Specification Clones: An empirical study of the structure of Event-B specifications

Marie Farrell*, Rosemary Monahan, and James F. Power

Department of Computer Science, Maynooth University, Ireland

Abstract. In this paper we present an empirical study of formal specifications written in the Event-B language. Our study is exploratory, since it is the first study of its kind, and we formulate metrics for Event-B specifications which quantify the diversity of such specifications in practice. We pay particular attention to refinement as this is one of the most notable features of Event-B. However, Event-B is less well-equipped with other standardised modularisation constructs, and we investigate the impact of this by detecting and analysing specification clones at different levels. We describe our algorithm used to identify clones at the machine, context and event level, and present results from an analysis of a large corpus of Event-B specifications. Our study contributes to furthering research into the area of metrics and modularisation in Event-B.

1 Introduction and Motivation

The Event-B language is a state-based formal method for system-level modelling and verification that combines set theoretic notation and event-driven modelling [3]. Event-B is an industrial-strength tool and examples of its industrial use include train systems, air-traffic control and medical devices. A long term goal of model-driven software development has been to integrate such formalisms with practical software engineering methods and tools. Since the introduction of Event-B, a large number of examples and case studies have been conducted using the formalism, yet there is very little data available on the typical size, scope or structure of Event-B specifications.

In this paper we analyse Event-B specifications essentially as *software artefacts*, and extend software engineering techniques to the Event-B language. We have approached this empirically, by assembling a large corpus of Event-B specifications and developing basic metrics to quantify their size and complexity. Since *refinement* is a key feature of the Event-B approach, we seek to quantify this aspect of Event-B specifications in particular, so we can understand how such refinement is carried out in practice [11].

Apart from refinement, the modularisation constructs in Event-B are not well-developed, and a number of alternatives have been proposed to address this. As a contribution to the development of modularisation constructs for Event-B,

* Contact author: mfarrell@cs.nuim.ie. This project is funded by a Government of Ireland Postgraduate Grant from the Irish Research Council.

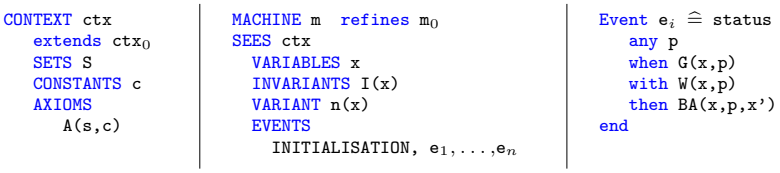

Fig. 1: The general structure of Event-B definitions of contexts, machines and events.

we conduct a study of *clones* in our corpus of Event-B specifications. Studies of this kind already exist for software written in a variety of programming languages, but we believe this is the first time this topic has been addressed at the specification level.

This paper is structured as follows. In section 2 we describe the background and motivation of our work. In section 3 we summarise our exploratory analysis of the corpus of Event-B projects that we have assembled. This allows us to quantify metrics and provide some insight into the refinement process used by developers. Section 4 describes the algorithm that we used to detect specification clones throughout our corpus. In section 5 we summarise the results of this clone detection under the specific headings of context, machine and event clones. We also outline potential ways of reducing the number of clones here. We identify threats to the validity of this work in section 6 and in section 7 we outline our contributions and potential for future work.

2 Background and Related Work

The primary objective of Event-B is to provide a basis for proving the *safety* of a given specification. This is achieved in practice through the Rodin Platform, an Eclipse-based IDE that is the de facto standard for Event-B [2]. Using Rodin, developers can write and type-check Event-B specifications, and use both automatic and interactive theorem proving to discharge *proof obligations* associated with the specification. The Event-B language supports formal refinement enabling the developer to start with a simple, abstract system and gradually add complexity in a verifiable way by means of *refinement steps* [11].

Figure 1 gives an overview of the general structure of Event-B specifications. Event-B models a system using two kinds of components: *contexts* and *machines*. A context is used to model static data using sets, constants and axioms [2], as shown in the leftmost column of Figure 1. The central column of Figure 1 shows the general format of a machine definition, which models dynamic behaviour in terms of a set of *events*. Machines can define the state variables and constrain them using variants and invariants.

The rightmost column of Figure 1 shows the general structure of an event. Here, p is a set of event parameters, $G(x, p)$ formalises a guard predicate over the set of event parameters p and the machine variables x. $W(x, p)$ is a witness predicate and the action $BA(x, p, x')$ is a before-after predicate where x' indicates the after values of the machine variables x. Each event is paired with a

status that can be one of `ordinary, convergent` or `anticipated`. Events that are labelled as `convergent` must strictly decrease the variant expression whereas those that are labelled as `anticipated` must not increase the variant expression. Events that have a status of `ordinary` do not need to obey any such properties.

There has been some work done on identifying suitable metrics for Event-B developments using the Halstead model [12]. Their objectives were to determine the size of an Event-B specification, the difficulty in constructing it and the effort required in designing and proving. Their case study was limited to just one project with 7 machines, and it is not clear whether the Halstead metrics, dependent on applying formulae to operations and operands, are the most appropriate way of characterising Event-B specifications in general.

2.1 Clones in Code and Specifications

The detection, analysis, management and tool evaluation corresponding to *code clones* represents a growing research area in the field of software engineering [15]. The reuse strategy indicated by code cloning is often beneficial in that it promotes the reuse of reliable code and can save time and effort in development. It is often the case, however, that duplicated code is caused by limitations in the programming paradigm's modularisation mechanisms and thus signals that improvements are required.

Roy et al. identify four different types of code clones [15], based on categorising the nature of the match between different pieces of code:

Type-1: identical code fragments that differ only in variations of white space and comments.

Type-2: structurally/syntactically identical code fragments that differ only in the names of identifiers, literals, types, layout and comments.

Type-3: a more liberal version of Type-2 clones which allow differences such as additions, deletions or modifications of statements.

Type-4: code fragments that exhibit the same functional behaviour but are implemented through very different syntactic structures.

In this paper we extend these definitions to detect clones between Event-B machines, contexts and events. Some work on identifying clones at the specification level has been done as part of the Arís project which retrieves reusable software artefacts using a graph matching approach [13]. However, this approach was based on finding matches in Spec#/C# code, and does not provide any data on the kind of clones found.

2.2 Modularisation of Event-B Specifications

There have been a number of suggested approaches to modularising Event-B specifications. One of the original methods proposed two styles of decomposition, based on the shared variable and shared event approaches [3]. Since then a variety of Rodin plugins have been developed to offer some degree of modularisation for Event-B. We do not have space to discuss them all here, but have listed the relevant plugins for Rodin in Table 1 along with a brief description of the

Name	Description	Reference
Feature Composition	Composition of Event-B machines and contexts and aids the user in resolving conflicts.	[8]
Generic Instantiation	Instantiate and reuse generic developments within other formal developments.	[17]
Model Decomposition	Decomposition of Event-B machines/contexts using the shared variable and shared event styles.	[18]
Pattern	Reuse of existing Event-B models including refinement steps within a development in order to save the modelling and proving effort.	[7]
Parallel Composition	Composition of Event-B machines using the shared event approach.	[14]
Modularisation	Allows the developer to construct modules and prove modular developments.	[9]
Renaming Refactory	Renames Event-B model elements so that the changes are propagated through the relevant machines, contexts and proof obligations.	See footnote[1].
Theory Extension	Extends the Event-B mathematical language (potentially with new data types) and the Rodin proving infrastructure.	[5]

Table 1: This table summarises the Rodin plugins that we have identified as relevant to our discussion in this paper.

modularisation features they provide. Since these plugins can potentially reduce the number of clones in Event-B specifications, we discuss them, where relevant, in our clone analysis results in section 5.

2.3 Experimental Setup

Since there has been no previous large scale study in this area, our focus will be on conducting an exploratory data analysis to identify and quantify the main characteristics of Event-B specifications.

In order to carry out this analysis we have assembled a corpus of Event-B specifications. We have obtained the projects in this corpus from a number of publicly-available Event-B resources, including the Event-B Wiki Page, the DEPLOY website and the case study tracks at the ABZ conference (2014 and 2016). Some additional projects were obtained directly from the developers who constructed them. In total we obtained 85 Event-B projects, ranging from smaller textbook-style examples through to large-scale developments.

All of the specifications in these 85 projects could be processed using the Rodin platform, and were thus available as a set of XML files in a standardised format. To analyse these projects we developed a suite of Python programs that read in the files in Rodin format, calculated and reported metrics, and searched for occurrences of clones at various levels.

[1] `http://wiki.event-b.org/index.php/Refactoring_Framework`

Smaller Projects								
Project	Macs	Cons	Evs	Refs	Sens	Auto	Inter	RP
Bepiv6.4*	2	10	45	1	948	560	370	0
SSF_pilot	4	4	35	3	842	170	2	19
DynStabLSR	7	1	69	6	788	247	140	37
ch8circarbiter	6	2	46	5	764	153	0	31
TreeFilePerm	4	4	33	3	655	107	52	18
RCPert	4	3	53	3	583	199	28	29
RCNorm	4	3	49	3	565	146	32	27
ch912_mobile	6	1	43	5	539	134	19	19
ch917_train	5	3	38	4	539	128	5	23
SignalControl	10	4	106	9	497	135	0	26
ch7_conc	5	1	45	4	484	239	9	22
routing_new	8	4	51	7	479	226	60	47
FloodSet	6	5	27	5	445	209	87	46
ch2_car	4	3	34	3	438	249	4	17
ssf	7	4	51	6	430	48	11	8
seqpattern	5	2	30	3	425	37	1	4
SharedBuffs	4	1	22	3	423	98	5	19
SimpleLyra	4	4	28	3	418	55	0	3
gcd	7	3	32	6	407	91	84	21
ch8circpulser	9	0	52	7	397	93	1	20
ch916_doors	5	3	31	4	380	101	2	14
ch8circroad	5	0	27	4	379	37	0	9
ch8circight	3	0	19	2	370	89	0	25
ch6_brp	6	3	47	5	360	149	0	16
Modes_v2	3	3	30	2	333	108	13	3
aocs_t2	3	2	29	2	297	105	13	18
CtsCtrl	4	3	18	2	274	150	19	21
Rabin	7	7	62	6	262	138	71	2
pomc	5	3	27	4	257	81	27	10
pomcwoterm	5	3	27	4	250	83	13	10
ch913_ieee	6	3	21	4	243	71	21	16
DSAOCSSv3	1	1	9	0	233	82	8	0
AStyleQR	5	1	19	4	226	70	5	14
DSAOCSSv2	1	1	9	0	219	81	8	0
ch4_file_1	5	2	17	4	192	47	5	9
FindP_P1	4	1	21	3	191	27	15	13
aocs_t2_um	2	2	16	1	178	95	8	13
pat9QR	5	0	22	4	161	44	6	8
BinarySearch	3	1	14	2	154	102	6	13
SSF1	1	3	6	0	148	25	0	0
ch911_tree	5	3	15	4	140	81	0	9
SSF_minipilot	1	1	8	0	127	20	4	0
Club-120130	3	4	11	2	105	50	7	5
BoschSwitch	2	1	10	0	102	15	4	0
ch915_sort	3	1	12	2	101	56	11	11
ex-bubblesort	2	1	7	1	98	46	6	7
FindP_D	2	2	8	1	98	47	2	5
FindP_G	1	1	6	0	94	0	0	0
program2	2	2	9	1	88	192	5	3
Zer_ess	0	5	0	0	88	40	15	0
ex-bubbles	2	1	8	1	83	41	1	13
HermanRing	2	3	8	1	82	35	22	2
cae-square	3	3	10	2	78	53	1	2
primrec	2	2	7	1	74	36	0	2
FindP_P2	1	1	4	0	73	0	0	0
ch915_bin	3	1	11	2	68	32	5	7
AStyleQR_2	1	1	5	0	65	10	0	0
TrafficLights	2	1	11	1	58	20	0	0
ch915_inv	2	2	7	1	55	32	0	5
Cowboy	2	1	7	1	53	14	1	1
ch910_ring	2	2	6	1	52	24	4	1
ch915_sqrt	3	1	9	2	44	17	0	5
ch915_rev	2	1	6	1	43	28	3	4
ch915_mini	2	1	6	1	42	24	1	1
DiningCrypt	3	1	6	1	42	21	3	0
AStyleQR_3	1	1	3	0	40	6	0	0
AStyleQR_1	1	1	3	0	37	5	0	0
pat8SynMC	2	0	5	1	34	15	0	0
ch915_search	2	1	6	1	29	17	0	3

Table 2: Metrics for the projects that fall into the "smaller" category.

Legend for column headings:
Macs: # of machines
Cons: # of contexts
Evs: # of events
Refs: # of refinement steps
Sens: # of sentences
Auto: # of automatic proofs
Inter: # of interactive proofs
RP: # of designated refinement proofs

Larger Projects								
Project	Macs	Cons	Evs	Refs	Sens	Auto	Inter	RP
Midas*	43	61	2500	40	26395	2034	3163	2183
FlashFileFS	18	6	320	13	5442	974	531	88
DepSatSpec	14	2	2094	13	4771	1309	549	0
ATM	7	12	129	6	3447	925	37	46
B2Bminip	12	0	228	11	2900	425	73	124
Bepiv3.3	6	6	137	1	2665	153	113	12
TSHHDMac	35	50	1487	18	2661	602	84	15
Bepiv5.0	9	10	329	8	2007	683	317	0
CDIS	7	6	103	6	1894	101	0	3
HDMac	19	25	718	16	1605	448	23	2
Pilot_v3	4	4	98	3	1586	134	9	0
MLLanding	11	2	313	10	1432	286	210	0
FlashFileFL	6	12	109	5	1243	379	13	11
HLanding	11	7	321	9	1213	173	68	17
ch3_press	8	3	144	7	1200	0	0	0
OnbCont	9	3	224	8	1108	438	1	14

Table 3: Metrics for the projects that fall into the "larger" category. Outliers are indicated by an asterisk*.

All Projects ($n = 85$)								
Project	Macs	Cons	Evs	Refs	Sens	Auto	Inter	RP
Minimum	0	0	0	0	29	0	0	0
Median	4	2	27	3	274	83	5	8
Maximum	43	61	2500	40	26395	2034	3163	2183
MADN	3.0	1.5	28.2	3.0	298.0	86.0	7.4	11.9

Smaller Projects ($n = 69$)								
Project	Macs	Cons	Evs	Refs	Sens	Auto	Inter	RP
Minimum	0	0	0	0	29	0	0	0
Median	3	2	17	2	192	56	5	8
Maximum	10	10	106	9	948	560	70	47
MADN	1.5	1.5	16.3	1.5	206.1	54.9	7.4	11.9

Larger Projects ($n = 16$)								
Project	Macs	Cons	Evs	Refs	Sens	Auto	Inter	RP
Minimum	4	0	98	1	1108	0	0	0
Median	10	6	270	8	1950	431	70	11
Maximum	43	61	2500	40	26395	2034	3163	21.83
MADN	5.2	5.9	203.9	4.4	1076.4	398.1	97.1	17.0

Table 4: Summary statistics for the whole data set, and for the two "smaller" and "larger" subdivisions.

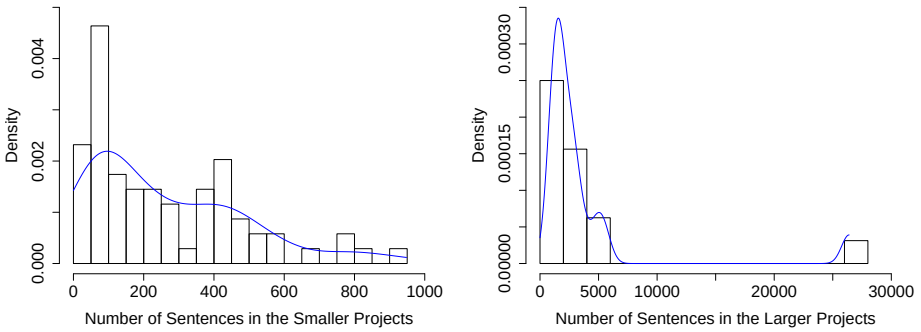

Fig. 2: Histograms showing the distribution of the numbers of sentences per project for the smaller and larger data sets. Note that the vertical axes here are on different scales.

3 Analysing a Corpus of Event-B specifications: Metrics and Refinement

The most obvious measurable entities in an Event-B specification correspond to the major syntactic categories. Just as the size of a software project might be measured using code metrics such as number of classes, methods or lines-of-code, we can get similar information from an Event-B specification in terms of the number of contexts/machines, events and sentences. Specific to a formal approach, we can also measure the number of proof obligations (automatically and interactively proved). The metric values for the 85 projects in the corpus are given in Tables 2 and 3.

In total, for all 85 projects in the corpus there are 359 contexts and 468 machines, which in turn contain 10828 events. One immediate difficulty in analysing the corpus is the overall range of the specifications, from small, textbook-style examples, through to major systems. We chose to divide the corpus based on the *number of sentences* in each project, since this was the metric closest to lines-of-code, which might best reflect a simple measure of size for a project. Thus the rows of Tables 2 and 3 are ordered based on the total number of sentences in a project.

In order to be able to represent this information meaningfully and extract useful information from it, we have split the corpus into two different data sets. We investigated a variety of ways by which to carry out this split, including:

- using the examples from the *Modeling in Event-B* textbook [1] as models of "smaller" projects, and regarding projects with more sentences than these as "larger" projects.
- extracting the outliers using Tukey's test (the median plus 1.5 times the inter-quartile range); all such outliers were larger projects.
- using trimming [10], to identify a fixed proportion at the extreme ends of the data set.

In practice, these three strategies resulted in almost the same set being identified, and we have used Tukey's test to categorise the 16 projects in Table 3 as "larger". This also corresponds to the top 19% of the projects, and excludes all but one of the textbook examples (the exception is the mechanical press controller from chapter 3). We refer to the 69 remaining projects listed in Table 2 as "smaller". These projects all have 10 contexts or under and 10 machines or under. Some of these are non-trivial projects, however and the number of sentences ranges from just 29 up to 948. Thus we have further divided Table 2 into quartiles based on the number of sentences.

Tables 2 and 3 demonstrate the diversity of Event-B developments and we provide them so that future studies have a measure with which they can gauge the comparative size of Event-B developments.

3.1 Metrics for Event-B Specifications

Figure 2 further illustrates the diversity in size between the projects, showing the distributions of the sentences in the smaller and larger projects. These measurements signal that one should be cautious when choosing a representative Event-B specification as the structures vary so much. In particular, the Midas project is a dramatic outlier of this data set on almost all metrics, as is shown by the rightmost bar in Figure 2, and thus should be considered quite distinctive as an Event-B specification.

Table 4 summarises the ranges for each of the metrics, giving the minimum, maximum, median and MADN values for the whole data set and its two subdivisions. Due to the uneven distribution we use the median and MADN as robust measures in place of the mean and standard deviation. MADN is the median of the absolute deviations from the median, divided by $z_{0.75}$ [10]. It is notable that in most cases the MADN is close to or exceeds the median, indicating a large spread of values for each of the metrics.

We analysed all of the metrics in Tables 2 and 3 to check for inter-relationships, using Spearman's rank correlation coefficient. The most notable very strong correlations (with $p < 0.001$ in all cases) were between the following variables:

- **the number of events and the number of sentences** in the small data set ($\rho = 0.905$), where the median number of sentences per event is 11 (MADN = 4.4). However, in the larger project set, this correlation is weak ($\rho = 0.391$). The larger projects contain a greater number of contexts, thus adding sentences to the projects that are not sentences within events.
- **the number of machines and the number of events** in both the smaller ($\rho = 0.849$) and larger ($\rho = 0.904$) project sets. The median number of events per machine is 25 (MADN = 9.8) in the larger set and 5 (MADN = 2.7) in the smaller.

There was also a (lower) strong correlation in the smaller projects between the numbers of events/sentences and the number of automatic proofs.

The data in Tables 2 and 3 shows that the number of automatic proofs required dramatically exceeds the number of interactive proofs in general. On average, in the larger projects, 78.6% of the proofs were done automatically with

91.1% of the proofs automatic in the smaller projects. This is important for automated verification, since it is a measure of the relative amount of theorem-proving work imposed on the user, as compared to that done by the underlying prover. It is notable that this percentage is much higher for smaller examples than for the larger ones. This is most likely due to the increased complexity in modelling large-scale systems. As Event-B continues to be used industrially, this metric can be useful in measuring the degree to which automated theorem-proving has increased in effectiveness.

3.2 Quantifying Refinements

Figure 3 contains a histogram with kernel distribution, showing the number of refinement steps for each of the project sets. As can be seen, in the larger project set the Midas project is again a dramatic outlier with 40 refinement steps. The smaller project set does not contain any dramatic outliers, with approximately 50% of these projects containing only one refinement step.

In both the smaller and the larger project sets there is a very strong correlation between the number of machines and the number of refinement steps in a project ($\rho = 0.989$ and $\rho = 0.993$ respectively, $p < 0.001$). In most cases the relationship is almost 1:1, showing that *linear* refinement chains are the most common refinement strategy used. By default, a machine can refine at most one other, so typically a machine will have one 'parent'. These refinement chains bear a striking similarity to the notion of refinement presented in the theory of institutions which is typically a single, linear chain [16]. While the *Feature Composition* plugin for Rodin allows the merging of machines in a refinement step [8], this is clearly not the usual approach taken in these examples.

In Event-B, proof obligations are one indicator of the complexity of the system being modelled. There is a specific set of proof obligations that are generated through the refinement of events (guard strengthening and merging, action simulation, equality of a preserved variable, witness well-definedness and witness feasibility). We list the number of these designated refinement proofs in the rightmost column of Tables 2 and 3. These proofs are only generated for refined events that are labelled as `not extended`. Events that are labelled as `extended` generate no proof obligations that are designated for refinement as they are specific to superposition refinement. This is quite an efficient approach to refinement as Rodin avoids the regeneration of these proofs [2], but is only applicable where no data refinement has taken place.

There is a strong correlation between the number of refinement proofs and the number of refinement steps in a project in the smaller project set ($\rho = 0.786$, $p < 0.001$) resulting in the median ratio of 3 refinement proofs to 1 refinement step. However, the correlation is not significant for the larger project set. We found that developers of the larger projects often opted to avoid data refinement and use event extending to streamline their developments. Based on the data in Table 3 we can identify 5 out 16 projects that used this approach.

We had expected that there might be a correlation between the number of refinements and the number of sentences, with machines increasing in size as

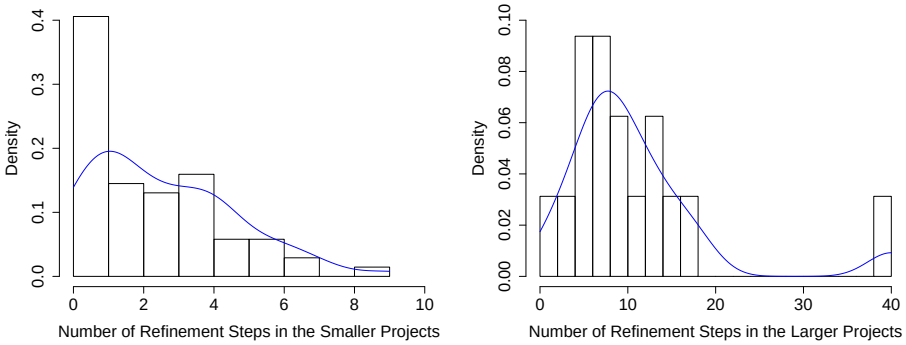

Fig. 3: Histograms with kernel distribution describing the number of refinement steps taken in both the smaller and larger project sets. Note that the vertical axes here are on different scales.

they became more concrete. However, this correlation is not strong even the smaller data set ($\rho = 0.695$, $p < 0.001$) and neither strong nor significant in the larger, which, as mentioned earlier, are also influenced by the large number of contexts.

4 Detecting Specification Clones

In this section we describe our strategy for applying the clone types discussed in section 2.1 to Event-B.

In all cases we will be comparing *sentences* from one specification with those in another: this includes axioms in contexts, invariants and variants in machines, and guards, witnesses and actions in events. There are a number of approaches to matching in the literature, including metric, token, text and abstract syntax comparison [4]. Since our sentences are relatively small constructs, we have used these as the smallest unit of matching. All sentences are tokenised to eliminate formatting and white-space, and we compare only sentences of the same kind (thus axioms with axioms etc.). We have discounted any machines/contexts/events with 2 or less sentences in order to ensure that we are only collecting meaningful clones.

We carry out this matching at three levels: contexts, machines and events. We base our search for clones on the clone types discussed in section 2.1. In all cases, (context, machine and event):

- Type-1 clones correspond to exact matches between the full sentence sequences in each case: that is all sentences in one component must match all those in the other.
- Type-2 clones are matches between the full sentence sequences, but where variable names are anonymised, each variable name being replaced by a positional indicator.

– Type-3 clones are also matches between two sentence sequences (with variable names anonymised or unanonymised), except that now we allow matches between *sub-sequences* of the sentences. We calculate the percentage of type-3 clone similarity using the maximum of the similarity calculated for both the anonymised and unanonymised versions.

We do not explicitly search for type-4 clones (functional equivalence) in what follows. From one perspective, all of our clones could be viewed as type-4, since we are not really comparing code but specifications, and thus identifying a degree of functional equivalence. However, a more robust search for type-4 clones would require us to prove the equivalence of the corresponding generated proof obligations for machines, contexts and events, which we have not attempted. As such, we omit type-4 clones from further discussion here as future work.

We have conducted an automated analysis of our corpus of projects by writing a series of Python scripts that read in the Rodin files, represent the components as an abstract syntax tree, and then perform comparisons at the context, machine and event level. We analyse machines and events both with and without any corresponding variants and invariants included, to distinguish between sentences that are global and local (to events) in the machine. Variants are only included with events that have a status of `anticipated` or `convergent`, since, unlike `ordinary` events, these are required to not increase the variant expression [1].

Our analysis returns pairs corresponding to instances of cloning that have occurred. We refer to these as clone pairs or clonings in what follows.

We have also identified the clones that occur the most frequently throughout our corpus, at the level of machines, contexts and events. As there are no libraries for Event-B specifications and since contexts typically supply custom data types, we were interested to examine whether or not similar contexts have been used in the Event-B projects across our corpus. Thus we also determine whether the clones that we have discovered are *inter*-project (across different projects) or *intra*-project (within the same project) clones.

5 Results of the Clone Analysis

In this section we summarise the results of our clone analysis through the entire corpus. In what follows we regard the three clone types as mutually exclusive: by type-2 we mean all those that are type-2 but not type-1, and by type-3 we mean those that are type-3 but not type-1 or type-2. Table 5 summarises the results of this analysis, providing counts for the number of clonings identified (type-1, type-2 and type-3) and also the number of clones.

5.1 Context Clones

As can be seen in the first row of Table 5, our analysis found 40 clone pairs at the context level in the corpus, consisting of 18 type-1 and 22 type-3 clone pairs. We had expected this, since contexts resemble data types in a programming

Event-B Component	Clone Pairs				Actual Clones	
	Type-1	Type-2	Type-3	Total	Total	Occur.
Contexts	18	0	22	40	22	51
Machines	13	7	937	957	19	40
Machines (+VI)	9	7	943	959	13	28
Events	276	942	4781	5999	131	417
Events (+VI)	35	158	7229	7422	65	175

Table 5: The occurrence of clone pairs and clones per type throughout the entire corpus. Note that '(+VI)' indicates that the variants (where appropriate) and invariants have been included in the analysis.

language. The *theory plugin* offers a potential solution to this problem as it provides a way of adding new data types to Rodin [5].

When we investigated the actual clones that were returned we found 22 context clones, of which 18 occurred on an inter project basis and 6 on an intra project basis. There were 2 which occurred both as inter and intra project clones. The fact that so many of them occurred between different projects supports our claim that they are being re-used in a manner similar to libraries. We found that the inter project clonings occurred mostly between projects that shared a common approach or were between projects that were modelling the same kind of system. For example, there were quite a few inter project clonings in the separate developments of a Hemodyalysis Machine, the different versions of BepiColombo, and the various kinds of file systems being modelled (Flash FS, Flash FL and Tree FS).

5.2 Machine Clones

In Event-B, a machine is generally reused by means of refinement and thus we did not expect to find many type-1 clonings or inter project clones. As can be seen in the second and third data rows of Table 5, we discovered a very small number of type-1 and type-2 machine clonings. We did, however, manage to identify 937 type-3 clone pairs in the analysis without the variants and invariants included.

Since the type-3 clone pairs are identified in terms of their similarity, expressed as percentages, we provide an illustration of the distribution of type-3 clones in Figure 4. The top two histograms in Figure 4 show the data for machine-level clone pairs, and the bottom two for event-level clone pairs. As expected, the distributions for machine-level clones skew to the left, as most clones had a low similarity percentage, indicating that there is some basic machine structure being reused over and over again but the part that is being cloned does not contain a large proportion of the sentences. Nonetheless, there is still a significant number of clone pairs that have at lest 50% of their sentences matching.

In total we found 5 inter and 14 intra project full machine clones. This reduced to 3 inter and 10 intra project clones when the variants and invariants were included. Most of these were within the same project and therefore were most likely caused by refinement chains. These numbers are quite small with

regards to the size of our corpus, thus we conclude that full machines typically do not incur a huge amount of cloning.

5.3 Event Clones

Since events are the smallest unit of modularisation, we expected a higher level of cloning to be found between pairs at this level. The fourth data row of Table 5 shows that we identified 276 type-1, 942 type-2 and 4781 type-3 clone pairs or instances of event clonings in our corpus. As can be seen from the fifth data row in Table 5, this number decreased for type-1 and type-2 when we included the appropriate variants and invariants (35 and 158) respectively. The number of type-3 clone pairs, however, increased quite dramatically to 7229. This is because the inclusion of variants and invariants increased the size of many small events past our threshold of 2 sentences, thus including events in the analysis that were absent when these variants and invariants were not included.

There were 131 different event clones, of which 30 were inter and 126 were intra project clones. Intra project clonings occurred 382 times and they occur in the scenarios where one event is refined throughout a project and also where there are event clonings within the same machine. We found 210 situations where one event in a machine was a clone of another event in the *same* machine. This accounts for approximately 1.9% of the total events in our corpus and 17.2% of the total type-1 and type-2 event clone pairs. Inter project clonings occurred a total of 37 times.

Based on this analysis, we conclude that there may be a relationship between the number of intra event clones between different machines in the same project and the level of refinement of that project. However, this needs to be examined in more detail.

5.4 Discussion: Dealing with Clones

One way of addressing the large number of type-2 clones at the event level would be through the the provision of facilities for event re-use. This could be done either through a renaming feature as a Rodin plugin, or by introducing parmeterisation constructs at the Event-B language level.

The *renaming refactory* plugin could offer some assistance here as it renames components of an Event-B model with the renamings propagating through to the proof obligation level. However, it does not offer any way of instantiating copies of events. The *Pattern* and the *Generic Instantiation* plugins are also relevant, but these currently work only at the machine level, rather than the event level [7, 17].

If more sophisticated modularisation constructs were made available for Event-B, they could potentially alter the development strategy taken by developers and turn what would have been type-3 clones into type-2 clones which could be parametrised and then added to in future refinements. We have proposed the theory of institutions as a mathematically sound framework to incorporate Event-B

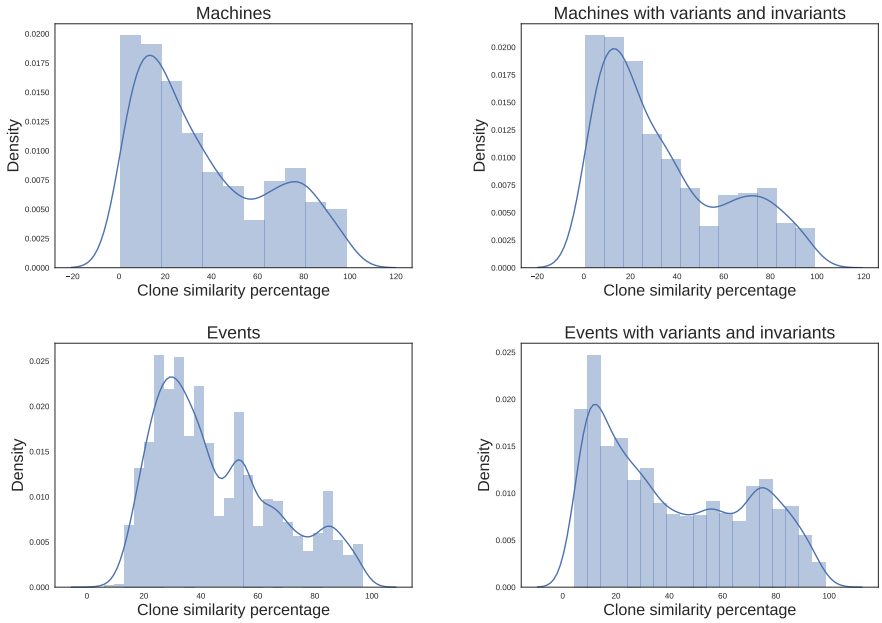

Fig. 4: Histograms describing the distribution of Type-3 clones across the entire corpus of Event-B specifications. Note that we have omitted type-3 *context* clones as there were relatively few of these.

into and thus provide users of Event-B with access to an array of generic and formalism independent modularisation constructs through the use of specification building operators [6]. These specification building operators could potentially provide a solution to these problems.

6 Threats to Validity

One feature of our work is the creation of a corpus of Event-B projects, and our division of this set into smaller and larger projects. The selection poses a threat to *conclusion validity*, since we are dealing with a heterogeneous group of projects, and there is a risk that the differences in metrics are due to other factors not measured here, such as heterogeneity in terms of the domain of application, e.g. railway, health-care, control systems, algorithms etc.

Our analysis of the projects is conducted based on the metrics that we have defined and measured. While these metrics corresponded to major syntactic categories in Event-B and have clear analogies with similar constructs in programming languages, there is a threat to *construct validity* here. In particular, further studies would be required to establish the predictive value, if any, of these metrics.

Similarly, in adapting the definition of code clones to Event-B we made a number of decisions on what should be measured and the degree of matching involved; altering these could yield different results. Our measurement of type-

3 clones was based on sentence sequences and the in-order anonymisation of variables: a more general technique could produce more clone-pairs, at the cost of a considerable increase in combinatorial matches.

Since our analysis was based on processing the XML files generated by Rodin, we have a high degree of confidence that the measurements are accurate, and do not pose a threat to the *internal validity* of our results. However, in three of the Event-B projects (`ch3_press`, `FindP_G` and `FindP_P2`) the corresponding .bps files, which hold information about the proofs, were empty. Thus these projects have no automatic or interactive proofs recorded even though proof obligations have been generated. We believe that these projects may have used an older version of Rodin or a plugin that we do not have access to. One approach to resolving this would be to remodel them using a current version of the software with no extra plugins installed. We chose not to do this as we wished to remain as impartial as possible with regards to the corpus that we collected.

In total, we have 85 Event-B projects in our corpus, but it is possible that this is not a large enough sample size to study. This causes a threat to *external validity* in terms of the generalisability of our results. We believe that assembling and maintaining a measured corpus of Event-B programs is a worthwhile task in this regard.

7 Conclusions and Future Work

Our work applies the existing software engineering approaches of calculating metrics and detecting code clones to specifications written using the Event-B formal method. This exploratory study is the first of its kind and has enabled us to provide and analyse the metrics of a corpus of Event-B specifications. In this way, we provide a benchmark against which other Event-B developments can gauge their comparative size and complexity level.

During the evolution of the Event-B formalism from Classical-B, certain facilities for the reuse of machine specifications disappeared such as the modularisation properties supplied by the keywords `INCLUDES` and `USES` which facilitated the use of an existing machine in other developments [17]. It is evident not only from experience with industrial projects [9] but also from the sheer abundance of attempts to regain such modularity features for Event-B that there is an underlying requirement for it. Our empirical study supports this claim by evaluating code clones at the specification level.

Future work includes the assessment of *clone genealogies*, particularly in the context of refinement - i.e. how clones evolve throughout successive refinements. This study would show us whether or not clones persist in the specification after it has undergone a (series of) refinement step(s). We are also interested in detecting non-typing invariant clones, this would allow us to analyse data refinement clones using gluing invariants.

References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
2. J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*, 12(6):447–466, 2010.
3. J.-R. Abrial and S. Hallerstede. Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. *Fundamenta Informaticae*, 77(1-2):1–28, 2007.
4. I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *International Conference on Software Maintenance*, pages 368–377, Maryland, USA, 1998.
5. M. Butler and I. Maamria. Practical theory extension in Event-B. In *Theories of Programming and Formal Methods*, volume 8051 of *LNCS*, pages 67–81, 2013.
6. M. Farrell, R. Monahan, and J. F. Power. Providing a semantics and modularisation constructs for Event-B using institutions. In *International Workshop on Algebraic Development Techniques*, Gregynog, Wales, 2016.
7. A. Fürst. Design patterns in Event-B and their tool support. Master's thesis, Department of Computer Science, ETH Zürich, March 2009.
8. A. Gondal, M. Poppleton, and C. Snook. Feature composition-towards product lines of Event-B models. In *International Workshop on Model-Driven Product Line Engineering*, pages 18–25, Twente, The Netherlands, 2009.
9. A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala. Supporting reuse in Event-B development: Modularisation approach. In *Abstract State Machines, Alloy, B and Z*, volume 5977 of *LNCS*, pages 174–188, 2010.
10. B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong. Robust statistical methods for empirical software engineering. *Empirical Software Engineering*, pages 1–52, 2016.
11. C. Morgan, K. Robinson, and P. Gardiner. *On the Refinement Calculus*. Springer, 1988.
12. M. Olszewska and K. Sere. Specification Metrics for Event-B Developments. In *International Conference on Quality Engineering in Software Technology*, Dresden, Germany, 2010.
13. M. Pitu, D. Grijincu, P. Li, A. Saleem, R. Monahan, and D. P. O'Donoghue. Arís: Analogical reasoning for reuse of implementation & specification. In *International Workshop on Artificial Intelligence for Formal Methods*, pages 13–16, Rennes, France, 2013.
14. M. Poppleton. The composition of Event-B models. In *International Conference on Abstract State Machines, B and Z*, pages 209–222. Springer, 2008.
15. C. K. Roy, M. F. Zibran, and R. Koschke. The vision of software clone management: Past, present, and future. In *Software Maintenance, Reengineering and Reverse Engineering*, pages 18–33, Antwerp, Belgium, 2014.
16. D. Sanella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Springer, 2012.
17. R. Silva and M. Butler. Supporting reuse of Event-B developments through generic instantiation. In *International Conference on Formal Engineering Methods*, volume 5885 of *LNCS*, pages 466–484, 2009.
18. R. Silva, C. Pascal, T. S. Hoang, and M. Butler. Decomposition tool for event-B. *Software: Practice and Experience*, 41(2):199–208, 2011.