Contract Usage and Evolution in Android Mobile Applications (Supplementary Material*)

DAVID R. FERREIRA, Faculty of Engineering, University of Porto, Portugal
ALEXANDRA MENDES, Faculty of Engineering, University of Porto, Portugal
JOÃO F. FERREIRA, INESC-ID & IST, University of Lisbon, Portugal
CAROLINA CARREIRA, Carnegie Mellon University, INESC-ID & IST, University of Lisbon, Portugal

This document contains additional material that is not included in the submission "Contract Usage and Evolution in Android Mobile Applications" (Last Revision: 29 April 2025).

Contents

Authors' addresses: David R. Ferreira, Faculty of Engineering, University of Porto, Porto, Portugal, david.regatia@gmail.com; Alexandra Mendes, Faculty of Engineering, University of Porto, Porto, Portugal, alexandra@archimendes.com; João F. Ferreira, INESC-ID & IST, University of Lisbon, Lisbon, Portugal, joao@joaoff.com; Carolina Carreira, Carnegie Mellon University, INESC-ID & IST, University of Lisbon, Lisbon, Portugal, carolinacarreira@cmu.edu.

SUPPLEMENTARY MATERIAL

The following appendices contain additional material that complements the main body of the paper.

A INFORMATION ABOUT AVAILABLE DATA / ARTIFACTS

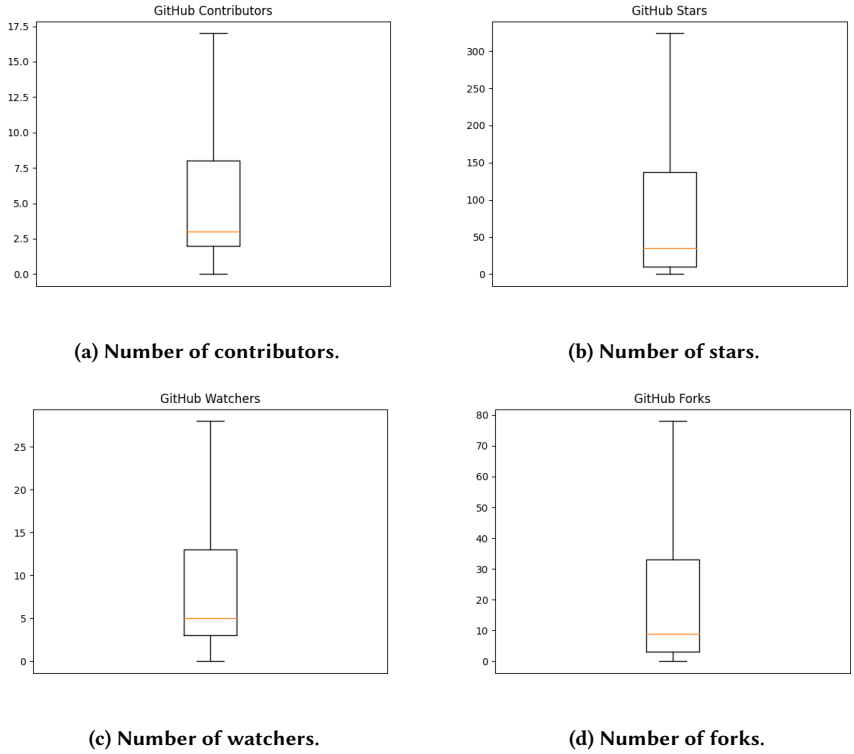
All the code and datasets are publicly available at https://figshare.com/s/d6eb7e5522bb535dc81a

Alternatively, the following links can also be used:

- The code is available at: https://anonymous.4open.science/r/contracts-android-3E30
- The dataset is available at: https://drive.google.com/file/d/1X8Qy3yamzjIZyc_h5AtNW84-91qFTiV4/view?usp=share_link

B DATASET: GITHUB STATISTICS

Figure 1 shows the distribution of GitHub-related metrics — including the number of *contributors*, *stars*, *watchers*, and *forks* — for the projects that form the evaluated dataset. While the number of *contributors* describes the project's team and its size, the number of *stars*, *watchers*, and *forks* help to assess each project's popularity and relevance among other developers. For reference, the maximum outlier for each metric is 1682 for watchers, 33,689 for stars, 11,633 for forks, and 398 for contributors. Again, this diversity ensures the quality of the dataset and reduces potential bias.



(a) Number of contributors. **(b) Number of stars.**

(c) Number of watchers. **(d) Number of forks.**

Fig. 1. The distribution of GitHub repositories-related metrics for the dataset's projects, without outliers.

C CONTRACTS IN ANDROID APPLICATIONS

C.1 CREs

An exception can be used to signal, at runtime, a contract violation. However, it is important to note that the exception itself does not represent a contract; it needs to be associated with a previous check — for example, an exception thrown inside an *if-else block* — to be considered so. Java and Kotlin offer many exceptions that can be used for this purpose, such as the *IllegalArgumentException*. The *android.util* package offers additional exceptions that we are also interested in analyzing, such as the case of the *AndroidRuntimeException*. We are also interested in a particular exception, the *UnsupportedOperationException*, which, according to the Java documentation, is thrown to indicate that the requested operation is not supported. An example of this is the following method *proceedWithCheckout*, which states that it can only perform its task when the *shoppingCart* has at least one item:

```
public void proceedWithCheckout(List<Item> shoppingCart)  {
  if (shoppingCart.isEmpty()) {
    throw new IllegalArgumentException();
  }
  ...
}
```

C.2 APIs

The methods provided by the *Validate* class are simply wrapping exceptions that we have already considered in the CREs. Still, as we can see in the following Kotlin code snippet, this API contributes to cleaner code compared to a raw CRE-based solution since we can specify the contracts in a single line and with meaningful wording. In particular, we specify a precondition stating that *the items list is not empty*:

```
fun addToShoppingCart(items: List<Item>): List<Item> {
  Validate.notEmpty(items)
  shoppingCart.addAll(items)
  return shoppingCart
}
```

D ALGORITHM FOR DIFF RECORDS OF THE CONTRACTS

The algorithm to create *diff records* is illustrated in Figure 2. The algorithm to create *diff records* consists of walking through each contract identified in the *usage* study (steps 1 and 2). We create a unique index for each contract in the loop to ensure we are not double-counting occurrences (steps 3 and 4). If the contract was not analyzed yet, we determine whether the contract belongs to the first version of the application (step 5). If this is the case, we create a *diff record* by retrieving all the contracts in both versions of this contract's method (steps 5.a and 5.b). Otherwise, if the contract belongs to the last version of the application (step 6), we determine whether the associated method already existed in the first version (step 7). If the method existed and its first version did not contain contracts, we create a diff record with only the last version's contracts (steps 8 and 9). If the first version contains contracts, we do not create a *diff record* to not double-count contracts since they will be reported by step 5.b. Ultimately, the program outputs the diff records created for each program-version method (step 10).

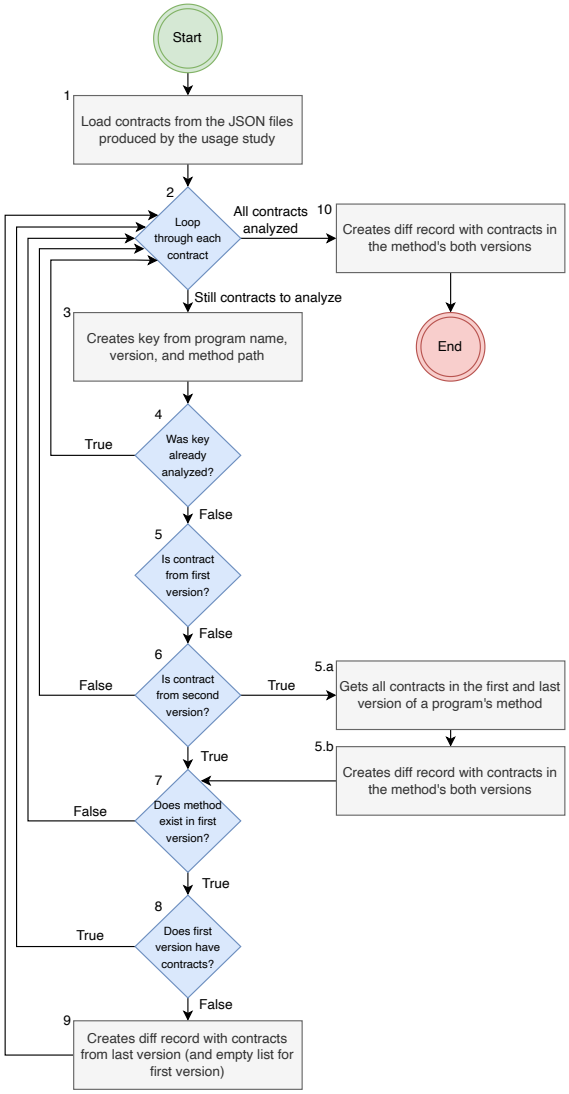


Fig. 2. An overview of the algorithm to create *diff records* of the contracts found in two versions of a method.

E LISKOV SUBSTITUTION PRINCIPLE STUDY: EXAMPLE

Listing 1 shows the *TagEntry* class that extends the *EntryItem* class. It also overrides the *setName* method inherited from its parent class. Note that while the superclass implementation contains no contract, the subclass implementation adds a CRE precondition throwing an *IllegalStateException* when the *id* property does not end with the *name* parameter. Therefore, we are in the presence of a *precondition strengthening* in the context of *inheritance*, i.e., a violation of the Liskov Substitution Principle.

```
public class EntryItem {
  public void setName(String name) {
    if (name != null) {
      this.name = name;
      this.normalizedName = StringNormalizer.normalizeWithResult(this.name, false);
    } else {
      this.name = "null";
      this.normalizedName = null;
    }
  }
}

public class TagEntry extends EntryItem {
  public final String id;

  @Override
  public void setName(String name) {
    if (name != null) {
      if (!id.endsWith(name))
        throw new IllegalStateException("The display name and tag name need to be equal.");
      super.setName(name);
    } else {
      super.setName(id.substring(SCHEME.length()));
    }
  }
}
```

Listing 1. A Java class that overrides the setName method from its parent class. This is an example of *pre-condition strengthening* in the context of *inheritance*, i.e., a violation of the Liskov Substitution Principle.

F FINDING 4: TOP 100 APPLICATIONS

Regarding Finding 4, we have also looked into whether there was any relation between the number of contracts in the last version and any of the GitHub metrics from Figure 1. However, no meaningful correlation was found.

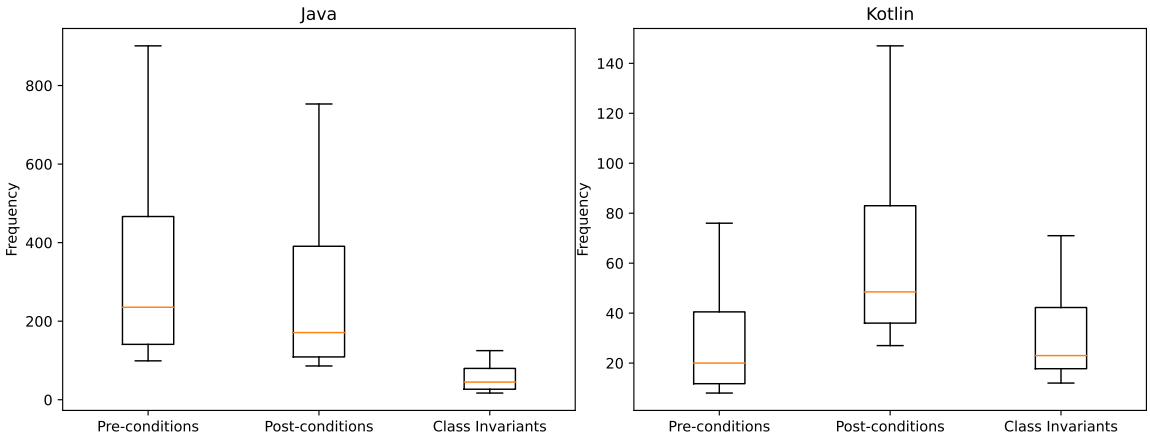


Fig. 3. Comparison of the distribution of the identified contract types in the top 100 applications with higher usage per type for Java and Kotlin.

G TABLE 6 EXTENDED

In this section, we present a version of Table 6 (in the submitted paper) that also includes the number of applications (last two columns).

Table 1. Number of contracts found in the dataset by construct and category.

		contracts (all ver.)		contracts (2nd ver.)		applications	
Construct	Category	Java	Kotlin	Java	Kotlin	Java	Kotlin
cond. runtime exc.	CRE	25,565	3,232	14,887	2,071	779	285
unsupp. op. exc.	CRE	511	142	308	116	97	27
java assert	assertion	3,525	-	2,217	-	325	-
kotlin assert	assertion	-	3,868	-	2,370	-	234
guava precond.	API	1,798	10	1,121	9	22	4
commons validate	API	148	0	3	0	1	0
spring assert	API	1	0	1	0	1	0
JSR303, JSR349	annotation	0	0	0	0	0	0
JSR305	annotation	4,195	20	2,133	13	40	4
findbugs	annotation	0	0	0	0	0	0
jetbrains	annotation	2,310	138	1,596	98	115	20
android	annotation	12,003	5,704	7,013	3,414	910	464
androidx	annotation	139,933	20,593	86,212	13,811	599	401
kotlin contracts	others	-	1	-	1	-	1

H LIST OF CONDITIONAL RUNTIME EXCEPTIONS ANALYZED

Table 2. List of exceptions analyzed in the CRE category.

CREs Constructs	
AndroidRuntimeException	MissingResourceException
ArithmeticException	NegativeArraySizeException
ArrayStoreException	NoSuchElementException
ArrayStoreException	NullPointerException
BufferOverflowException	ParcelFormatException
BufferUnderflowException	ParseException
ClassCastException	ProviderException
CompletionException	ProviderNotFoundException
ConcurrentModificationException	RejectedExecutionException
DOMException	SQLException
DateTimeException	SecurityException
EmptyStackException	TypeNotPresentException
EnumConstantNotPresentException	UncheckedIOException
FileSystemAlreadyExistsException	UndeclaredThrowableException
FileSystemNotFoundException	UnsupportedOperationException
IllegalArgumentException	WrongMethodTypeException
IllegalMonitorStateException	AcceptPendingException

IllegalStateException	AccessControlException
IncompleteAnnotationException	AlreadyBoundException
IndexOutOfBoundsException	AlreadyConnectedException
LSException	ArrayIndexOutOfBoundsException
MalformedParameterizedTypeException	BadParceableException
MalformedParametersException	CancellationException
UnsupportedAddressTypeException	UnsupportedCharsetException
WritePendingException	ZoneRulesException
CancelledKeyException	PatternSyntaxException
ClosedDirectoryStreamException	StringIndexOutOfBoundsException
ClosedFileSystemException	ReadOnlyBufferException
ClosedFileSystemException	ReadOnlyFileSystemException
ClosedSelectorException	ReadPendingException
ClosedWatchServiceException	ShutdownChannelGroupException
ConnectionPendingException	StringIndexOutOfBoundsException
NonReadableChannelException	UnknownFormatConversionException
NonWritableChannelException	UnknownFormatFlagsException
NotYetBoundException	UnresolvedAddressException
NotYetConnectedException	UnsupportedTemporalTypeException
NumberFormatException	OverlappingFileLockException

I LIST OF API'S METHODS ANALYZED

Table 3. List of the methods analyzed from each API.

Annotations analyzed	
Apache lang2 Validate	allElementsOfType() isTrue() noNullElements() notEmpty() notNull()
Apache lang3 Validate	allElementsOfType() exclusiveBetween() inclusiveBetween() assignableFrom()

	isInstanceOf()
	matchesPattern()
	notBlank()
	validIndex()
	validState()
Guava Preconditions	
	checkArgument()
	checkState()
	checkElementIndex()
	checkPositionIndex()
	checkNotNull()
	checkPositionIndexes()
Spring Assert	doesNotContain()
	hasLength()
	hasText()
	notEmpty()
	noNullElements()
	isInstanceOf()
	isAssignable()
	state()
	isNull()
	isTrue()
	notNull()

J LIST OF ANNOTATIONS ANALYZED

Table 4. List of annotations analyzed per package.

Annotations analyzed	
@CheckForNull	@CheckForSigned
@MatchesPattern	@Nonnegative
@Nonnull	@Nullable
@OverridingMethodsMustInvokeSupper	@ParametersAreNonnullByDefault
JSR305	

	@RegEx	@Signed
	@Syntax	@Syntax
	@Tainted	@Untainted
	@WillClose	@WillCloseWhenClosed
	@WillNotClose	@Guardedby
	@Immutable	@NotThreadSafe
	@ThreadSafe	
JSR303, JSR349	@Null	@DecimalMin
	@NotNull	@Size
	@AssertTrue	@Digits
	@AssertFalse	@Past
	@Min	@Future
	@Max	@Pattern
	@DecimalMax	
JetBrain		
	@Contract	@NotNull
	@Nullable	@PropertyKey
	@TestOnly	
IntelliJ	@BoxLayoutAxis	@CalendarMonth
	@CursorType	@FlowLayoutAlignment
	@FontStyle	@HorizontalAlignment
	@InputEventMask	@ListSelectionMode
	@PatternFlags	@TabLayoutPolicy
	@AdjustableOrientation	@Flow
	@Identifier	@TabPlacement
	@TitledBorderJustification	@TitledBorderTitlePosition
	@Language	@MagicConstant
	@Pattern	@PrintFormat
	@PrintFormat	@RexExp
	@Subst	
FindBugs	@CheckForNull	@NonNull
	@Nullable	@PossiblyNull

	@FontStyle	@HorizontalAlignment
	@UnkownNullness	@CreateObligation
	@DischargesObligation	@CleanupObligation
Android	@AndroidSupressLint	@AndroidTargetApi
Androidx		
	@AnimatorRes	@AnimRes
	@AnyRes	@AnyThread
	@AnyThread	@ArrayRes
	@AttrRes	@BinderThread
	@BinderThread	@BoolRes
	@CallSuper	@CheckResult
	@ChecksSdkIntAtLeast	@ColorInt
	@ColorLong	@ColorRes
	@ContentView	@DimenRes
	@Dimension	@NotInline
	@DrawableRes	@FloatRange
	@FloatRange	@FontRes
	@FontRes	@FractionRes
	@FractionRes	@GuardedBy
	@GuardedBy	@HalfFloat
	@IdRes	@InspectableProperty
	@IntDef	@IntegerRes
	@InterpolatorRes	@IntRange
	@Keep	@LayoutRes
	@LongDef	@MainThread
	@MainThread	@MenuRes
	@NavigationRes	@NonNull
	@Nullable	@PluralsRec
	@Px	@RawRes
	@RequiresApi	@RequiresFeature
	@RequiresPermission	@RestrictTo
	@Size	@StringDef
	@StringRes	@StyleableRes
	@StyleRes	@TransitionRes
	@UiThread	@VisibleForTesting
	@WorkerThread	@XmlRes

K USER STUDY

In the next page we present the survey used in the user study.

Design by Contract Survey

Start of Block: Technical Background

experience_software How many years of experience do you have in software development?

- ○ I don’t have any experience in software development (6)
- ○ <1 year (1)
- ○ 1-3 years (2)
- ○ 4-6 years (3)
- ○ 7-10 years (4)
- ○ >10 years (5)

experience_android How many years of experience do you have in Android Software development?

- ○ I don’t have any experience in Android software development (6)
- ○ <1 year (1)
- ○ 1-3 years (2)
- ○ 4-6 years (3)
- ○ 7-10 years (4)
- ○ >10 years (5)

End of Block: Technical Background

Start of Block: Defining Contract

Q8 **Design by Contract** is a technique in which software systems are seen as components that interact amongst themselves based on precisely defined specifications of client-supplier obligations (contracts). Contracts are specifications of an agreement between the client and the supplier of a component, where the supplier expects that certain conditions are met by the client before using the component (preconditions), maintains certain properties from entry to the component to exit (invariants), and guarantees that certain properties are met upon exit (postconditions). These contracts can be written as assertions directly into the code. For example, a way of enforcing a precondition in Java using exceptions might be: public void proceedWithCheckout (List < Item> shoppingCart) { if (shoppingCart.isEmpty ()) { throw new IllegalArgumentException () ; } ... } Other examples include annotations such as @NonNull , which can be used to express preconditions. In Java and Kotlin, the assert keyword can be used to enforce the validity of a condition (for example, an invariant). APIs such as org.apache.commons.lang.Validate.* or com.google.common.base.Preconditions.* are also used to denote contracts. Finally, Kotlin offers features such as @ExperimentalContracts that allow the developer to state a method's behavior to the compiler explicitly.

confidence How confident are you that you understand the definition of contract?

- ○ Very confident (1)
- ○ Somewhat Confident (2)
- ○ Not Confident at all (3)

frequency_of_use How often do you use contracts?

- ○ Never (1)
- ○ Sometimes (2)
- ○ Every time I code (3)
- ○ I'm not sure (4) _____

End of Block: Defining Contract

Start of Block: Follow-up-if-never

reason-for-not-using Why do you not use contracts?

_____

_____

_____

_____

_____

End of Block: Follow-up-if-never

Start of Block: Follow-up-if-sometimes

language-contracts Which programming languages do you primarily use to implement Design by Contract principles in your projects? Please select all that apply

- ☐ Eiffel (native support for DbC) (1)
- ☐ Ada (SPARK – with DbC features) (4)
- ☐ Java (using JML or contracts libraries such as CoFoJa) (5)
- ☐ C# (using Code Contracts) (6)
- ☐ Python (using assertions or contracts libraries such as PyContracts) (7)
- ☐ C++ (using assertions or contracts libraries) (8)
- ☐ Ruby (using contracts gem) (9)
- ☐ Dafny (designed with built-in support for formal verification and DbC) (11)
- ☐ Other (please specify) (10)

_____

reason-for-using Why do you use contracts?

_____

challenges What challenges, if any, did you encounter when using contracts?

_____

End of Block: Follow-up-if-sometimes

Start of Block: user-recomendations

user-recomend1 Please suggest some recommendations that can help improve the adoption of contracts.

_____

_____

_____

_____

_____

End of Block: user-recomendations

Start of Block: our-recommendations

recommendations-rank Please rank these recommendations:

	Very Important (1)	Somewhat Important (2)	Not Important at All (3)	Not sure (4)
Extend Java and Kotlin standard libraries with specialized constructs to specify contracts and with proper official documentation (1)	○	○	○	○
Have libraries that standardize contract specifications in Java and Kotlin (2)	○	○	○	○
Integrate into IDEs contract suggestion features supported by tools for by tools that automatically generate assertions and contracts (3)	○	○	○	○
IDE and continuous integration plugins to automatically detect contract violations (4)	○	○	○	○

user-recomend2 After reading these recommendations, what other recommendations can you think of?

_____

End of Block: our-recommendations

Start of Block: demographic

Q198 In the **last part** of the survey, we want to learn more about your background. You will also have an opportunity to give us feedback at the end.

age How old are you?

- Under 18 (1)
- 18-24 years old (2)
- 25-34 years old (3)
- 35-44 years old (4)
- 45-54 years old (5)
- 55-64 years old (6)
- 65+ years old (7)

gender How do you describe yourself?

- Male (1)
- Female (2)
- Non-binary / third gender (3)
- Prefer to self-describe (4) _____
- Prefer not to say (5)

education What is the highest level of education you have completed?

- ○ Some high school or less (1)
- ○ High school diploma or GED (2)
- ○ Some college, but no degree (3)
- ○ Associates or technical degree (4)
- ○ Bachelor's degree (5)
- ○ Graduate or professional degree (MA, MS, MBA, PhD, JD, MD, DDS etc.) (6)
- ○ Prefer not to say (7)

race Choose one or more races that you consider yourself to be

- ☐ White or Caucasian (1)
- ☐ Black or African American (2)
- ☐ American Indian/Native American or Alaska Native (3)
- ☐ Asian (4)
- ☐ Native Hawaiian or Other Pacific Islander (5)
- ☐ Other (6)
- ☐ Prefer not to say (7)

programming_geral Which of the following programming languages do you regularly use for your projects? Please select all that apply.

- [ ] C (1)
- [ ] C++ (4)
- [ ] Java (5)
- [ ] Python (6)
- [ ] JavaScript (7)
- [ ] TypeScript (8)
- [ ] Ruby (9)
- [ ] PHP (10)
- [ ] Swift (11)
- [ ] Kotlin (12)
- [ ] Go (13)
- [ ] Rust (14)
- [ ] Dafny (16)
- [ ] Other (please specify) (15)

Computing-M Are you employed in a computing field (e.g., IT, software engineer, programmer)?

- ○ Yes (1)
- ○ No (2)
- ○ No, but I have been in the past (3)
- ○ I'm not sure/Other (4) _____

ComputingEd-M Do you have formal education in a computing field (e.g., degree in computer science or computer engineering)?

- ○ Yes (1)
- ○ No (2)
- ○ I'm not sure/Other (4) _____

Page Break

other_feedback Do you have any other feedback for us? Feel free to share your thoughts about anything, from our survey to Design by Contract.

_____

_____

_____

_____

_____

End of Block: demographic