“This API is !@#$ing unusable.

When APIs are Intentionally Bypassed: An Exploratory Study of API Workarounds
Lamothe and Shang, ICSE 2020

Why do developers bypass APIs?

- implement functionality not directly supported by an API
- work around a bug in an API
- maintain version parity between an API and their own system

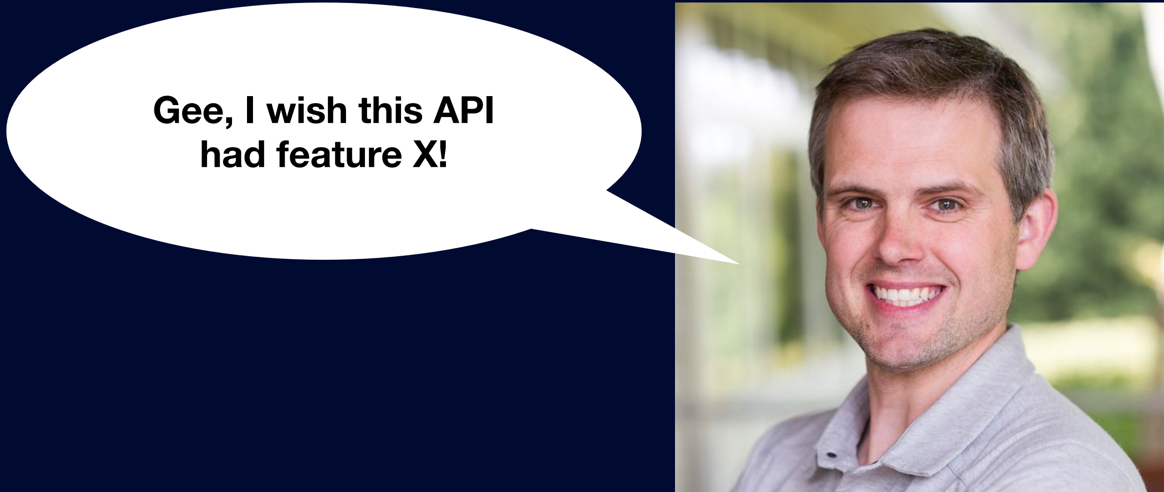

Table 3: Categories of answers derived from Stack Overflow posts on API workarounds

Answer Type	Quote	Frequency
Already supported by the API		**111**
Change your current implementation	“I have found the solution. I switched the direction of this mapping”	9
Use API as is	“As others have said, there’s nothing wrong with using [...]”	24
You will need small adjustments	“The simplest way is to just append [...] to the end of your command”	78
Not currently supported by the API		**260**
Need to implement a workaround	“One possible workaround is to write "setter/getter-like" methods, that uses a singleton to save the variables [...] or — of course — write a custom class [...]”	107
Not supported/No solution	“The reason you can’t do this is because it is specifically forbidden in the C♯ language specification”	38
Not recommended	“This is asserted as "by design" [...]. Consider a post-processing step that hacks the paths the way you want them.”	3
Use another API	“Do you absolutely have to use java.util.Date? I would thoroughly recommend that you use Joda Time or the java.time package from Java 8 instead.”	80
Wait for/apply new version/patch	“I think you are experiencing a likely symptom of [...]. This bug exists in 3.2 and higher and was only fixed recently (4.2).”	32
User is confused	“Don’t hack something together using JavaScript, as soon as Twitter makes an update to their widget, that’s it, you’re screwed. Use a server-side language and do it properly as per their documentation.”	**13**
Unusable		**73**
No answer		20
Useless (Unrelated to API workarounds)		53

Workaround patterns

- *Functionality extension*
  - add new features to an existing API
  - override existing behaviour

- *Deep copy*
  - keep a copy of data obtained from an API
  - don’t query the API directly, use the copy

- *Multi-version*
  - developers want functionality that’s found in disparate API versions
  - use two (or more) versions of the same API

Functionality extension

```
public static class PortofinoGetGeneratedKeysDelegate extends GetGeneratedKeysDelegate implements
    InsertGeneratedIdentifierDelegate {
  ...
  //Override in order to unquote the primary key column name, else it breaks (at least on Postgres)
  @Override
  public Serializable executeAndExtract(PreparedStatement insert, SessionImplementor session) throws SQLException {
    session.getTransactionCoordinator().getJdbcCoordinator().getResultSetReturn()
    .executeUpdate(insert);
    ResultSet rs = null;
    try {
      rs = insert.getGeneratedKeys();
      return IdentifierGeneratorHelper.getGeneratedIdentity(rs,
          unquotedIdentifier(persister.getRootTableKeyColumnNames()[0]),
          persister.getIdentifierType()
        );
      } finally {
      if (rs != null) {
        session.getTransactionCoordinator().getJdbcCoordinator().release(rs, insert);
      }
    }
  }
  protected String unquotedIdentifier(String identifier) { //This is a hack.
    if(identifier.startsWith(dialect.openQuote() + "")) {
      return identifier.substring(1, identifier.length() - 1);
    }
    return identifier;
  }
}
```

Deep copy

```
@Override
public String getText(){
    JsonToken t = _currToken;
    ...
    return _getText2(t);
}
protected String _getText2(JsonToken t){
    ...
    switch (t) {
        case FIELD_NAME:
            return _parsingContext.getCurrentName();
        case VALUE_NUMBER_FLOAT:
            return _textBuffer.contentsAsString();
    }
    return t.asString();
}
```

Keep interface the same, but modify existing information

Multi-version

```
private Set<String> resolveProperties(Map<String, PropertyContext> pCon){
    ...
    classLoader = getLoader(getStringOrArrayLitteral(pCon.beanValue()));
    URI log4JUri = null;
    if (pCon.containsKey("log4j.configURL")) {
        log4JUri = getLitteral(pCon.beanValue(), i->i.stringLiteral().getText(), j->{})
    }
    if (log4JUri != null) {
        InputStream is = log4JUri.toURL().openStream());
        int toread = 0;
        while ((toread = is.available()) != 0) {is.skip(toread);}
        LoggerContext ctx = (LoggerContext) LogManager.getContext(classLoader, true);
        ctx.setConfigLocation(log4JUri);
    }
    ...
}
```

Url for Log4j config

Dynamically set/load Log4j config rather than use current Log4j2 config

Discussion

- Have you ever bypassed an API?
  - did it fit into the 3 patterns discussed by Lamothe and Shang?
- What do you think about the suggestions that Lamothe and Shang provide for API developers?
- Do you agree with what the authors discussed?
- What did you like/dislike about this paper?
- Was there anything new/surprising you found?