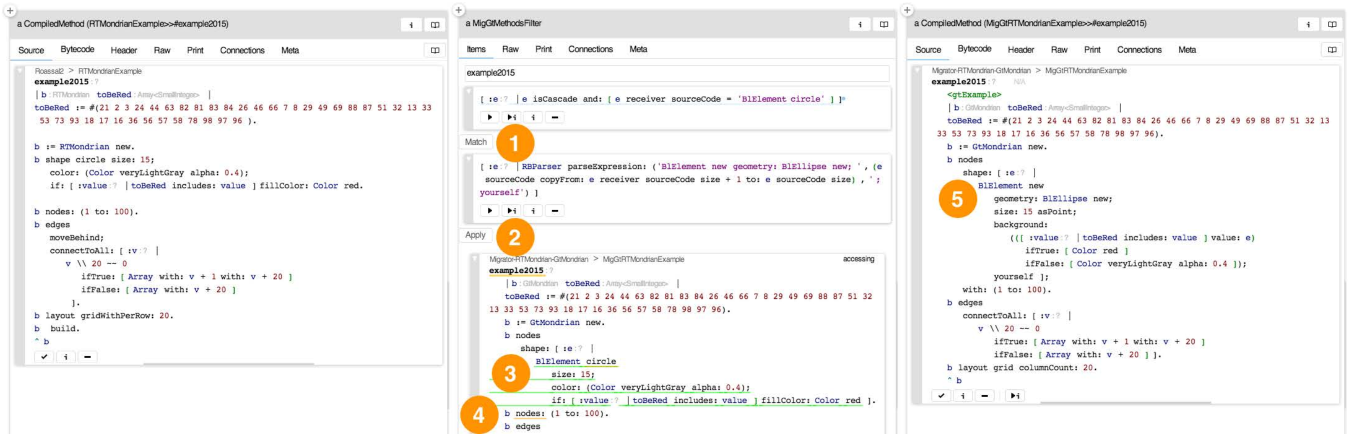
Exploring Example-Driven Migration

Manuel Leuenberger
Software Composition Group
University of Bern
Bern, Switzerland
manuel.leuenberger@inf.unibe.ch


a CompiledMethod (RTMondrianExample>>#example2015)
Source
Bytecode
Header
Raw
Print
Connections
Meta
Roassal2 > RTMondrianExample

```
example2015
| b : RTMondrian  toBeRed : Array<SmallInteger> |
toBeRed := #(21 2 3 24 44 63 82 81 83 84 26 46 66 7 8 29 49 69 88 87 51 32 13 33
 53 73 93 18 17 16 36 56 57 58 78 98 97 96 ).

b := RTMondrian new.
b shape circle size: 15;
   color: (Color veryLightGray alpha: 0.4);
   if: [ :value | toBeRed includes: value ] fillColor: Color red.

b nodes: (1 to: 100).
b edges
   moveBehind;
   connectToAll: [ :v |
      v \\ 20 ~~ 0
         ifTrue: [ Array with: v + 1 with: v + 20 ]
         ifFalse: [ Array with: v + 20 ]
      ].
b layout gridWithPerRow: 20.
b  build.
^ b
```

```
[ :e | e isCascade and: [ e receiver sourceCode = 'BlElement circle' ] ]
```

```
[ :e | RBParser parseExpression: ('BlElement new geometry: BlEllipse new; ' , (e
 sourceCode copyFrom: e receiver sourceCode size + 1 to: e sourceCode size) , ';
yourself') ]
```

```
example2015
   | b : GtMondrian  toBeRed : Array<SmallInteger> |
   toBeRed := #(21 2 3 24 44 63 82 81 83 84 26 46 66 7 8 29 49 69 88 87 51 32
13 33 53 73 93 18 17 16 36 56 57 58 78 98 97 96).
   b := GtMondrian new.
   b nodes
      shape: [ :e |
         BlElement circle
            size: 15;
            color: (Color veryLightGray alpha: 0.4);
            if: [ :value | toBeRed includes: value ] fillColor: Color red ].
   b nodes: (1 to: 100).
   b edges
```

a CompiledMethod (MigGtRTMondrianExample>>#example2015)
Source
Bytecode
Header
Raw
Print
Connections
Meta
Migrator-RTMondrian-GtMondrian > MigGtRTMondrianExample
N/A
5

```
example2015
   <gtExample>
   | b : GtMondrian  toBeRed : Array<SmallInteger> |
   toBeRed := #(21 2 3 24 44 63 82 81 83 84 26 46 66 7 8 29 49 69 88 87 51 32 13
 33 53 73 93 18 17 16 36 56 57 58 78 98 97 96).
   b := GtMondrian new.
   b nodes
      shape: [ :e |
         BlElement new
            geometry: BlEllipse new;
            size: 15 asPoint;
            background:
               (([ :value | toBeRed includes: value ] value: e)
                  ifTrue: [ Color red ]
                  ifFalse: [ Color veryLightGray alpha: 0.4 ]);
            yourself ];
      with: (1 to: 100).
   b edges
      connectToAll: [ :v |
         v \\ 20 ~~ 0
            ifTrue: [ Array with: v + 1 with: v + 20 ]
            ifFalse: [ Array with: v + 20 ] ].
   b layout grid columnCount: 20.
   ^ b
```

Figure 1: An example before migration (left), during the migration in the workbench (middle), and after the migration (right). A match (1) and a transformation (2) define a refactoring. Matches (3) and missing messages (4) are highlighted. Refactorings can be arbitrarily complex, *e.g.*, compare (3) and (5).

ABSTRACT

Despite many research efforts to automate API usage migration, it remains often a manual task for developers. We aim to reduce the developer's pain by exploring ways to integrate the migration process into the IDE. Our migration workbench leverages API usage examples and interactive refactorings to migrate code from one API version to the other.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**; *Patterns.*

KEYWORDS

API migration, refactoring, IDE, API usage

ACM Reference Format:
Manuel Leuenberger. 2019. Exploring Example-Driven Migration. In *Companion of the 3rd International Conference on Art, Science, and Engineering of Programming (Programming '19), April 1–4, 2019, Genova, Italy.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3328433.3328463

1 INTRODUCTION

Replacing an API with a different version requires developers to migrate their code from the old API to the new one, a task that developers are often reluctant to perform [1, 7, 10]. Automatic migration has been an ongoing subject of study in research [4, 6], yet the impact on developers has been limited [8]. IDEs still support only basic refactorings [5, 9], and libraries provide migration guides that require the developer to build an individual migration path.

We claim that the lack of tools supporting the migration process in the IDE is partly due to refactorings being non-trivial to create, predict, and inspect. With our migration workbench we aim at turning API migration into an *integrated* and transparent process by focussing on the following aspects:

(1) enable navigation and search within examples of the old and new APIs
(2) allow developers to define and inspect refactorings interactively
(3) make refactorings shareable to benefit other users of an API

In this work we explore the first two aforementioned points, while the third point is left as future work.

2 EXAMPLE-DRIVEN MIGRATION

Our workbench[1] is grounded on examples as they are highly valued by developers as a form of documentation [2], and they provide a cohesive view on how to use an API. In this section we motivate our approach through a concrete scenario of how examples and interactive refactorings enable the migration between two different versions of an API.

Source & Target API. Our source API is `RTMondrian`, the target API is `GtMondrian`. Mondrian is a graph-based visualization engine. Nodes and edges can be decorated, *e.g.*, by using shapes and colors, and they are interactive, *i.e.*, clicking on a visual element inspects the underlying domain object. The two versions differ slightly, *e.g.*, they use different rendering engines, and their builder DSLs use different messages and require different message sequences (Figure 1.4/5). Both versions provide examples to showcase different visualization aspects.

Navigation & Search. We can browse and search examples of both API versions within our workbench. For example, to find a replacement for `RTMondrian >> #nodes:`, which does not exist in `GtMondrian` (Figure 1.4), a keyboard shortcut searches the `GtMondrian` examples to find uses of similar messages. The matching examples in the target API then serve as a baseline to create a refactoring.

Refactorings. The workbench allows the developer to define refactorings as AST transformations. The matcher (Figure 1.1) selects the nodes to be migrated. The transformer (Figure 1.2) transforms and replaces a matched node. Matched nodes are highlighted in the method source (Figure 1.3) giving the developer immediate feedback on the region affected by the refactoring.

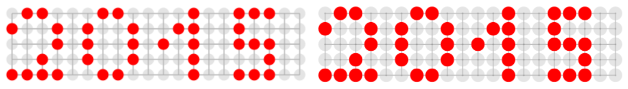


Figure 2: Rendering of the original `RTMondrian` example (left) and of the migrated `GtMondrian` example (right).

[1]https://github.com/maenu/migrator-rtmondrian-gtmondrian

Results. The workbench is implemented on top of the moldable inspector [3]. We use the workbench to create the 11 refactorings to migrate the example in Figure 1. Figure 2 shows the rendered output of the original and migrated example. We notice a difference in the grid spacing as the only difference between the two renderings.

In our current implementation we use Pharo code to define AST transformations. We took this decision as the Pharo's refactoring browser pattern language [9] turned out not to be powerful enough to express the refactorings we required for migrating the presented example. Expressing the matcher for the refactoring of the `#color:` and `#if:fillColor:` messages to `#background:` as `b shape circle size: ``s; color: ``c1; if: `` ↪ `b fillColor: ``c2` would not match, due to the pattern language's limitations on matching sequenced and cascaded message patterns. Using Pharo code to define define AST transformations is rather verbose, a declarative refactoring language that supports more general AST transformations is needed.

3 CONCLUSIONS AND FUTURE WORK

Using the migration workbench we are able to derive and define appropriate refactorings to migrate an example using `RTMondrian` to an example using `GtMondrian`. The combination of navigation and user-defined refactorings in the migration workbench serves as a powerful and *integrated* approach to API migration.

The format in which to define and share refactorings remains an open question nevertheless. Our future focus will move from migrating methods independently to systems as a whole, supported by a more expressive refactoring language, and refactoring proposals mined from examples.

ACKNOWLEDGMENTS

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Agile Software Assistance" (SNSF project No. 200020-181973, Feb 1, 2019 - Apr 30, 2022).

REFERENCES

[1] Gabriele Bavota, Gerardo Canfora, Massimiliano D. Penta, Rocco Oliveto, and Sebastiano Panichella. 2013. The Evolution of Project Inter-dependencies in a Software Ecosystem: The Case of Apache. In *2013 IEEE International Conference on Software Maintenance*. 280–289. https://doi.org/10.1109/ICSM.2013.39

[2] Raymond P. L. Buse and Westley Weimer. 2012. Synthesizing API Usage Examples. In *Proceedings of the 34th International Conference on Software Engineering (ICSE ’12)*. IEEE Press, Piscataway, NJ, USA, 782–792. http://dl.acm.org/citation.cfm?id=2337223.2337316

[3] Andrei Chiş. 2016. *Moldable Tools*. PhD thesis. University of Bern. http://scg.unibe.ch/archive/phd/chis-phd.pdf

[4] Danny Dig and Ralph Johnson. 2006. How do APIs evolve? A story of refactoring. *Journal of Software Maintenance and Evolution: Research and Practice (JSME)* 18, 2 (April 2006), 83–107.

[5] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. 1999. *Refactoring: Improving the Design of Existing Code*. Addison Wesley.

[6] Andre Hora, Anne Etien, Nicolas Anquetil, Stéphane Ducasse, and Marco Túlio Valente. 2014. APIEvolutionMiner: Keeping API Evolution under Control. In *Proceedings of the Software Evolution Week (CSMR-WCRE’14)*. http://rmod.inria.fr/archives/papers/Hora14a-CSMR-WCRE-APIEvolutionMiner.pdf

[7] Raula Gaikovina Kula, Daniel M German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. 2017. Do developers update their library dependencies? *Empirical Software Engineering* (2017), 1–34.

[8] Huiqing Li and Simon Thompson. 2012. Automated API Migration in a User-extensible Refactoring Tool for Erlang Programs. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*. ACM, New York, NY, USA, 294–297. https://doi.org/10.1145/2351676.2351727

[9] Don Roberts, John Brant, and Ralph E. Johnson. 1997. A Refactoring Tool for Smalltalk. *Theory and Practice of Object Systems (TAPOS)* 3, 4 (1997), 253–263.

[10] Martin P. Robillard, Eric Bodden, David Kawrykow, Mira Mezini, and Tristan Ratchford. 2013. Automated API Property Inference Techniques. *Software Engineering, IEEE Transactions on* 39, 5 (2013), 613–637. https://doi.org/10.1109/TSE.2012.63