and models existing in dynamic software updating?

On the question of DSU techniques, as classified in Fig. 5, we identified update points, runtime, safety, and state transformation techniques.

In this part, we present the results of DSU techniques as follows:

(i) Update points techniques

In this study, we have identified existing types of update points and techniques to nominate update points in DSU as follows:

A. Existing types of update points

A group of studies have presented 11 different types of update points including checkpoints [44, 49, 51, 74, 87, 114, 115], synchronisation points [3, 36, 47, 116], version-consistent points [35, 115], breakpoints [76], single join-points [96], injection points [43], scheduling points [5], rescue points [76], mapping points [76], writing object points [73], and version-consistent points [115].

B. Current techniques to nominate DSU update points

A number of studies have demonstrated 15 techniques to recommend update points including automatic definition [35, 73], insert checkpoints [51, 74], enumerate potential points [104, 107], expressing update points in the code [117], pre-compiled at fixed

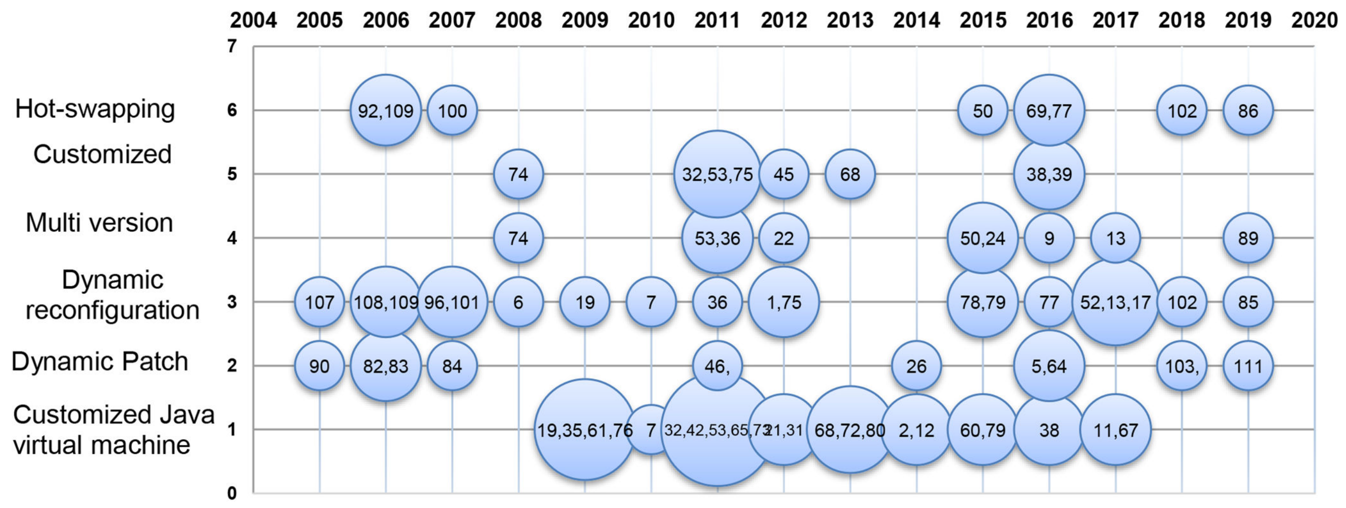


Fig. 6 *Bubble plot chart of the top six most cited dynamic software updating approaches by year*

Table 8 Top ten most cited tools (DSU system)

Tool	Primary studies
POLUS	[4, 6, 29, 39, 41–44, 47, 48, 51, 74, 76, 79, 83, 96, 97, 101–106]
Ginseng	[6, 29, 37, 39, 42, 44, 47, 48, 51, 74, 77, 94, 95, 101–103, 105–109]
UpStare	[6, 37, 42, 44, 51, 74, 76, 94, 103, 105, 106, 108]
Jvolve	[4, 35, 37, 41, 44, 46–49, 51–53]
Kitsune	[6, 22, 37, 51, 74, 76, 90, 94, 102, 105, 108]
OPUS	[6, 29, 51, 76, 101, 104–106]
DCE VM	[41, 42, 45, 48, 49, 53]
DUSC	[42–44, 93, 103, 110]
Javeleon	[22, 45, 46, 48, 52, 91]
Rubah	[38, 46, 49, 56, 59, 111]

Table 9 Top six most demonstrated DSU safety types

Safety type	Primary studies
type safety	[4–6, 29, 37, 42–49, 52, 54, 55, 64, 76, 80, 84, 85, 87, 89, 95–97, 101, 102, 104, 105, 109, 115, 116, 118, 119, 122–125]
update points safety	[4, 6, 35, 37, 42, 46, 48, 51, 69, 73, 77, 79–81, 83, 94, 99, 101, 103, 112, 115, 126]
state safety	[5, 9, 38, 43, 47, 48, 57, 63, 64, 116]
steady state safety	[4, 6, 41, 44, 45, 87]
thread safety	[45, 47, 51, 52, 59, 74]
activeness safety	[6, 73, 95, 107, 112]

points [104], bring the software to an update point [83], define update points implicitly [111], define update points dynamically [44], using collector and point filters [73], locating injection points [43], automatic recommendation [35], using continuation points [6], session typing [118], restrictions on runtime [118], and observing the heap state [39].

(ii) Runtime techniques

In this examination, we have identified an existing code segment, data segment, stack segment, heap segment, and multithread dynamic updating techniques as follows:

A. Code segment

A number of investigations have presented five types of code segment techniques including indirection [4, 46, 47, 53, 55, 74, 77, 94, 99, 106, 119], injection [46, 74, 79, 91, 103, 120], rewriting [47, 50, 53, 74, 106], bytecode manipulation [41, 43, 55], and trampolines [4, 106].

B. Data segment

A group of studies have demonstrated eight different types of data segment techniques including migration [56, 57, 105, 111, 119], shadow data structure [59, 74, 105], copy field values [41], swapping [95], type wrapping [105], screen buffering and data caching [73], Struct replacement [22], and shared data structure [68].

C. Stack segment

A group of examinations have highlighted four different types of stack segment techniques including stack reconstruction [6, 41, 42, 51, 74, 76, 94, 103, 105, 121], stack mapping [6, 42, 51, 76, 88, 94, 105, 121], stack inspection [5, 70, 79, 101] and on-stack-replacement [4, 42, 44, 76].

D. Heap segment

A number of explorations have demonstrated five different types of heap segment techniques including heap garbage collection [4, 42, 49, 63], virtual heap [114], map the heap from locations to values [115], modify heap bindings [115], and heap state transformations [87].

E. Multithread

In this exploration, we have recognised ten techniques utilised for multithread including achieving quiescence [3, 58, 72], immediate update of multi-thread [44, 51], scheduling [37, 67], constraint-based [115], pre-emptive multi-threading [93], utilising thread-shared locations [115], two-phase thread-based locking [83], utilising critical sections [104], blocking on condition variables [3], and controlling between threads [99].

(iii) Safety techniques

In this study, we have identified existing types of safety, techniques to define safety, and techniques to apply safety in DSU as follows:

A. Existing types of safety for DSU

In this study, we have identified 29 types of safety in DSU. Based on the number of primary studies for each type, we list the top six most demonstrated safety types in Table 9.

In addition, number of studies have presented other 23 types of safety including con-freeness safety [6, 73, 95], semantics safety [51, 76, 84], version consistency safety [64, 115], deadlock-free safety [112, 118], stack safety [76], reconcile safety [116], update positions safety [51], domain safety [53], change function safety [51], compile-time safety [54], composition-time safety [54], data access safety [76], deadlock safety [51], trace safety [115], disjunction safety [64], dynamic reconfiguration safety [5], memory safety [95], patch safety [101], Virtual Machine safety [4], safe transition state [69], safe trace [69], safe function redefinition [99], and safe reconfiguration point [70].

B. Current techniques to define safety for DSU

A group of studies have highlighted 19 different types of techniques to define safety for DSU including update points testing [37, 107], exhaustive testing [111], formal methods [101], checking assertion [95], program analysis [117], stack map checking [4], program monitoring [115], static safety checks [57], dodging unsafe points [46], flow-sensitive updateability analysis [77], marking the unsafe points [46], systematic testing [37],

Table 10 Top six most cited DSU techniques

Technique	Primary studies
state mapping	[9, 16, 43, 51, 52, 60, 94, 101, 104, 115, 122, 128]
indirection	[4, 46, 47, 53, 55, 74, 77, 94, 99, 106, 119]
sack reconstruction	[6, 41, 42, 51, 74, 76, 94, 103, 105, 121]
transformation function	[9, 36, 37, 41, 43, 83, 91, 95, 131]
stack mapping	[6, 42, 51, 76, 88, 94, 105, 121]
lazy state transfer	[6, 43, 45, 48, 52, 55, 116]

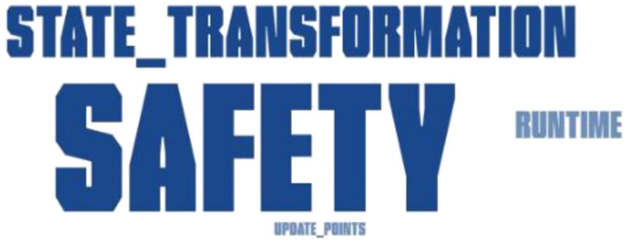


Fig. 7 *Word cloud of DSU techniques facets*

Table 11 Top five most cited models

Model	Primary studies
interrupt	[29, 80, 102, 104, 130, 132]
relax consistency	[29, 42, 83, 102, 104]
invoke	[29, 80, 102, 104]
indirection	[53, 74, 105]
formal model	[5, 9, 113],

updateability analysis [127], update-specific testing [111], interrupts to detect safety [118], verification of binary [53], operation-oriented testing [111], ensure correctness [35], and verification of architecture [53].

C. Existing techniques to apply safety of DSU

A number of explorations have also demonstrated a list of 15 techniques to apply safety in DSU including lazy state transfer [6, 43, 45, 48, 52, 55, 116], synchronisation [5, 87, 101, 116], check points [49, 74, 90, 106], inspection of the backwards-compatibility [39, 76, 128], rollback [51, 87, 129], data race detection [49, 115], safeguard execution [38, 40], dump heap snapshots [39, 73], preclude safe dynamic replacement [93], abort [49], automatic checks [107], maintain in-memory state [105], steady state execution [42], automatic repair [95], control-flow reboots [111].

Additionally, a group of selected primary studies have demonstrated a list of 20 safety techniques including state-loss prevention [52], scheduling [130], uniformly manage the stack and heap [76], using container interfaces [53], static type-checking [49], featuring type-safe execution [49], avoids deadlock and data-races [49], resolve symbol references and binary verification at design-time [53], strongly-typed interfaces [54], automatic recovery [51], dynamic instrumentation for rescue [76], verification of symbol [53], reconciliation [56], statically prevent a runtime dereference of dangling pointers [127], state replication [90], using control state [90], using monitor agent [129], reader-writer-locks [68], make copies of data by requiring a backward type transform [77], and track write access to the protected data using signal mechanism [79].

(iv) State transformation techniques

In this study, we have identified existing types of states, techniques to define states, and techniques for states transformations in DSU as follows:

A. Existing types of states

A group of studies have highlighted a group of 15 states that gets attention in DSU including initial state [9, 29, 36, 58, 64, 83, 95, 102, 104, 122, 128], quiescent state [5, 9, 40, 65, 83, 91, 112, 126], heap state [36, 39, 42, 87, 95, 116], object state [4, 5, 45, 116], shared state [3, 5], backward compatibility state [128], tranquil state [63], old-version state [39], buggy state [83], class state [45], components state [65], deadlock state [102], thread states [104], inconsistent state [56], and multi-cycle state [65].

B. Current techniques to define states

A number of studies have demonstrated 12 techniques to define states in DSU including state transformer [45, 73, 83, 95, 115], formalise as state machine [9, 61, 83, 120], adaptation [41, 65], prepare ready tainted states [101], state convergence algorithm [126], state-based specification language [83], enforcing restrictions [116], model checkers [83], using check points [37], custom state transfer [116], convert states bi-directionally [42], and event-based [66].

C. Existing techniques to transfer states in DSU

Before we list the identified states transformations techniques, it is interesting to mention number of techniques utilised in DSU to assist in states transformations in the selected primary studies, the techniques are including wrappers [16, 22, 43, 46, 52, 77, 94, 95], adaptation [4, 38, 41, 43, 59, 100, 129], refactoring [47, 50, 55, 74, 86, 99, 123], proxies [41, 46, 53, 113, 119, 131], class loaders [55, 85, 86, 123], in-Place Proxification [43, 55, 91], adjustment [41, 68], reflection [85, 86], and publisher subscriber pattern [66, 121].

Also, as demonstrated in the selected primary studies, other techniques are utilised to support states transformations including virtual classes [116], XML-based configuration [68], meta-object protocol [35], Object Translation Table [86], type name map file [80], type conversion file [80], position-independent code [53], relocatable code [53]. symbol resolution [53], interface code file [80], and producer/consumer pattern [118], modular solution [54], introspection [99], pre-compile [4], reconfiguration scheduler [67], Object replacement [105], shared libraries [53], and function cloning [78].

For the states transformations, a group of studies have highlighted state transformation techniques in DSU including state mapping [9, 16, 43, 51, 52, 60, 94, 101, 104, 115, 122, 128], transformation function [9, 36, 37, 41, 43, 83, 91, 95, 131], state migration [52, 55, 105, 114], targeted object synthesis [9, 35, 36], relinking [29, 117], primitive support [39, 116], synchronisation [93, 116], and dynamic loading [86, 91].

In addition, further studies demonstrated additional state transformation techniques including customisation [116], atomic transformation [83], switching [67], coexisting transformation [67], hybrid executions [128], state relocation [48], multi-cycle transfer [65], executing from initial states [104], transfer the state of components [126], dynamic reference using containers [131], transferring when load [126], garbage collector [118], change class hierarchy [16], automatically reach state [64], automating object transformations [36], replacements [41], multi-level mediator-based [85], dynamic object binding [38], and dynamic redefinition [55].

In summary of the identified DSU techniques, technique, Table 10 illustrates the top six most cited DSU techniques based on the number of selected primary studies for each technique.

Moreover, it is interesting to highlight the most discussed DSU techniques facets. Fig. 7 is a word cloud that demonstrates the discussed DSU technique facets based on the number of facets. As in Fig. 7, DSU safety is the most discussed followed by the runtime and state transformation facets. On the other hand, update points facets are less discussed facets in the domain.

In this study, we have identified a list of 27 models for DSU. Based on the number of primary studies for each model, we list the top five most cited models in Table 11.

Furthermore, the result shows other models including binary rewriting [53, 74], labelled transition systems [9, 69], incremental [83, 102], instantaneous [83, 102], mix objects object [42, 44], dynamic loading [97, 133], stack mapping [105], adaptation [64], aspect-based [96], class-loading [45], dynamic linking [53], relocatable code [53], event-based [9], multiple consistency [44], object space [56], dynamic rebinding [5], representation consistency [77], shared library [53], stack reconstruction [74], strict consistency [87], V-shape protocol [60], state machine [89],

Table 12 Top five most cited programming languages

Language	Primary studies
Java	[4–6, 16, 30, 35, 37–52, 54–57, 59, 60, 65, 67, 73, 76, 77, 84, 85, 89, 91, 96, 97, 99, 101, 104, 105, 109, 111–113, 116, 117, 122–124, 131, 134, 135]
C/C + +	[4–6, 22, 29, 37, 42–44, 46, 47, 51–53, 66, 70, 77, 80, 82, 88, 90, 94–97, 99, 101, 103–105, 107, 108, 110, 115, 117, 119, 124, 127, 131, 135]
Erlang	[37, 47, 61, 77, 117, 120]
C#	[4, 37, 59, 96]
Smalltalk	[37, 52, 56, 116]

Table 13 Top ten most discussed application domains

Application domain	Primary studies
internet-based applications	[3–6, 14, 16, 35–40, 42, 44, 51, 54, 60, 73, 74, 76, 81, 82, 84, 87–90, 94–96, 101–103, 105–107, 111, 114, 116, 120, 128–130, 132, 136]
operating systems	[3, 41, 72, 74, 75, 78, 92, 93, 95, 126, 127]
network applications	[54, 75, 92, 96, 110, 114, 120, 127, 132]
distributed systems	[40, 44, 50, 59, 61, 62, 93, 110, 118]
databases	[6, 41, 42, 46, 52, 73, 96, 111]
resource-constrained systems	[6, 53, 95, 103, 112, 126, 134]
time-critical systems	[6, 110, 126, 130, 132–134]
cyber-physical systems	[81, 82, 98, 103, 108, 121]

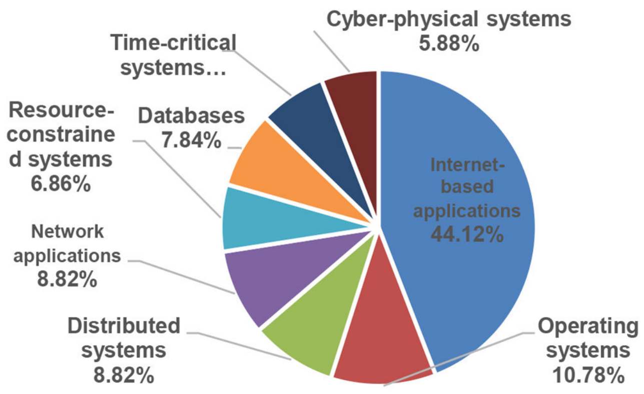


Fig. 8 *Most cited DSU application domains*

Table 14 Top six most demonstrated evaluation metrics

Metrics	Primary studies
update time	[3, 4, 9, 22, 35, 37, 38, 40–42, 44, 46, 47, 51, 54, 56, 58–60, 62, 73, 75, 76, 79, 80, 83, 90, 93, 95, 96, 99, 101, 106, 108, 109, 111, 115, 116, 120, 123, 127, 131, 133, 136]
Overhead	[6, 14, 22, 41, 42, 45, 51, 54, 56, 78, 80, 94, 105, 131]
Disruption	[42, 62, 77, 79, 101]
Data	[38, 75, 101]
throughput	[4, 76, 84]
Latency	[4, 76, 84]

and component model [70]. *RQ 3: What programming languages and algorithms employed in dynamic software updating?*

In this study, we have identified 30 programming languages cited in implementing DSU. Based on the number of studies for each programming language, we list the top five most cited programming languages in Table 12.

A number of explorations have demonstrated other programming languages including JavaScript [84, 100, 116], Assembly [47, 77, 80], ML [77, 101, 104], Ruby [4, 52], Python [52, 99], DSL [49, 105], HTML [100, 116], CafeOBJ [29, 104], EDDL [98], CLOS [37], ContextJS [38], Context Traits [38], Fortress [47], Modula [47], Pascal [104], Maude [104], CoreABS [5], Creol [5], stage [117], XML [14], AsmL [89], Lisp [56], Pharo [56], W [128], and PROMELA [112].

In this investigation, we have identified a list of 45 algorithms demonstrated in DSU, most illustrated algorithms in the selected primary studies are garbage collection [42, 44, 77], configuration management [58, 62], and distributed management [58, 62].

In addition, the result shows other algorithms including greedy [39], minimisation [107], breadth-first search [9], depth-first search [97], class-loading [43], monitor-based [9], controller synthesis [9], ReDAC (Dynamic Reconfiguration of Distributed Component-Based Applications with Cyclic Dependencies) [59], two-person game [9], encryption/decryption [9], thread-safe and lazy state migration [45], identifying update points [83], computing safe update points based on model checking [83], and transformation algorithm [4].

Furthermore, several investigations have shown more algorithms including distributed synchronisation [47], payload [110], Rsync [75], scheduling UCs.T [132], Deluge [75], genetic algorithms [91], BSDiff [75], distributed consistency [47], concurrent processes synchronisation [47], smith–Waterman [47], state copy [121], binary [114], image encryption [114], Xdelta [75], TOS matching [39], Max-flow-Min-cut [132], Two-Stage Diff for performing [75], OSTOS for update service pack [132], PROMELA [112], main synthesis [39], PID [65], Gulwani's [73], process-based validation [30], HotSwap-based validation [30], patch generation [80], reconfiguration [67], dynamic update of controllers [69], and control algorithm [65].

RQ 4: Which application domains have dynamic software updating been applied?

In this study, we have identified 35 DSU application domains. Based on the number of primary studies for each application domain, Table 13 shows the top ten most discussed application domains.

Moreover, number of studies have demonstrated further application domains including the Internet of Things [38, 54, 58, 60, 81, 82, 84, 106, 136], cloud computing [84, 88, 136], air traffic control systems [78–80, 123, 127, 134], component-based systems [53, 58–60, 63, 71], mobile applications [3, 38, 98], production plant [61, 120], text editors [39, 87], robot systems [64, 122], games and multimedia [6, 91], and financial transaction processors [80, 127].

Furthermore, the number of investigations have illustrated extra application domains including onboard software attitude orbit control [110], power plant controller system [63], high-performance computing [47], braking applications [133], data destitution centric applications [121], scientific applications [50], intelligent field devices [98], self-adaptive systems [91], safety-critical embedded systems [53], critical software-intensive systems [122], message-passing programs [118], transportation or space applications [122], dynamic vehicle software [66], speed up automatic program repair [30], and enhance security and privacy in high availability energy management applications in smart cities [106].

As shown in Fig. 8, the most cited application domain is internet-based applications representing 44.12%, followed by operating systems amounting to 10.78%. In addition, distributed systems and network applications, and databases with 8.82, 8.82, and 7.84% respectively. Moreover, resource-constrained systems, time-critical systems, and cyber-physical systems are less cited with 6.86, 6.86, and 5.85%, respectively.

RQ 5: What evaluation metrics, target programs, datasets, and open issues existing in dynamic software updating?

In this study, we have identified 21 evaluation metrics utilised in DSU. Table 14 shows the top six most demonstrated evaluation metrics in the selected primary studies.

For DSU time metrics, number of studies have presented various metrics including compile-time [41, 44, 46, 54, 96], load time [41, 45, 60, 133], waiting time [51, 73, 76, 83], reconfiguration time [59, 60, 62], rollback time [51, 101],

Table 15 Available programs in Defects4j benchmark [30]

Identifier	Project name	Number of bugs
chart	JFreeChart	26
lang	apache commons-lang	65
math	apache commons-math	106
time	Joda-Time	27

Table 16 Top nine most cited target programs

Target program	Primary studies
Vsftpd	[5, 6, 37, 74, 76, 77, 79, 90, 94, 95, 101–103, 105, 107, 128]
SSHD	[6, 36, 74, 76, 77, 79, 101, 102, 128]
Apache HTTP	[6, 14, 36, 73, 76, 94, 101, 102]
Memcached	[3, 6, 51, 90, 105]
Httpd	[14, 74, 79, 85, 101]
IceCast	[3, 6, 105, 128]
Openssh	[6, 37, 107, 127]
Redis	[6, 51, 90, 105]
Linux operating system	[72, 74, 78, 127]

Table 17 Top six most addressed DSU open issues

Open issue	Primary studies
time and safety	[35, 54, 62, 73, 116, 126, 133]
steady-state execution	[6, 41, 44, 45, 87]
synchronisation overhead	[5, 43, 83, 101, 116]
correctness	[29, 30, 69, 81, 82]
deadlock	[49, 52, 74, 104, 118]
timely and low disruptive	[38, 58, 62, 84]

Table 18 DSU research type facets

Type	Primary studies
validation research	[4, 5, 9, 16, 22, 30, 35, 36, 40, 43, 46, 50, 53, 54, 57, 64–66, 68, 70, 72, 74, 76–80, 85, 88, 89, 91, 94, 95, 97, 99, 100, 107, 110, 114, 116, 120, 123, 127, 131–134]
evaluation research	[3, 14, 37, 38, 41, 42, 44, 45, 49, 51, 52, 56, 67, 71, 81, 84, 96, 105, 106, 108, 111, 115, 136]
solution proposal	[39, 59–62, 69, 73, 75, 83, 86, 87, 90, 92, 93, 98, 101, 103, 109, 112, 117, 119, 121, 122, 125, 126, 128–130, 135]
philosophical papers	[29, 102, 104, 113, 118, 124]
opinion papers	[58]
experience papers	[6, 47, 48, 55, 63, 82]

quiescence time [3, 58], timeout [9, 111], debug time [60], disruption time [4], down-time [56], garbage collection time [44], interrupt time [38], synchronisation time [93], pausing time [42], restart time [60], and reconfiguration period [67], and warm-up time [56].

Moreover, number of authors have demonstrated other 15 evaluation metrics including availability metrics [6, 37], memory usage [41, 136], number of violations [77, 81], memory footprint [22, 77], energy cost [54, 106], update cost [63], clock cycles for DSU model [93], duration of time-division multiplexing slot in multiprocessor [133], measuring the impact changes on resource demands [41], patch size [14], number of errors [90], success-rate [35], simulation of two threads to measure the number of accepted and/or rejected update requests in different workloads [134], run the old version in a standard JVM (Java Virtual Machine) to record some performance metric as a baseline [87], and before/during/after the update measures [65].

Furthermore, in this study, we have identified one dataset demonstrated in DSU as a benchmark. Table 15 shows available programs in the Defects4j dataset benchmark [30].

In this examination, we have identified 56 target programs used as benchmarks in the DSU domain. Table 16 shows the top nine most cited target programs in the selected primary studies.

A number of authors have demonstrated other target programs including Tomcat [36, 42, 44, 87], CrossFTP [4, 39, 111], TinyOS [75, 92, 93], Jedit [39, 87, 110], Jetty web server [4, 96], H2 SQL database server [42, 111], DB-Derby [41, 96], RailCab [64, 69], Green Hills Integrity [41, 126], Tor [6, 105], Zebra [6, 77], Space Tyrant game [3, 6].

Moreover, other investigations have also shows more target programs containing Open Telecom Platform [120], ngIRCd [37], Air Traffic Control System [134], HyperSQL [52], MP3 [133], JPEG Decoder [133], Susan [133], Attitude and Orbit Control [110], Personal Data Resource Network [114], Azureus [39], java card [112], CORBA [59], JDNSS [96], PostGreSQL [6], WebLogic Server [16], Snake demo [16], proftpd [76], Wind River [126], Apache Empire-db [73], and QNX Neutrino [126].

Furthermore, extra examinations have also shows additional target programs including Siena [35], HSQLDB [46], Space Invaders [91], Lego Mindstorms robot's software [122], Over-Relaxation [50], JOnAS [5], Xerces [14], KissFFT [6], FASA [65], PaintDotNet [59], Zt-zip [57], iperf [3], PKUAS [113], Flash webserver [80], and suricate [3].

In this study, the selected primary studies have shown a list of 32 open issues in DSU. Based on the number of primary studies for each open issue, we list the top six most addressed DSU open issues in Table 17.

In addition, number of authors have recognised other open issues including correctness and timeliness [3, 64, 115], reliability [51, 66, 87, 90], time and memory space [4, 41, 108], effectiveness [30, 35, 76], safety timeliness and practicality [116, 128], safe and low-disruptive [58, 62], granularity [58, 62], locality [58, 62], and update of dependencies [56, 63].

Moreover, more studies have also illustrated other open issues including safe, efficient, flexible and low disruptive [87], availability and safety [37], correctness and safety [101], safety and immediate [116], immediate safe live update [51], safety and liveness [103], migration of instances [116], scope [54], less overhead in steady-state execution [4], automatically generate transfer functions [41], maintain up-to-date state in version [43], safety and liveness [118], flexibility and transparency [55], backward-compatibility [84], resilience [63], configurability [43], reliability and availability [90], and referential transparency [113].

RQ 6: What research type facets and contribution type facets focused on DSU studies?

In this investigation, the selected primary studies were classified according to the research type facets in Table 18 to highlight how the research type facets have been focused on DSU studies using the undertaken study.

As in Fig. 9, the most applied research type is validation research representing 41.96%, followed by solution proposals amounting to 26.79%. Also, evaluation research represents 19.64%, philosophical papers represent 5.36%, experience papers amounting to 4.46%, and opinion papers represent 1.79%, respectively.

To highlight the trend of DSU research type facets, Fig. 10 is a bubble plot chart shows DSU research type facets in the *X*-axis, in relation to the year of the publication in the *Y*-axis, the intersection bubble is the reference numbers of the primary studies.

As shown in Fig. 10, validation research, evolution research, and solution proposal are ongoing popular research type facets in the domain. Furthermore, the figure shows that experience papers, philosophical papers, and opinion papers are getting less attention in the area.

To map existing DSU research outcomes, the selected primary studies are classified according to their contribution type facets in Table 19.

As illustrated in Fig. 11, a significant number of selected primary studies contributed to the technique with 33.39%, followed by tool, framework, and theory with 11.61, 11.61, and 10.71%

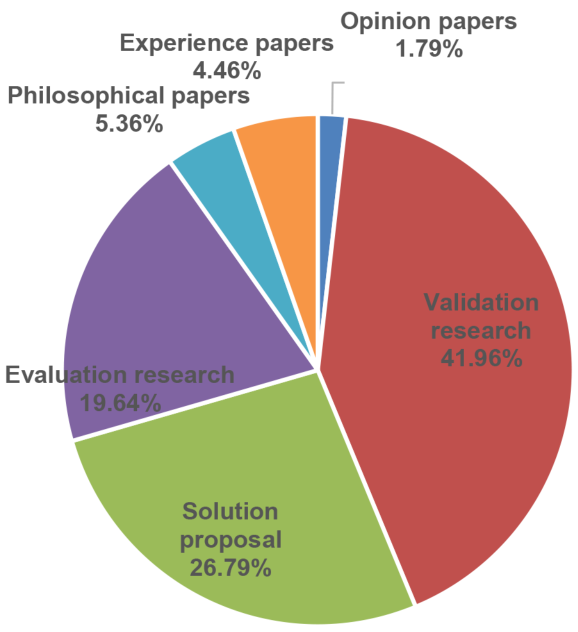


Fig. 9 *DSU research type facets*

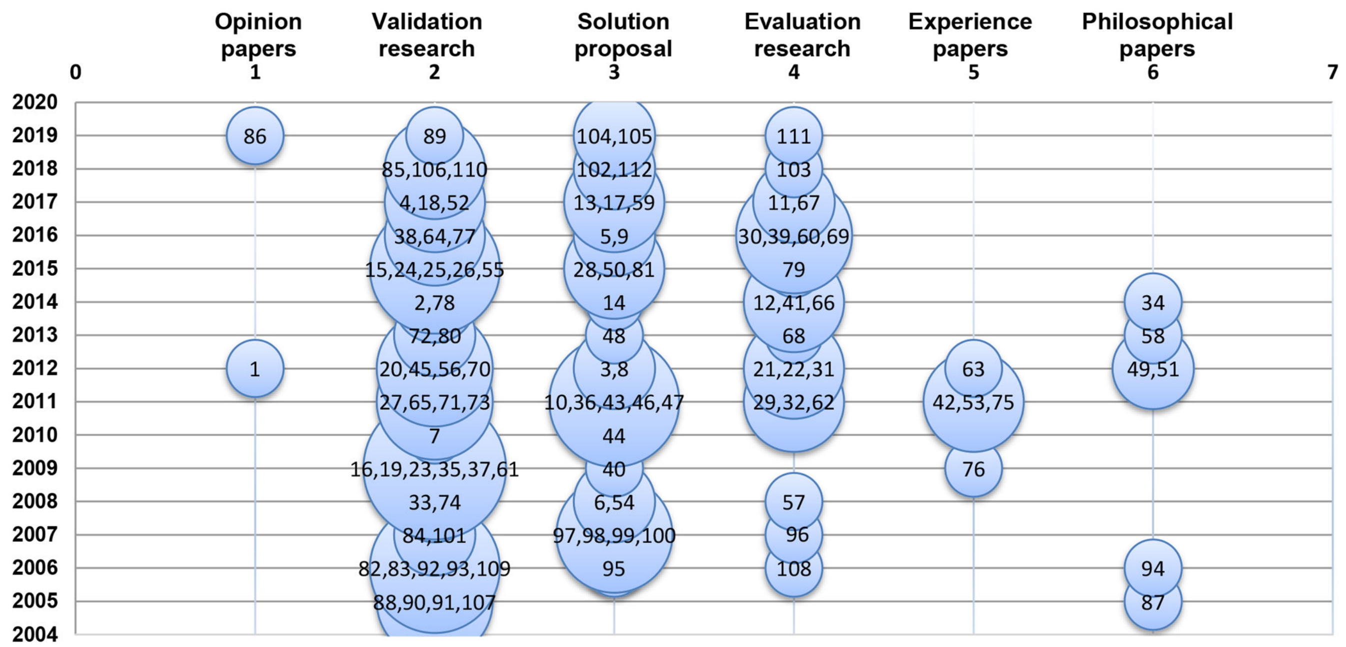


Fig. 10 *Bubble plot of DSU research type facets*

Table 19 DSU research contribution facets

Type	Primary studies
approaches	[56, 67, 80, 90]
models	[5, 22, 58, 64, 89, 96, 113, 117]
techniques	[4, 16, 35, 36, 38, 39, 43, 46, 47, 49, 50, 52–54, 57, 61, 62, 65, 73, 75, 84–86, 94, 97, 103, 109, 110, 112, 116, 120, 121, 123, 126–128, 133, 134]
frameworks	[9, 40, 60, 68, 70, 76, 95, 98, 129, 131, 132, 135, 136]
algorithm	[59, 130]
prototype	[30, 72, 88, 92, 93, 122]
theory	[29, 69, 71, 81, 83, 99, 100, 104, 115, 118, 119, 124]
tool	[42, 44, 51, 74, 77–79, 82, 87, 91, 101, 105, 108]
guideline	[48, 125]
lessons learnt	[45, 63]
advice	[6, 55, 66]
architecture	[106, 114]
metric	[3, 14, 37, 41, 102, 107, 111]

respectively. On another hand, algorithms, guidelines, lessons learnt, and architecture scored the lowest percentage with 1.79% each.

To highlight the trend of DSU research and contribution type facets, Fig. 12 is a bubble plot diagram shows DSU contribution type facets in the *X*-axis, in relation to the year of the publication in the *Y*-axis, the intersection bubble is the reference number. As demonstrated in Fig. 12, the technique is the most popular contribution type facets employed in the area. Moreover, the figure shows that architecture, algorithms, lesson learnt, guidelines, and advice are getting less attention in the area.

5 Principal findings

In this investigation, six RQs were announced, an SMS on DSU was conducted, five electronic databases were utilised, 112 primary studies were selected, and the answers to the RQs were reported. In this section, we declare the main finding (s):

- Based on the identified DSU approaches, our results demonstrated that the customised JVM, dynamic

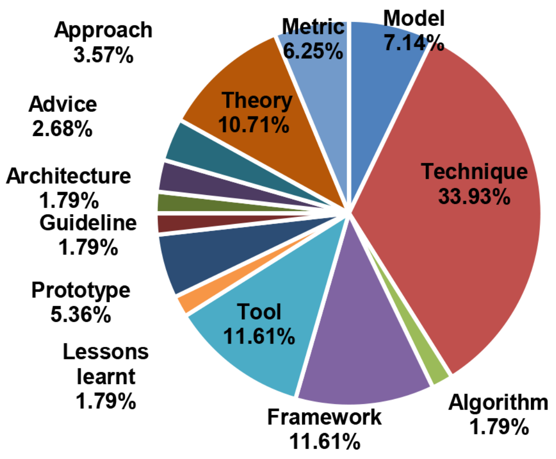


Fig. 11 *DSU contribution type facets*

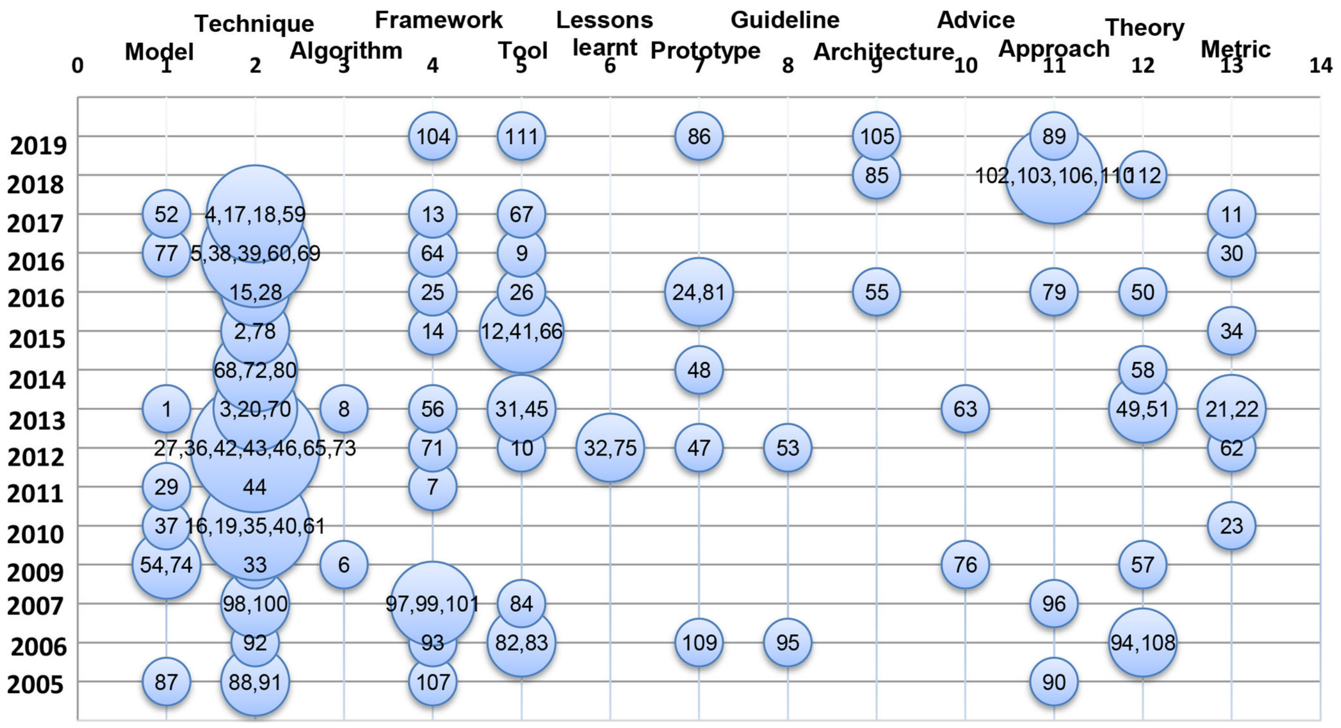


Fig. 12 *Bubble plot of DSU contribution type facets*

reconfiguration, dynamic Patch, Hot-swapping, multi-version programming, and customised class loaders are most cited in the domain. Also, our results highlighted trend values to argue that multi-version programming, dynamic patch, and dynamic reconfiguration are ongoing most popular in the area.

- It is interesting to note that, the current investigation adds to a growing body of the literature an aggregation of 55 tools (DSU systems). Based on a number of primary studies for each tool, this analysis found evidence that most cited DSU tools are POLUS, Ginseng, UpStare, Jvolve, and Kitsune.
- This investigation has gone some way towards enhancing our understanding of DSU techniques, in specific, this study found that widespread attention was paid to research on DSU techniques; however, a list of 195 of DSU techniques facets were identified. Based on the results, it is clear that DSU safety is the most investigated facets followed by state transformation facets. On another hand, runtime and update points facets received less attention.
- Our results cast a new light on existing models for DSU including 27 models. It is interesting to note that the most cited models are interrupt, relax consistency, invoke, indirection, and formal models.
- For programming languages and algorithms employed in DSU, the results of this study show 30 programming languages and 46 algorithms employed in DSU. Together, the present findings confirm most used programming languages in DSU are Java, C/C++, Erlang, C#, and Smalltalk, Also, the results at least hint garbage collection, configuration management, and distributed management are most employed algorithms in the area.
- This study discovered that significant consideration was offered to DSU in 35 application domains. Accordingly, it is important to highlight the fact that DSU is becoming a focal point to many application domains. From the results, it is clear that the most discussed application domain in DSU is internet-based applications that appeared in 46 of the selected primary studies.
- It is extraordinary to note that this investigation presented a list of 21 evaluation metrics, one dataset, and 56 target programs considered in DSU research domain. The result shows most used evaluation metrics include update time, overhead, disruption, data, throughput and latency. Also, the result highlights most used target programs contain Vsftpd, SSHD, Apache HTTP, Memcached, and Httpd. Overall, our findings on open issues, evaluation metrics, datasets, and target programs at least provide a hint to the new researchers in the domain.
- It is remarkable to note that this paper highlighted a list of 32 open issues in DSU. Our results show most discussed open issues in DSU includes time and safety, steady-state execution,

synchronisation overhead, correctness, deadlock, and timely and low disruptive.
- Our results cast a new light on research type facets and contribution type facets in DSU. The results demonstrate two things. First, validation research, solution proposals, and evaluation research are the most utilised research type facets in DSU. Secondly, the technique is dominating contribution type facets in the domain.

6 Threats to validity

A number of authors have recognised particular threats that may undermine the validity of SMS [137, 138]. It includes conclusion, internal, constructs, and external validity [137, 138]. In the following parts, we clarify threats to the validity of this study.

6.1 Conclusion validity

Conclusion validity thread is occurring when we had biases when we analysed and categorised the retrieved studies. In this study, we are not aware of biases we may have had when we analysed and categorised the retrieved studies. In particular, the reader should be aware of the impact of our occupied interests on the studies. In short, to mitigate this threat, we utilised the following:

- A well-defined IC, EC [17, 19] was utilised in this investigation.
- For the conflicting evaluations, during the data collection, our research group operated as data exacter, reviser, and final corrector to resolve the conflicts.

Another threat is that we defined the search strings based on our experience. To mitigate this threat, our search string was defined based on the RQs following PICO criteria [18].

6.2 Internal validity

Internal validity thread refers to the probability that some studies are not selected. To mitigate this threat, in this study, a total of 1066 studies on DSU were accounted which we consider is not a limited number of studies. Therefore, it minimises the risk of unreturned papers during the searching process.

6.3 Construct validity

Construct validity thread refers to the exclusion of relevant papers presenting DSU. To mitigate this threat, this study followed a rigorous selection process [18–20, 26–28] as shown in Fig. 3, and IC, EC [17, 19] as presented in Table 2.

6.4 External validity

External validity thread is when we take into consideration or analyse the results of the previous SMS in the domain. For this study, our research group found no SMS published in DSU.

7 Concluding remarks and future work

DSU is a process of modifying a running software system without a halt [4–6, 11]. A closer look at the literature on DSU, in recent years, there exists a considerable number of studies that have been conducted on DSU [2, 3, 8, 36]. In addition, this study aggregated and reviewed existing knowledge in DSU studies published over 15 years.

In this study, an SMS [17–20] was conducted with a set of six RQs and 1066 papers published from 2005 to 2019 were recorded. After a filtering process, 112 primary studies were selected and inspected.

In summary, this paper highlights the current state-of-the-art of DSU including approaches, tools, models, and techniques. Also, this paper outlines application domains, research type, and contributions type facets of DSU. In addition, this paper demonstrates benchmarks, datasets, evaluation metrics, programming languages, algorithms, and existing open issues for future research in DSU.

Based on the trends and open issues in the results of this study, we believe there are ample rooms for more investigations in DSU. Moreover, it would be interesting to demonstrate the outcome of this investigation to assist existing studies, future research, and practice on DSU. Furthermore, a future study inspecting DSU in an SLR would be very interesting.

8 References

[1] Miedes, E., Muñoz-Escoí, F.D.: 'A survey about dynamic software updating'. Instituto Universitario Mixto Tecnologico de Informatica, Universitat Politecnica de Valencia, Campus de Vera s/n, 46022, 2012

[2] Nettles, S.M.: 'Dynamic software updating michael hicks'. Doctoral dissertation, University of Pennsylvania, 2001

[3] Hayden, C.M., Saur, K., Hicks, M., *et al.*: 'A study of dynamic software update quiescence for multithreaded programs'. 2012 Fourth Workshop on Hot Topics in Software Upgrades (HotSWUp), Zurich, June 2012, pp. 6–10

[4] Subramanian, S., Hicks, M., McKinley, K.S.: 'Dynamic software updates: a VM-centric approach', In Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation, Dublin, Ireland, 2009, pp. 1–12

[5] Bravetti, M., Giachino, E., Lienhardt, M., *et al.*: 'Dynamic rebinding for concurrent object groups: theory and practice', *J. Logical Algebraic Methods Program.*, 2017, **86**, (1), pp. 349–390

[6] Smith, E.K., Hicks, M., Foster, J.S.: 'Towards standardized benchmarks for dynamic Software updating systems'. 2012 Fourth Workshop on Hot Topics in Software Upgrades (HotSWUp), Zurich, June 2012, pp. 11–15

[7] Neamtiu, I., Foster, J.S., Hicks, M.: 'Understanding source code evolution using abstract syntax tree matching', *ACM SIGSOFT Softw. Eng. Notes*, 2005, **30**, (4), pp. 1–5

[8] Neamtiu, I., Stoyle, G., Oriol, M., *et al.*: 'Toward practical dynamic software updating', 2005

[9] An, S., Ma, X., Cao, C., *et al.*: 'An event-based formal framework for a dynamic Software update'. 2015 IEEE Int. Conf. on Software Quality, Reliability and Security (QRS), University of Maryland, USA, August 2015, pp. 173–182

[10] Ilvonen, V., Ihantola, P., Mikkonen, T.: 'Dynamic software updating techniques in practice and educator's guides: a review'. 2016 IEEE 29th Int. Conf. on Software Engineering Education and Training (CSEET), Dallas, TX, USA, April 2016, pp. 86–90

[11] Seifzadeh, H., Abolhassani, H., Ashkenazi, M.S.: 'A survey of dynamic software updating', *J. Softw., Evol. Process*, 2013, **25**, (5), pp. 535–568

[12] Gregersen, A.R., Rasmussen, M., Jørgensen, B.N.: 'State of the art of dynamic software updating in Java'. Int. Conf. on Software Technologies, Berlin, Heidelberg, July 2013, pp. 99–113

[13] Mugarza, I., Parra, J., Jacob, E.: 'Analysis of existing dynamic software updating techniques for safe and secure industrial control systems', *Int. J. Safety Secur. Eng.*, 2018, **8**, (1), pp. 121–131

[14] Gharaibeh, B., Rajan, H., Chang, J.M.: 'Analyzing software updates: should you build a dynamic updating infrastructure?'. Int. Conf. on Fundamental Approaches to Software Engineering, Berlin, Heidelberg, March 2011, pp. 371–385

[15] Lee, I.: '*Dymos: a dynamic modification system*' (University of Wisconsin-Madison Department of Computer Sciences, United States, 1983)

[16] Pukall, M., Grebhahn, A., Schröter, R., *et al.*: 'Javadaptor: unrestricted dynamic software updates for Java'. Proc. of the 33rd Int. Conf. on Software Engineering, Waikiki, Honolulu HI USA, May 2011, pp. 989–991

[17] Petersen, K., Feldt, R., Mujtaba, S., *et al.*: 'Systematic mapping studies in software engineering'. EASE, University of Bari, Italy, June 2008, vol. 8, pp. 68–77

[18] Kitchenham, B., Charters, S.: 'Guidelines for performing systematic literature reviews in software engineering'. Keele University and Durham University Joint Report, Technical Report, EBSE 2007-001, 2007

[19] Petersen, K., Vakkalanka, S., Kuzniarz, L.: 'Guidelines for conducting systematic mapping studies in software engineering: an update', *Inf. Softw. Technol.*, 2015, **64**, pp. 1–18

[20] Bernardino, M., Rodrigues, E.M., Zorzo, A.F., *et al.*: 'Systematic mapping study on MBT: tools and models', *IET Softw.*, 2017, **11**, (4), pp. 141–155

[21] Kosar, T., Bohra, S., Mernik, M.: 'Domain-specific languages: a systematic mapping study', *Inf. Softw. Technol.*, 2016, **71**, pp. 77–91

[22] Jalili, M., Parsa, S., Seifzadeh, H.: 'A hybrid model in dynamic software updating for c'. Int. Conf. on Advanced Software Engineering and Its Applications, Berlin, Heidelberg, December 2009, pp. 151–159

[23] Kitchenham, B.: 'What's up with software metrics? –A preliminary mapping study', *J. Syst. Softw.*, 2010, **83**, (1), pp. 37–51

[24] Budgen, D., Turner, M., Brereton, P., *et al.*: 'Using mapping studies in software engineering'. Proc. of PPIG, Lancaster, UK, September 2008, vol. 8, pp. 195–204

[25] Fernandez, A., Insfran, E., Abrahão, S.: 'Usability evaluation methods for the web: a systematic mapping study', *Inf. Softw. Technol.*, 2011, **53**, (8), pp. 789–817

[26] Dyba, T., Dingsoyr, T., Hanssen, G.K.: 'Applying systematic reviews to diverse study types: an experience report'. First Int. Symp. on Empirical Software Engineering and Measurement, 2007. ESEM 2007, Madrid, Spain, September 2007, pp. 225–234

[27] Kitchenham, B., Brereton, P.: 'A systematic review of systematic review process research in software engineering', *Inf. Softw. Technol.*, 2013, **55**, (12), pp. 2049–2075

[28] Anderson, L.M., Petticrew, M., Rehfuess, E.*, et al.*: 'Using logic models to capture complexity in systematic reviews', *Res. Synth. Methods.*, 2011, **2**, (1), pp. 33–42

[29] Zhang, M., Ogata, K., Futatsugi, K.: 'Formalization and verification of behavioral correctness of dynamic software updates', *Electron. Notes Theor. Comput. Sci.*, 2013, **294**, pp. 12–23

[30] Guo, R., Gu, T., Yao, Y.*, et al.*: 'Speedup automatic program repair using dynamic software updating: an empirical study'. Proc. of the 11th Asia-Pacific Symp. on Internetware, Fukuoka Japan, October 2019, p. 14

[31] Brereton, P., Kitchenham, B.A., Budgen, D.*, et al.*: 'Lessons from applying the systematic literature review process within the software engineering domain', *J. Syst. Softw.*, 2007, **80**, (4), pp. 571–583

[32] O'Donovan, P., Leahy, K., Bruton, K.*, et al.*: 'Big data in manufacturing: a systematic mapping study', *J. Big. Data.*, 2015, **2**, (1), p. 20

[33] Sungur, M.O., Seyhan, T.Ö.: 'Writing references and using citation management software', *Turk. J. Urol.*, 2013, **39**, (Suppl 1), p. 25

[34] Robertson, G., Fernandez, R., Fisher, D.*, et al.*: 'The effectiveness of animation in trend visualization', *IEEE Trans. Vis. Comput. Graphics*, 2008, **14**, (6), pp. 1325–1332

[35] Zhao, Z., Ma, X., Xu, C.*, et al.*: 'Automated recommendation of dynamic software update points: an exploratory study'. Proc. of the 6th Asia-Pacific Symp. on Internetware on Internetware, Hong Kong China, November 2014, pp. 136–144

[36] Gu, T., Ma, X., Xu, C.*, et al.*: 'Synthesizing object transformation for dynamic software updating'. Proc. of the 39th Int. Conf. on Software Engineering Companion, Buenos Aires Argentina, May 2017, pp. 336–338

[37] Hayden, C.M., Smith, E.K., Hardisty, E.A.*, et al.*: 'Evaluating dynamic software update safety using systematic testing', *IEEE Trans. Softw. Eng.*, 2012, **38**, (6), pp. 1340–1354

[38] Weißbach, M., Taing, N., Wutzler, M.*, et al.*: 'Decentralized coordination of dynamic software updates in the Internet of Things'. 2016 IEEE 3rd World Forum on the Internet of Things (WF-IoT), Reston, VA, USA, December 2016, pp. 171–176

[39] Magill, S., Hicks, M., Subramanian, S.*, et al.*: 'Automating object transformations for dynamic software updating', *ACM SIGPLAN Notices*, 2012, **47**, (10), pp. 265–280

[40] Banno, F., Marletta, D., Pappalardo, G.*, et al.*: 'Handling consistent dynamic updates on distributed systems'. 2010 IEEE Symp. on Computers and Communications (ISCC), Riccione, Italy, June 2010, pp. 471–476

[41] Mlinarić, D., Mornar, V.: 'Dynamic software updating in Java: comparing concepts and resource demands'. Companion to the first Int. Conf. on the Art, Science and Engineering of Programming, Brussels Belgium, April 2017, p. 12

[42] Gu, T., Cao, C., Xu, C.*, et al.*: 'Low-disruptive dynamic updating of Java applications', *Inf. Softw. Technol.*, 2014, **56**, (9), pp. 1086–1098

[43] Gregersen, A.R., Jørgensen, B.N.: 'Dynamic update of Java applications – balancing change flexibility vs programming transparency', *J. Softw., Evol. Process*, 2009, **21**, (2), pp. 81–112

[44] Gu, T., Cao, C., Xu, C.*, et al.*: 'Javelus: A low disruptive approach to dynamic software updates'. 2012 19th Asia-Pacific Software Engineering Conf. (APSEC), Hong Kong, China, December 2012, vol. 1, pp. 527–536

[45] Gregersen, A.R.: 'Implications of modular systems on dynamic updating'. Proc. of the 14th Int. ACM Sigsoft Symp. on Component based Software engineering, Marcq-en-Bareul France, June 2011, pp. 169–178

[46] Cazzola, W., Jalili, M.: 'Dodging unsafe update points in java dynamic software updating systems'. 2016 IEEE 27th Int. Symp. on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, October 2016, pp. 332–341

[47] Kim, D.K., Tilevich, E., Ribbens, C.J.: 'Dynamic software updates for parallel high-performance applications', *Concurrency Comput., Pract. Exp.*, 2011, **23**, (4), pp. 415–434

[48] Gregersen, A.R., Jørgensen, B.N.: 'Run-time phenomena in dynamic software updating: causes and effects'. Proc. of the 12th Int. Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, Szeged Hungary, September 2011, pp. 6–15

[49] Neumann, M.A.: 'Towards practical release-level dynamic Software updating on stock Java: evaluating an efficient and safely programmable Java dynamic updating system'. Companion Proc. of the 2016 ACM SIGPLAN Int. Conf. on Systems, Programming, Languages, and Applications: Software for Humanity, Amsterdam Netherlands, October 2016, pp. 24–26

[50] Kim, D.K., Song, M., Tilevich, E.*, et al.*: 'Dynamic software updates for accelerating scientific discovery'. Int. Conf. on Computational Science, Berlin, Heidelberg, May 2009, pp. 237–247

[51] Chen, Z., Qiang, W.: 'ISLUS: an immediate and safe live update system for C program'. 2017 IEEE Second Int. Conf. on Data Science in Cyberspace (DSC), Shenzhen, China, June 2017, pp. 267–274

[52] Pukall, M., Kästner, C., Cazzola, W.*, et al.*: 'Javadaptor – flexible runtime updates of Java applications', *Softw., Pract. Exp.*, 2013, **43**, (2), pp. 153–185

[53] Kajtazovic, N., Preschern, C., Kreiner, C.: 'A component-based dynamic link support for safety-critical embedded systems'. 2013 20th IEEE Int. Conf. and Workshops on the Engineering of Computer Based Systems (ECBS), NW Washington, DC, United States, April 2013, pp. 92–99

[54] Porter, B., Roedig, U., Coulson, G.: 'Type-safe updating for modular WSN software'. 2011 Int. Conf. on Distributed Computing in Sensor Systems and Workshops (DCOSS), Barcelona, Spain, June 2011, pp. 1–8

[55] Gregersen, A.R., Simon, D., Jørgensen, B.N.: 'Towards a dynamic-update-enabled JVM'. Proc. of the Workshop on AOP and Meta-Data for Software Evolution, Genova Italy, July 2009, p. 2

[56] Polito, G., Ducasse, S., Bouraqadi, N.*, et al.*: 'Virtualization support for dynamic core library update'. 2015 ACM Int. Symp. on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!), Pittsburgh PA USA, October 2015, pp. 211–223

[57] Šelajev, O., Raudjärv, R., Kabanov, J.: 'Static analysis for dynamic updates'. Proc. of the 9th Central & Eastern European Software Engineering Conf. in Russia, Russia, October 2013, p. 7

[58] Panzica La Manna, V.: 'Local dynamic update for component-based distributed systems'. Proc. of the 15th ACM SIGSOFT Symp. on Component-Based Software Engineering, Bertinoro Italy, June 2012, pp. 167–176

[59] Rasche, A., Polze, A.: 'ReDAC – dynamic reconfiguration of distributed component-based applications with cyclic dependencies'. 2008 11th IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC), Orlando, FL, May 2008, pp. 322–330

[60] Boyer, F., Gruber, O., Pous, D.: 'A robust reconfiguration protocol for the dynamic update of component-based Software systems', *Softw., Pract. Exp.*, 2017, **47**, (11), pp. 1729–1753

[61] Roder, S.Q., Provost, J.: 'Dynamic software update of stateflow charts using erlang runtime system', *IFAC-PapersOnLine*, 2017, **50**, (1), pp. 5855–5860

[62] Panzica La Manna, V.: 'Dynamic Software update for component-based distributed systems'. Proc. of the 16th int. workshop on Component-oriented programming, Boulder Colorado, USA, June 2011, pp. 1–8

[63] Gama, K., Rudametkin, W., Donsez, D.: 'Resilience in dynamic component-based applications'. 2012 26th Brazilian Symp. on Software Engineering (SBES), Natal, Brazil Brazil, September 2012, pp. 191–195

[64] Nahabedian, L., Braberman, V., D'Ippolito, N.*, et al.*: 'Assured and correct dynamic update of controllers'. 2016 IEEE/ACM 11th Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Austin Texas, May 2016, pp. 96–107

[65] Wahler, M., Oriol, M.: 'Disruption-free Software updates in automation systems'. 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, Spain, September 2014, pp. 1–8

[66] Jakobs, C., Tröger, P., Werner, M.*, et al.*: 'Dynamic vehicle software with AUTOCONT'. 2018 55th ACM/ESDA/IEEE Design Automation Conf. (DAC), San Francisco, CA, USA, June 2018, pp. 1–6

[67] Zhao, Z., Li, W.: 'Influence control for dynamic reconfiguration'. 2007 Australian Software Engineering Conf. (ASWEC'07), Melbourne, Australia, April 2007, pp. 59–70

[68] Rasche, A., Schult, W.: 'Dynamic updates of graphical components in the. Net framework'. Communication in Distributed Systems-15. ITG/GI Symp., VDE Verlag, 2007, pp. 1–12

[69] Nahabedian, L., Braberman, V., D'lppolito, N.*, et al.*: 'Dynamic update of discrete event controllers', *IEEE Trans. Softw. Eng.*, 2018, p. 1, doi: 10.1109/TSE.2018.2876843

[70] Lee, Y.F., Chang, R.C.: 'Java-based component framework for dynamic reconfiguration', *IEE Proc., Softw.*, 2005, **152**, (3), pp. 110–118

[71] Niamanesh, M., Nobakht, N.F., Jalili, R.*, et al.*: 'On validity assurance of dynamic reconfiguration for component-based programs', *Electron. Notes Theor. Comput. Sci.*, 2006, **159**, pp. 227–239

[72] Lee, Y.F., Chang, R.C.: 'Hot swapping Linux kernel modules', *J. Syst. Softw.*, 2006, **79**, (2), pp. 163–175

[73] Zhao, Z., Gu, T., Ma, X.*, et al.*: 'Cure: automated patch generation for a dynamic Software update'. 2016 23rd Asia-Pacific Software Engineering Conf. (APSEC), Hamilton, New Zealand, December 2016, pp. 249–256

[74] Zou, D., Wang, H., Jin, H.: 'Strongupdate: an immediate dynamic software update system for multi-threaded applications'. Int. Conf. on Human-Centered Computing, Phnom Penh, Cambodia, November 2014, pp. 365–379

[75] Chiang, M.L., Lu, T.L.: 'Two-stage diff: an efficient dynamic software update mechanism for wireless sensor networks'. 2011 IFIP 9th Int. Conf. on Embedded and Ubiquitous Computing (EUC), Melbourne, Australia, October 2011, pp. 294–299

[76] Chen, G., Jin, H., Zou, D.*, et al.*: 'A framework for practical dynamic Software updating', *IEEE Trans. Parallel Distrib. Syst.*, 2016, **27**, (4), pp. 941–950

[77] Neamtiu, I., Hicks, M., Stoyle, G.*, et al.*: 'Practical dynamic software updating for' *C. ACM SIGPLAN Notices*, 2006, **41**, (6), pp. 72–83

[78] Makris, K., Ryu, K.D.: 'On-the-fly kernel updates for high-performance computing clusters'. Proc. 20th IEEE Int. Parallel & Distributed Processing Symp., Rhodes Island, Greece, April 2006, p. 8

[79] Chen, H., Yu, J., Chen, R.*, et al.*: 'Polus: A powerful live updating system'. 29th Int. Conf. on Software Engineering (ICSE'07), Minneapolis, MN, USA, May 2007, pp. 271–281

[80] Hicks, M., Nettles, S.: 'Dynamic software updating', *ACM Trans. Program. Languages Syst. (TOPLAS)*, 2005, **27**, (6), pp. 1049–1096

[81] Qian, J.: 'Toward a unified operational semantics-based approach to modeling and verifying dynamic software updating'. 2018 IEEE 9th Int. Conf. on Software Engineering and Service Science (ICSESS), Beijing, China, China, November 2018, pp. 1–4

[82] Qian, J., Zhang, M., Wang, Y.*, et al.*: 'Kupc: a formal tool for modeling and verifying dynamic updating of C programs'. Int. Conf. on Fundamental Approaches to Software Engineering, Prague, Czech Republic, April 2019, pp. 299–305

[83] Zhang, M., Ogata, K., Futatsugi, K.: 'Towards a formal approach to modeling and verifying the design of dynamic software updates'. 2015 Asia-Pacific Software Engineering Conf. (APSEC), New Delhi, India, December 2015, pp. 159–166

[84] Neumann, M.A., Bach, C.T., Miclaus, A.*, et al.*: 'Always-on web of things infrastructure using dynamic software updating'. Proc. of the Seventh Int. Workshop on the Web of Things, Stuttgart Germany, November 2016, pp. 5–10

[85] Alhazbi, S., Jantan, A.: 'Multi-level mediator-based technique for classes hot swapping in java applications'. 2006 2nd Int. Conf. on Information & Communication Technologies, Damascus, Syria, April 2006, vol. 2, pp. 2889–2892

[86] Ye, H., Wang, H., Liang, Y.: 'Research on mission hot-swapping and dynamic replacement mechanism of mission component'. 2007 Int. Conf. on

Computational Intelligence and Security Workshops (CISW 2007), Heilongjiang, China, December 2007, pp. 890–893

[87] Gu, T., Zhao, Z., Ma, X., *et al.*: 'Improving the reliability of dynamic software updating using runtime recovery'. 2016 23rd Asia-Pacific Software Engineering Conf. (APSEC), Hamilton, New Zealand, December 2016, pp. 257–264

[88] Chen, F., Qiang, W., Jin, H., *et al.*: 'Multi-version execution for the dynamic updating of cloud applications'. 2015 IEEE 39th Annual Computer Software and Applications Conf. (COMPSAC), Taichung, Taiwan, July 2015, vol. 2, pp. 185–190

[89] Wu, J., Huang, L., Wang, D.: 'ASM-based model of dynamic service update in OSGi', *ACM SIGSOFT Softw. Eng. Notes*, 2008, **33**, (2), p. 8

[90] Pina, L., Andronidis, A., Hicks, M., *et al.*: 'MVEDSUA: higher availability dynamic software updates via multi-version execution'. Proc. of the Twenty-Fourth Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Providence RI USA, April 2019, pp. 573–585

[91] Gregersen, A.R., Jørgensen, B.N., Koskimies, K.: 'Javeleon: an integrated platform for dynamic software updating and its application in self-* systems'. 2012 Spring Congress on Engineering and Technology (S-CET), Xian, China, May 2012, pp. 1–9

[92] Chi, T.Y., Wang, W.C., Kuo, S.Y.: 'Uflow: dynamic software updating in wireless sensor networks'. Int. Conf. on Ubiquitous Intelligence and Computing, Berlin, Heidelberg, September 2011, pp. 405–419

[93] Chang, Y.C., Chi, T.Y., Wang, W.C., *et al.*: 'Dynamic software update model for remote entity management of machine-to-machine service capability', *IET Commun.*, 2013, **7**, (1), pp. 32–39

[94] Bazzi, R.A., Topp, B., Neamtiu, I.: 'How to have your cake and eat it too: dynamic Software updating with just-in-time overhead'. 2012 Fourth Workshop on Hot Topics in Software Upgrades (HotSWUp), Zurich, June 2012, pp. 1–5

[95] Hayden, C.M., Magill, S., Hicks, M., *et al.*: 'Specifying and verifying the correctness of dynamic software updates'. Int. Conf. on Verified Software: Tools, Theories, Experiments, Berlin, Heidelberg, January 2012, pp. 278–293

[96] Cech Previtali, S., Gross, T.R.: 'Aspect-based dynamic Software updating: a model and its empirical evaluation'. Proc. of the tenth Int. Conf. on Aspect-oriented Software development, Porto de Galinhas Brazil, March 2011, pp. 105–116

[97] Zhenxing, Y., Zhixiang, Z., Kerong, B.: 'An approach to dynamic software updating for java'. Pacific-Asia Workshop on Computational Intelligence and Industrial Application, 2008. PACIIA'08, Wuhan, China, December 2008, vol. 2, pp. 930–934

[98] Schmitt, M., Loskyll, M., Zuehlke, D.: 'Development of a framework for dynamic function deployment and extension by using apps on intelligent field devices', *IFAC Proc. Vol.*, 2014, **47**, (3), pp. 2611–2616

[99] Tang, W., Zhang, M.: 'Pyreload: dynamic updating of python programs by reloading'. 2018 25th Asia-Pacific Software Engineering Conf. (APSEC), Nara, Japan, December 2018, pp. 229–238

[100] Cazzola, W., Chitchyan, R., Rashid, A., *et al.*: 'μ-DSU: a micro-language based approach to dynamic software updating', *Comput. Lang., Syst. Struct.*, 2018, **51**, pp. 71–89

[101] Chen, H., Yu, J., Hang, C., *et al.*: 'Dynamic Software updating using a relaxed consistency model', *IEEE Trans. Softw. Eng.*, 2011, **37**, (5), pp. 679–694

[102] Zhang, M., Ogata, K., Futatsugi, K.: 'Verifying the design of dynamic software updating in the OTS/CafeOBJ method'. Specification, Algebra, and Software, Berlin, Heidelberg, 2014, pp. 560–577

[103] Kim, D.K., Kim, W.T., Park, S.M.: 'DSUENHANCER: a dynamic update system for resource-constrained software'. Control and Automation, and Energy System Engineering, Berlin, Heidelberg, 2011, pp. 195–201

[104] Zhang, M., Ogata, K., Futatsugi, K.: 'An algebraic approach to the formal analysis of dynamic software updating mechanisms'. 2012 19th Asia-Pacific Software Engineering Conf. (APSEC), Hong Kong, China, December 2012, vol. 1, pp. 664–673

[105] Hayden, C.M., Smith, E.K., Denchev, M., *et al.*: 'Kitsune: efficient, general-purpose dynamic software updating for C', *ACM SIGPLAN Notices*, 2012, **47**, (10), pp. 249–264

[106] Mugarza, I., Amurrio, A., Azketa, E., *et al.*: 'Dynamic software updates to enhance security and privacy in high availability energy management applications in smart cities', *IEEE Access*, 2019, **7**, pp. 42269–42279

[107] Hayden, C.M., Hardisty, E.A., Hicks, M., *et al.*: 'Efficient systematic testing for dynamically updatable software'. Proc. of the 2nd Int. Workshop on Hot Topics in Software Upgrades, Orlando Florida, October 2009, p. 9

[108] Kang, S., Chun, I., Kim, W.: 'Dynamic software updating for cyber-physical systems'. The 18th IEEE Int. Symp. on Consumer Electronics (ISCE 2014), JeJu Island, South Korea, June 2014, pp. 1–3

[109] Zhang, S., Huang, L.: 'Type-safe dynamic update transaction'. 31st Annual Int. Computer Software and Applications Conf. (COMPSAC 2007), Beijing, China, July 2007, vol. 2, pp. 335–340

[110] Lv, W., Zuo, X., Wang, L.: 'Dynamic software updating for onboard software'. 2012 Second Int. Conf. on Intelligent System Design and Engineering Application (ISDEA), Sanya, Hainan, China, January 2012, pp. 251–253

[111] Pina, L., Hicks, M.: 'Tedsuto: A general framework for testing dynamic software updates'. 2016 IEEE Int. Conf. on Software Testing, Verification and Validation (ICST), Chicago, IL, USA, April 2016, pp. 278–287

[112] Lounas, R., Jafri, N., Legay, A., *et al.*: 'A formal verification of safe update point detection in dynamic software updating'. Int. Conf. on Risks and Security of Internet and Systems, Roscoff, France, September 2016, pp. 31–45

[113] Shen, J., Sun, X., Huang, G., *et al.*: 'Towards a unified formal model for supporting mechanisms of dynamic component update', *ACM SIGSOFT Softw. Eng. Notes*, 2005, **30**, (5), pp. 80–89

[114] Liu, J., Mao, X.: 'Towards realization of evolvable runtime variability in internet-based service systems via dynamical software update'. Proc. of the 6th Asia-Pacific Symp. on Internetware on Internetware, Hong Kong, China, November 2014, pp. 97–106

[115] Nesmith, I., Hicks, M., Foster, J.S., *et al.*: 'Contextual effects for version-consistent dynamic software updating and safe concurrent programming', *ACM SIGPLAN Notices*, 2008, **43**, (1), pp. 37–49

[116] Wernli, E., Gurtner, D., Nierstrasz, O.: 'Using first-class contexts to realize dynamic software updates'. Proc. of the Int. Workshop on Smalltalk Technologies, Prague Czech Republic, August 2011, p. 2

[117] Duquesne, P., Bryce, C.: 'A language model for dynamic code updating'. Proc. of the 1st Int. Workshop on Hot Topics in Software Upgrades, Nashville Tennessee, October 2008, p. 2

[118] Anderson, G., Rathke, J.: 'Dynamic software update for message passing programs'. the Asian Symp. on Programming Languages and Systems, Berlin, Heidelberg, December 2012, pp. 207–222

[119] Johnsen, E.B., Yu, I.C.: 'Dynamic software updates and context adaptation for distributed active objects', in '*Principled software development*' (Springer, Cham, 2018), pp. 147–164

[120] Prenzel, L., Provost, J.: 'Dynamic software updating of IEC 61499 implementation using Erlang runtime system', *IFAC-PapersOnLine*, 2017, **50**, (1), pp. 12416–12421

[121] Park, M.J., Kim, D.K., Kim, W.T., *et al.*: 'Dynamic software updates in cyber-physical systems'. 2010 Int. Conf. on Information and Communication Technology Convergence (ICTC), Jeju, South Korea, November 2010, pp. 425–426

[122] La Manna, V.P., Greenyer, J., Clun, D., *et al.*: 'Towards executing dynamically updating finite-state controllers on a robot system'. Proc. of the Seventh Int. Workshop on Modeling in Software Engineering, Florence, Italy, May 2015, pp. 42–47

[123] Zhang, S., Huang, L.: 'Research on dynamic update transaction for Java classes', *Frontiers Comput. Sci. China*, 2007, **1**, (3), pp. 313–321

[124] Zhang, S., Huang, L.: 'Formalizing class dynamic software updating'. 2006 Sixth Int. Conf. on Quality Software (QSIC'06), Beijing, China, October 2006, pp. 403–409

[125] Murarka, Y., Bellur, U., Joshi, R.K.: 'Safety analysis for dynamic update of object oriented programs'. 2006 13th Asia Pacific Software Engineering Conf. (APSEC'06), Kanpur, India, December 2006, pp. 225–232

[126] Wahler, M., Richter, S., Oriol, M.: 'Dynamic software updates for real-time systems'. Proc. of the 2nd Int. Workshop on Hot Topics in Software Upgrades, Orlando Florida, October 2009, p. 2

[127] Stoyle, G., Hicks, M., Bierman, G., *et al.*: 'Mutatis mutandis: safe and predictable dynamic software updating', *ACM SIGPLAN Notices*, 2005, **40**, (1), pp. 183–194

[128] Shen, J., Bazzi, R.A.: 'A formal study of backward compatible dynamic software updates'. Software Engineering and Formal Methods, Trento, Italy, 2015, pp. 231–248

[129] Xu, X., Huang, L., Wang, D.: 'Supporting dynamic updates of componentized service'. IEEE Int. Conf. on Services Computing (SCC 2007), Salt Lake City, UT, USA, July 2007, pp. 699–700

[130] Kim, D.K.: 'Scheduling real-time tasks in the presence of dynamic software updates'. Int. Conf. on Hybrid Information Technology, Berlin, Heidelberg, August 2012, pp. 650–656

[131] Stanek, J., Kothari, S., Nguyen, T.N., *et al.*: 'Online software maintenance for mission-critical systems'. 2006 22nd IEEE Int. Conf. on Software Maintenance, Philadelphia, PA, USA, September 2006, pp. 93–103

[132] Liu, J., Tong, W.: 'A framework for dynamic updating of the service pack on the internet of things'. the Internet of Things (iThings/CPSCom), 2011 Int. Conf. on and 4th Int. Conf. on Cyber, Physical and Social Computing, Dalian, China, October 2011, pp. 33–42

[133] Sinha, S., Koedam, M., Breaban, G., *et al.*: 'Composable and predictable dynamic loading for time-critical partitioned systems on multiprocessor architectures', *Microprocess. Microsyst.*, 2015, **39**, (8), pp. 1087–1107

[134] Seifzadeh, H., Kazem, A.A.P., Kargahi, M., *et al.*: 'A method for dynamic software updating in real-time systems'. Eighth IEEE/ACIS Int. Conf. on Computer and Information Science, 2009. ICIS 2009, Shanghai, China, June 2009, pp. 34–38

[135] Wu, Q., Fan, L.: 'On a dynamic software update framework for messaging oriented middleware services'. 2007 Int. Conf. on Wireless Communications, Networking and Mobile Computing, Shanghai, China, September 2007, pp. 6521–6524

[136] Gharsellaoui, H., Khalgui, M.: 'Dynamic reconfiguration of intelligence for high behaviour adaptability of autonomous distributed discrete-event systems', *IEEE Access*, 2019, **7**, pp. 35487–35498

[137] Wohlin, C., Runeson, P., Höst, M., *et al.*: '*Experimentation in software engineering*' (Springer Science & Business Media, Springer-Verlag Berlin Heidelberg, 2012)

[138] Cook, T.D., Campbell, D.T.: '*Quasi-experimentation: design and analysis for field settings*', vol. **3** (Rand McNally, Chicago, 1979)