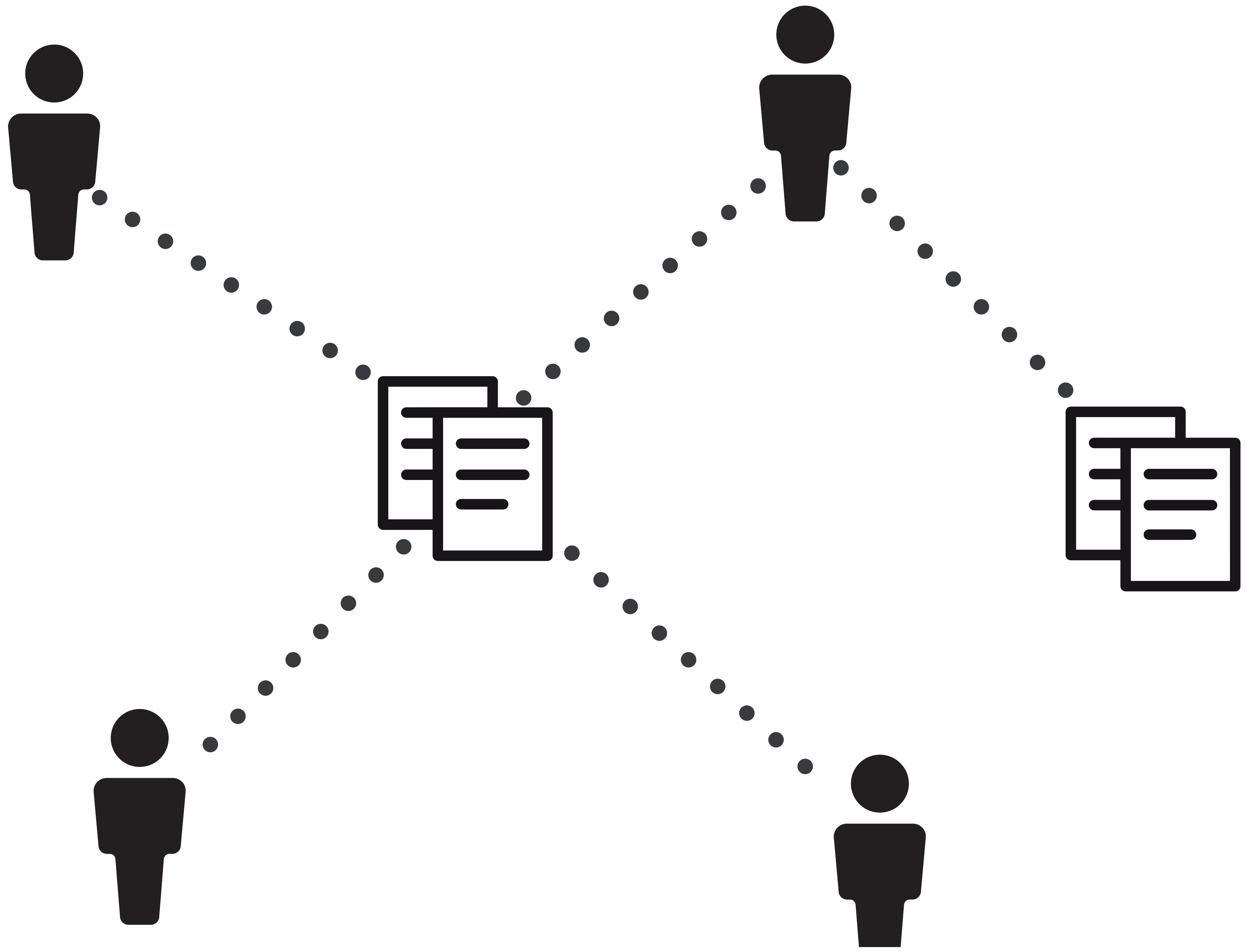
EggShell

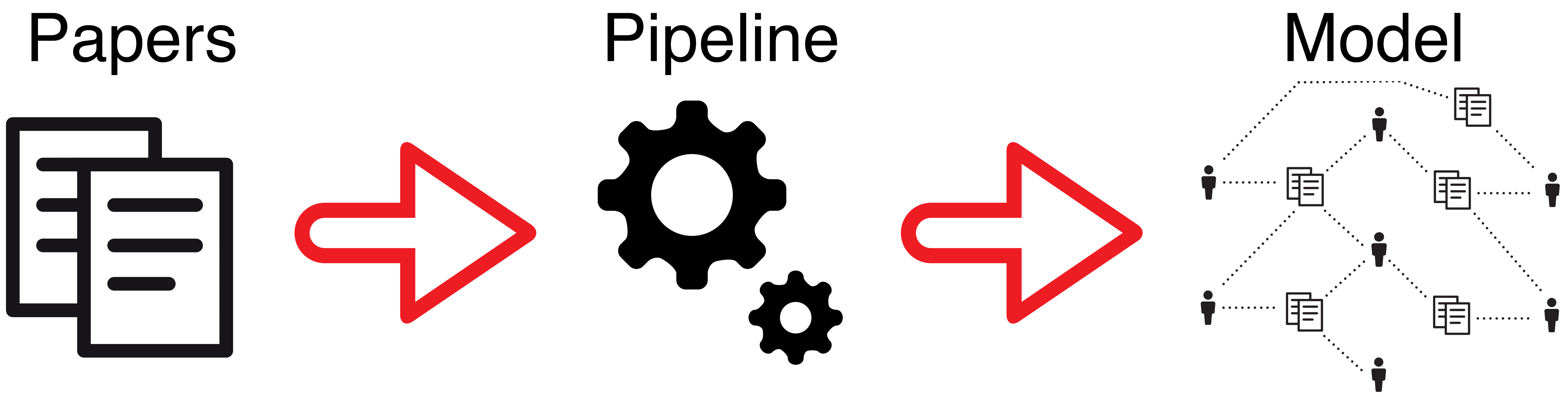
a workbench for the assessment of modeling pipelines for scientific communities

Dominik Seliner
selinerdominik@gmail.com
05.07.2016

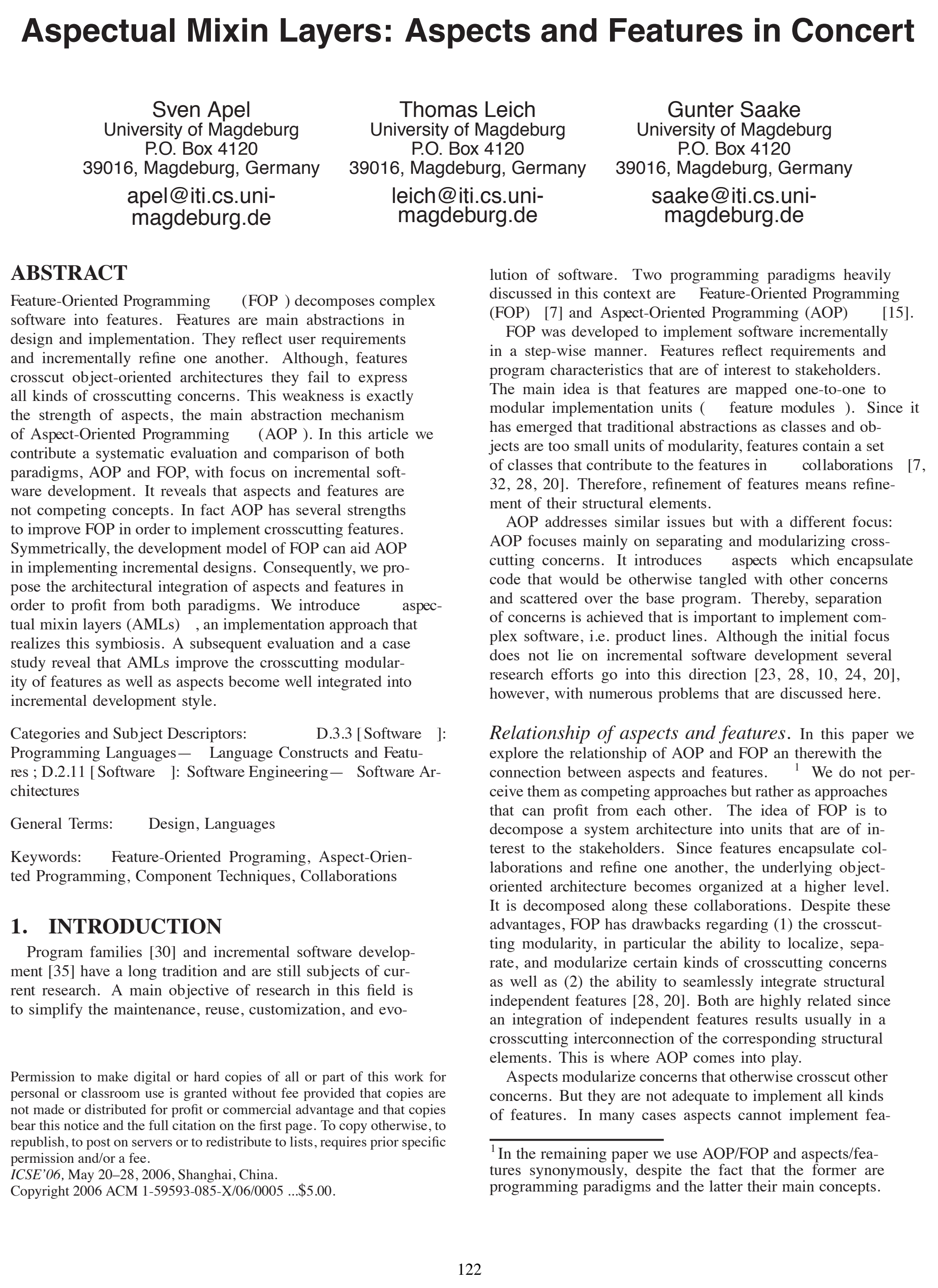
Motivation (1)



Motivation (2)



Motivation (3)



Aspectual Mixin Layers: Aspects and Features in Concert

Sven Apel
University of Magdeburg
P.O. Box 4120
39016, Magdeburg, Germany
apel@iti.cs.uni-magdeburg.de

Thomas Leich
University of Magdeburg
P.O. Box 4120
39016, Magdeburg, Germany
leich@iti.cs.uni-magdeburg.de

Gunter Saake
University of Magdeburg
P.O. Box 4120
39016, Magdeburg, Germany
saake@iti.cs.uni-magdeburg.de

ABSTRACT

Feature-Oriented Programming (FOP) decomposes complex software into features. Features are main abstractions in design and implementation. They reflect user requirements and incrementally refine one another. Although, features crosscut object-oriented architectures they fail to express all kinds of crosscutting concerns. This weakness is exactly the strength of aspects, the main abstraction mechanism of Aspect-Oriented Programming (AOP). In this article we contribute a systematic evaluation and comparison of both paradigms, AOP and FOP, with focus on incremental software development. It reveals that aspects and features are not competing concepts. In fact AOP has several strengths to improve FOP in order to implement crosscutting features. Symmetrically, the development model of FOP can aid AOP in implementing incremental designs. Consequently, we propose the architectural integration of aspects and features in order to profit from both paradigms. We introduce aspectual mixin layers (AMLs), an implementation approach that realizes this symbiosis. A subsequent evaluation and a case study reveal that AMLs improve the crosscutting modularity of features as well as aspects become well integrated into incremental development style.

Categories and Subject Descriptors: D.3.3 [Software]: Programming Languages— Language Constructs and Features; D.2.11 [Software]: Software Engineering— Software Architectures

General Terms: Design, Languages

Keywords: Feature-Oriented Programing, Aspect-Oriented Programming, Component Techniques, Collaborations

1. INTRODUCTION

Program families [30] and incremental software development [35] have a long tradition and are still subjects of current research. A main objective of research in this field is to simplify the maintenance, reuse, customization, and evolution of software. Two programming paradigms heavily discussed in this context are Feature-Oriented Programming (FOP) [7] and Aspect-Oriented Programming (AOP) [15].

FOP was developed to implement software incrementally in a step-wise manner. Features reflect requirements and program characteristics that are of interest to stakeholders. The main idea is that features are mapped one-to-one to modular implementation units (feature modules). Since it has emerged that traditional abstractions as classes and objects are too small units of modularity, features contain a set of classes that contribute to the features in collaborations [7, 32, 28, 20]. Therefore, refinement of features means refinement of their structural elements.

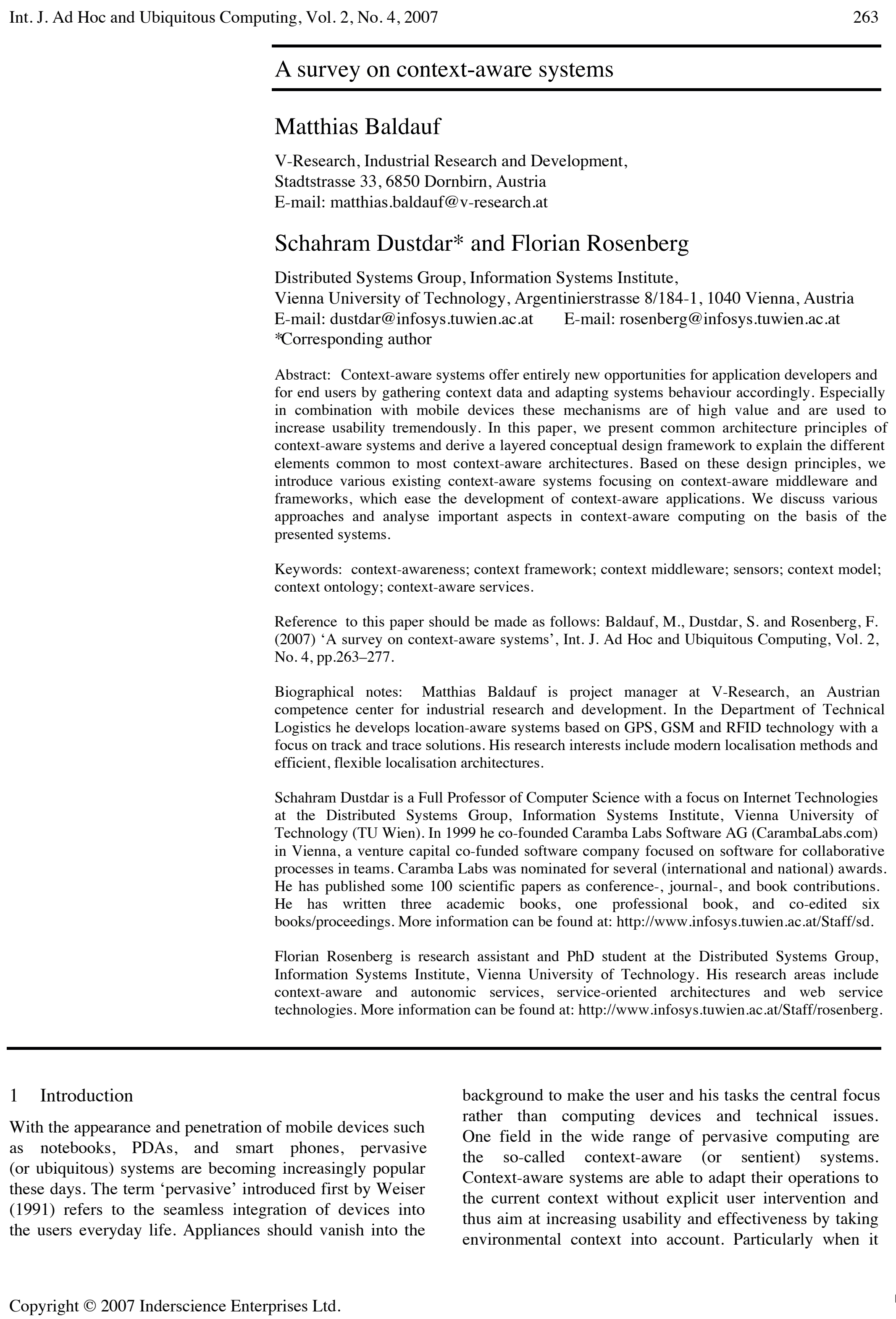
AOP addresses similar issues but with a different focus: AOP focuses mainly on separating and modularizing crosscutting concerns. It introduces aspects which encapsulate code that would be otherwise tangled with other concerns and scattered over the base program. Thereby, separation of concerns is achieved that is important to implement complex software, i.e. product lines. Although the initial focus does not lie on incremental software development several research efforts go into this direction [23, 28, 10, 24, 20], however, with numerous problems that are discussed here.

Relationship of aspects and features. In this paper we explore the relationship of AOP and FOP an therewith the connection between aspects and features.[1] We do not perceive them as competing approaches but rather as approaches that can profit from each other. The idea of FOP is to decompose a system architecture into units that are of interest to the stakeholders. Since features encapsulate collaborations and refine one another, the underlying object-oriented architecture becomes organized at a higher level. It is decomposed along these collaborations. Despite these advantages, FOP has drawbacks regarding (1) the crosscutting modularity, in particular the ability to localize, separate, and modularize certain kinds of crosscutting concerns as well as (2) the ability to seamlessly integrate structural independent features [28, 20]. Both are highly related since an integration of independent features results usually in a crosscutting interconnection of the corresponding structural elements. This is where AOP comes into play.

Aspects modularize concerns that otherwise crosscut other concerns. But they are not adequate to implement all kinds of features. In many cases aspects cannot implement fea-

[1] In the remaining paper we use AOP/FOP and aspects/features synonymously, despite the fact that the former are programming paradigms and the latter their main concepts.

122



Int. J. Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4, 2007 263

A survey on context-aware systems

Matthias Baldauf
V-Research, Industrial Research and Development,
Stadtstrasse 33, 6850 Dornbirn, Austria
E-mail: matthias.baldauf@v-research.at

Schahram Dustdar* and Florian Rosenberg
Distributed Systems Group, Information Systems Institute,
Vienna University of Technology, Argentinierstrasse 8/184-1, 1040 Vienna, Austria
E-mail: dustdar@infosys.tuwien.ac.at E-mail: rosenberg@infosys.tuwien.ac.at
*Corresponding author

Abstract: Context-aware systems offer entirely new opportunities for application developers and for end users by gathering context data and adapting systems behaviour accordingly. Especially in combination with mobile devices these mechanisms are of high value and are used to increase usability tremendously. In this paper, we present common architecture principles of context-aware systems and derive a layered conceptual design framework to explain the different elements common to most context-aware architectures. Based on these design principles, we introduce various existing context-aware systems focusing on context-aware middleware and frameworks, which ease the development of context-aware applications. We discuss various approaches and analyse important aspects in context-aware computing on the basis of the presented systems.

Keywords: context-awareness; context framework; context middleware; sensors; context model; context ontology; context-aware services.

Reference to this paper should be made as follows: Baldauf, M., Dustdar, S. and Rosenberg, F. (2007) 'A survey on context-aware systems', Int. J. Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4, pp.263–277.

Biographical notes: Matthias Baldauf is project manager at V-Research, an Austrian competence center for industrial research and development. In the Department of Technical Logistics he develops location-aware systems based on GPS, GSM and RFID technology with a focus on track and trace solutions. His research interests include modern localisation methods and efficient, flexible localisation architectures.

Schahram Dustdar is a Full Professor of Computer Science with a focus on Internet Technologies at the Distributed Systems Group, Information Systems Institute, Vienna University of Technology (TU Wien). In 1999 he co-founded Caramba Labs Software AG (CarambaLabs.com) in Vienna, a venture capital co-funded software company focused on software for collaborative processes in teams. Caramba Labs was nominated for several (international and national) awards. He has published some 100 scientific papers as conference-, journal-, and book contributions. He has written three academic books, one professional book, and co-edited six books/proceedings. More information can be found at: http://www.infosys.tuwien.ac.at/Staff/sd.

Florian Rosenberg is research assistant and PhD student at the Distributed Systems Group, Information Systems Institute, Vienna University of Technology. His research areas include context-aware and autonomic services, service-oriented architectures and web service technologies. More information can be found at: http://www.infosys.tuwien.ac.at/Staff/rosenberg.

1 Introduction

With the appearance and penetration of mobile devices such as notebooks, PDAs, and smart phones, pervasive (or ubiquitous) systems are becoming increasingly popular these days. The term 'pervasive' introduced first by Weiser (1991) refers to the seamless integration of devices into the users everyday life. Appliances should vanish into the background to make the user and his tasks the central focus rather than computing devices and technical issues. One field in the wide range of pervasive computing are the so-called context-aware (or sentient) systems. Context-aware systems are able to adapt their operations to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking environmental context into account. Particularly when it

Heuristic

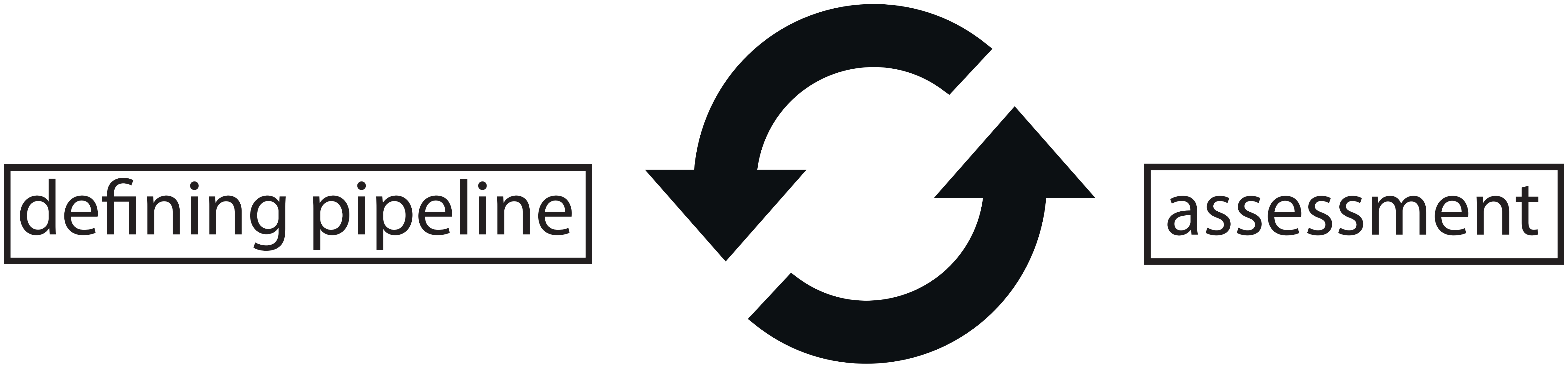
A heuristic is an approach to problem solving, lear-ning, or discovery that employs a practical method **not** guaranteed to be optimal or perfect, but **varies** in its accuracy depending on the data set at hand.

-Definition adapted from wikipedia

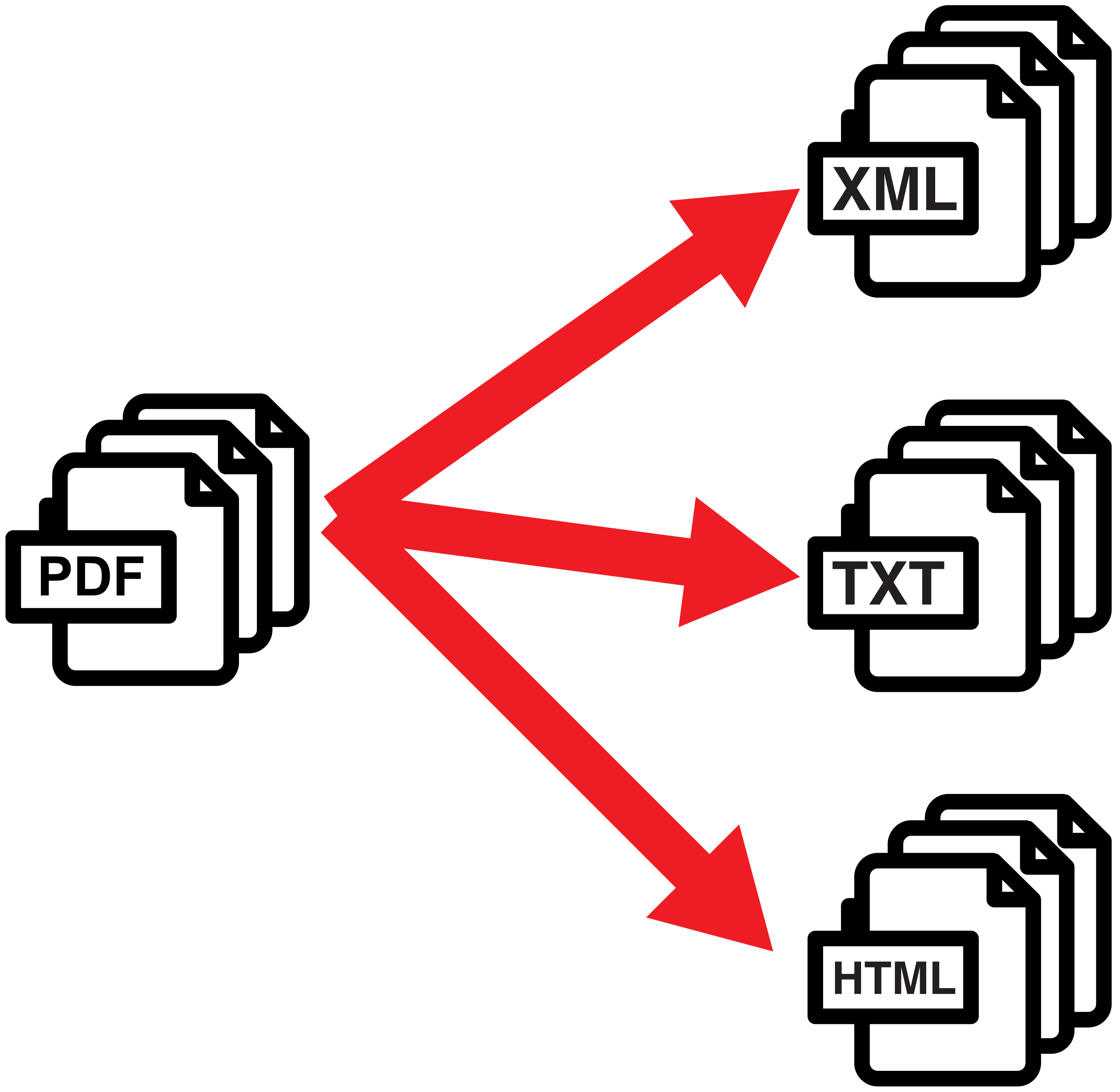
EggShell

a workbench for the assessment of modeling pipelines for scientific communities

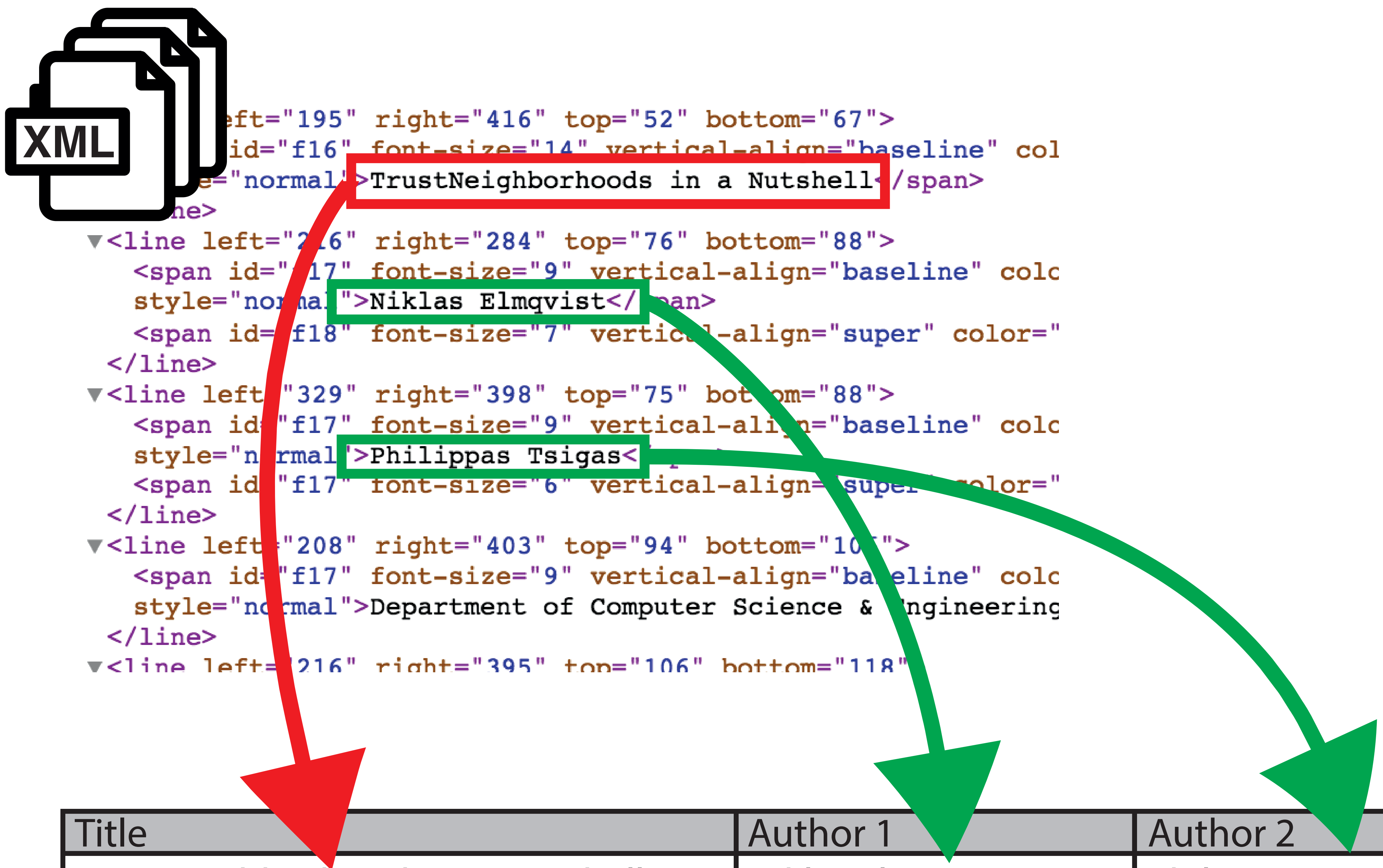
EggShell



Modeling-Pipeline (1)



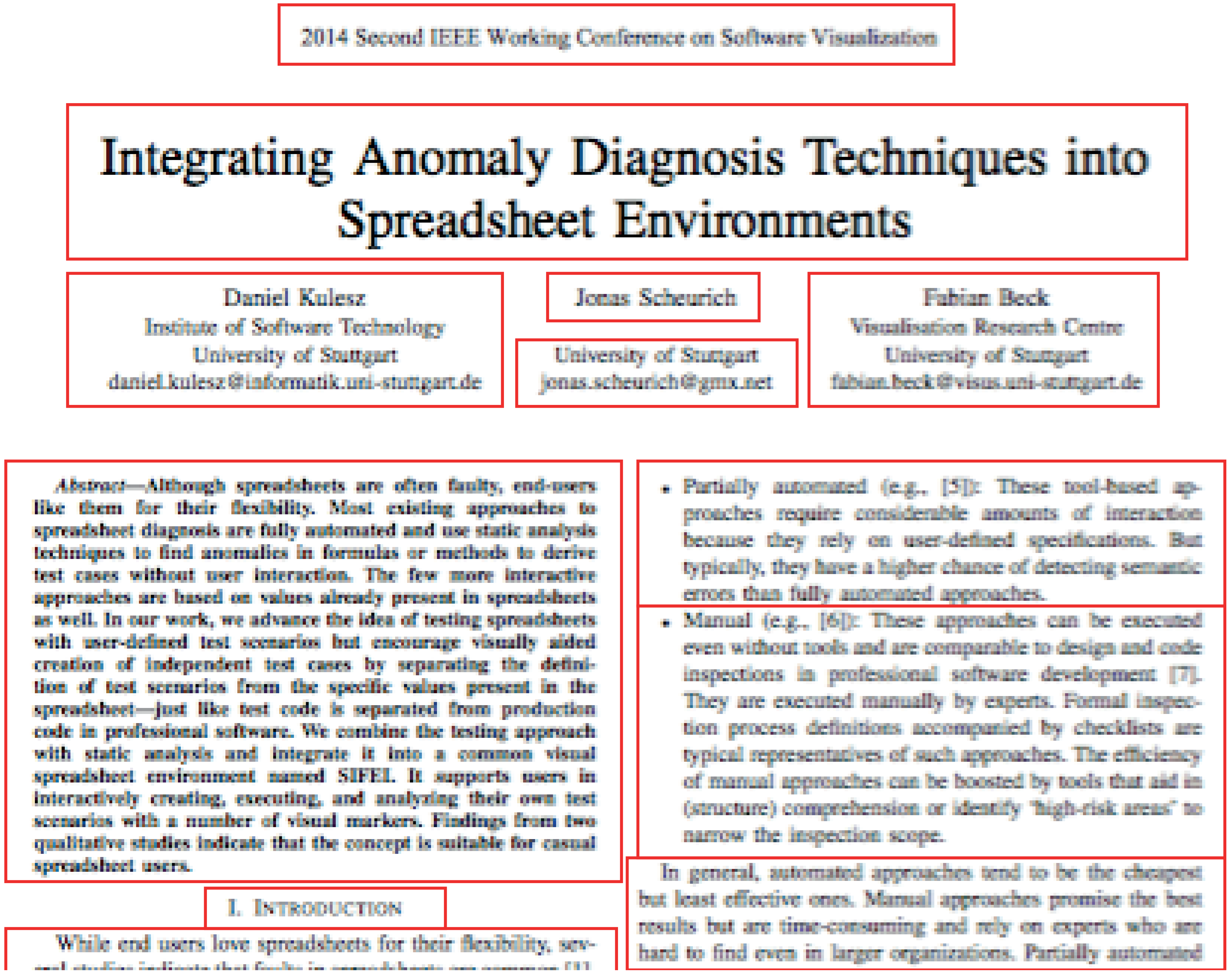
Modeling-Pipeline (2)



```
        eft="195" right="416" top="52" bottom="67">
        id="f16" font-size="14" vertical-align="baseline" col
      e="normal">TrustNeighborhoods in a Nutshell</span>
     ne>
▼<line left="216" right="284" top="76" bottom="88">
   <span id="f17" font-size="9" vertical-align="baseline" colo
   style="normal">Niklas Elmqvist</span>
   <span id="f18" font-size="7" vertical-align="super" color="
 </line>
▼<line left="329" right="398" top="75" bottom="88">
   <span id="f17" font-size="9" vertical-align="baseline" colo
   style="normal">Philippas Tsigas<
   <span id="f17" font-size="6" vertical-align="super" color="
 </line>
▼<line left="208" right="403" top="94" bottom="106">
   <span id="f17" font-size="9" vertical-align="baseline" colo
   style="normal">Department of Computer Science & Engineering
 </line>
▼<line left="216" right="395" top="106" bottom="118"
```

Title	Author 1	Author 2	...
TrustNeighbothoods in a Nutshell	Niklas Elmqvist	Philippas Tsigas	...
Feature-centric environment	David Röthlisberger	Orla Greevy	...
...	...	...	...

Modeling-Pipeline (3)



2014 Second IEEE Working Conference on Software Visualization

Integrating Anomaly Diagnosis Techniques into Spreadsheet Environments

Daniel Kulesz
Institute of Software Technology
University of Stuttgart
daniel.kulesz@informatik.uni-stuttgart.de

Jonas Scheurich
University of Stuttgart
jonas.scheurich@gmx.net

Fabian Beck
Visualisation Research Centre
University of Stuttgart
fabian.beck@visus.uni-stuttgart.de

Abstract—Although spreadsheets are often faulty, end-users like them for their flexibility. Most existing approaches to spreadsheet diagnosis are fully automated and use static analysis techniques to find anomalies in formulas or methods to derive test cases without user interaction. The few more interactive approaches are based on values already present in spreadsheets as well. In our work, we advance the idea of testing spreadsheets with user-defined test scenarios but encourage visually aided creation of independent test cases by separating the definition of test scenarios from the specific values present in the spreadsheet—just like test code is separated from production code in professional software. We combine the testing approach with static analysis and integrate it into a common visual spreadsheet environment named SIFEI. It supports users in interactively creating, executing, and analyzing their own test scenarios with a number of visual markers. Findings from two qualitative studies indicate that the concept is suitable for casual spreadsheet users.

I. Introduction

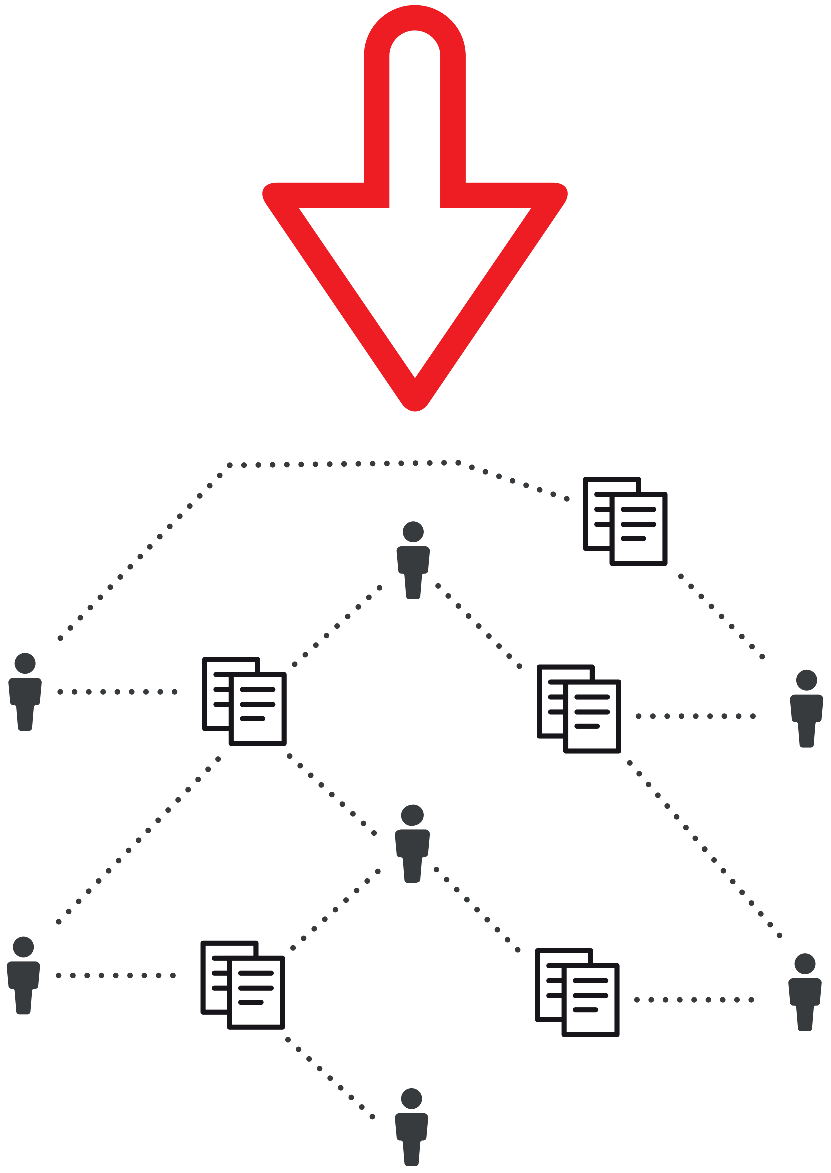
While end users love spreadsheets for their flexibility, several studies indicate that faults in spreadsheets are common [1]

- Partially automated (e.g., [5]): These tool-based approaches require considerable amounts of interaction because they rely on user-defined specifications. But typically, they have a higher chance of detecting semantic errors than fully automated approaches.
- Manual (e.g., [6]): These approaches can be executed even without tools and are comparable to design and code inspections in professional software development [7]. They are executed manually by experts. Formal inspection process definitions accompanied by checklists are typical representatives of such approaches. The efficiency of manual approaches can be boosted by tools that aid in (structure) comprehension or identify "high-risk areas" to narrow the inspection scope.

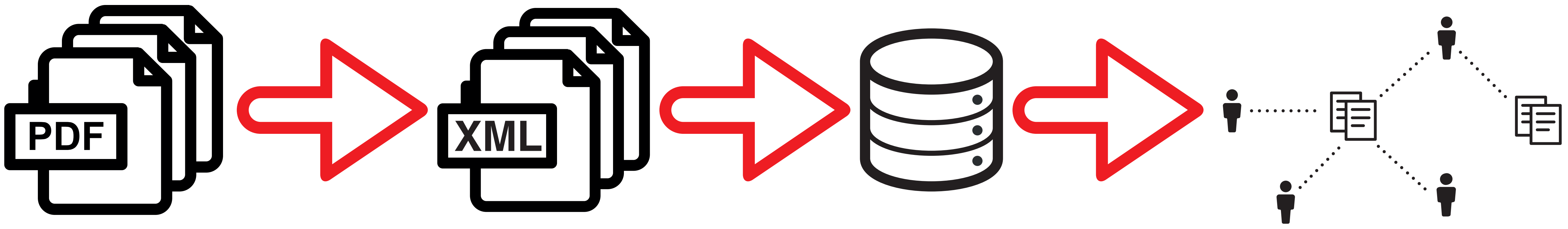
In general, automated approaches tend to be the cheapest but least effective ones. Manual approaches promise the best results but are time-consuming and rely on experts who are hard to find even in larger organizations. Partially automated

Modeling-Pipeline (4)

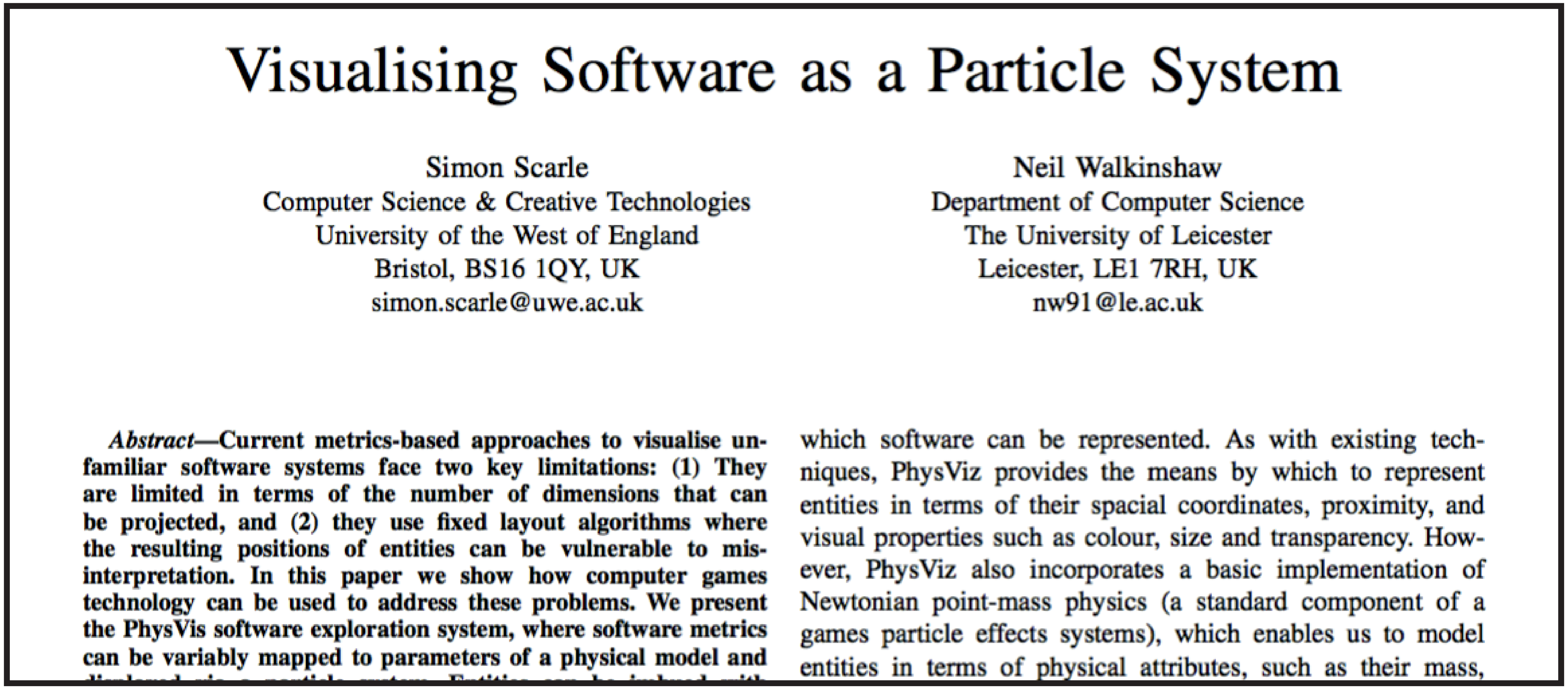
Title	Author 1	Author 2	...
TrustNeighbothoods in a Nutshell	Niklas Elmqvist	Philippas Tsigas	...
Feature-centric environment	David Röthlisberger	Orla Greevy	...
...	...	...	...



Modeling-Pipeline (5)



Questions about the accuracy of a model



Visualising Software as a Particle System

Simon Scarle
Computer Science & Creative Technologies
University of the West of England
Bristol, BS16 1QY, UK
simon.scarle@uwe.ac.uk

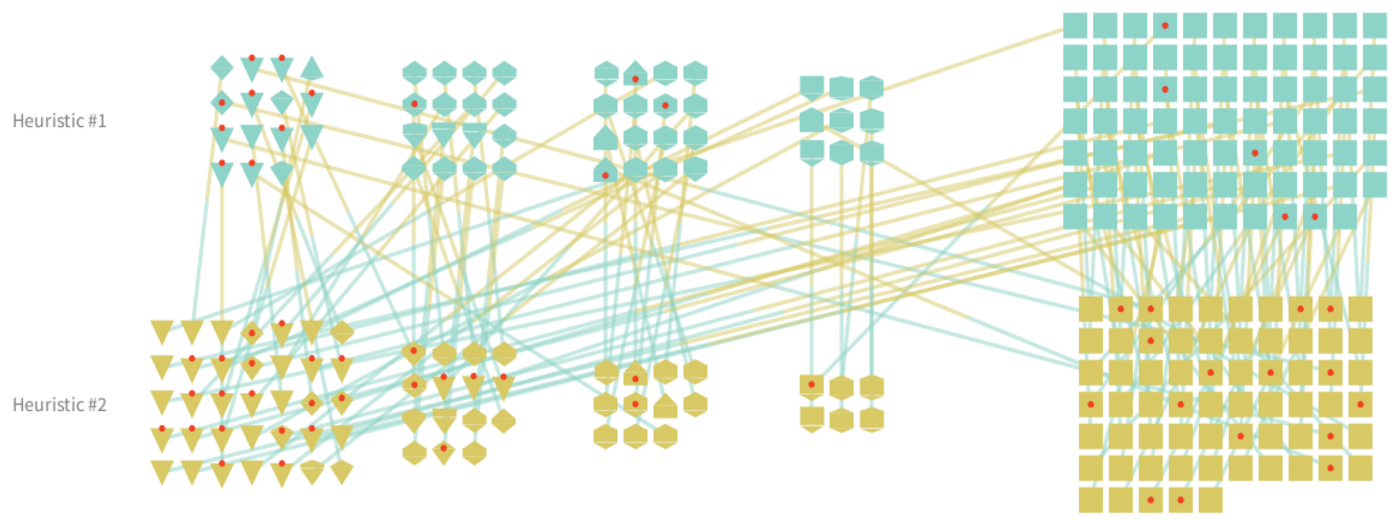
Neil Walkinshaw
Department of Computer Science
The University of Leicester
Leicester, LE1 7RH, UK
nw91@le.ac.uk

***Abstract*—Current metrics-based approaches to visualise unfamiliar software systems face two key limitations: (1) They are limited in terms of the number of dimensions that can be projected, and (2) they use fixed layout algorithms where the resulting positions of entities can be vulnerable to misinterpretation. In this paper we show how computer games technology can be used to address these problems. We present the PhysVis software exploration system, where software metrics can be variably mapped to parameters of a physical model and**

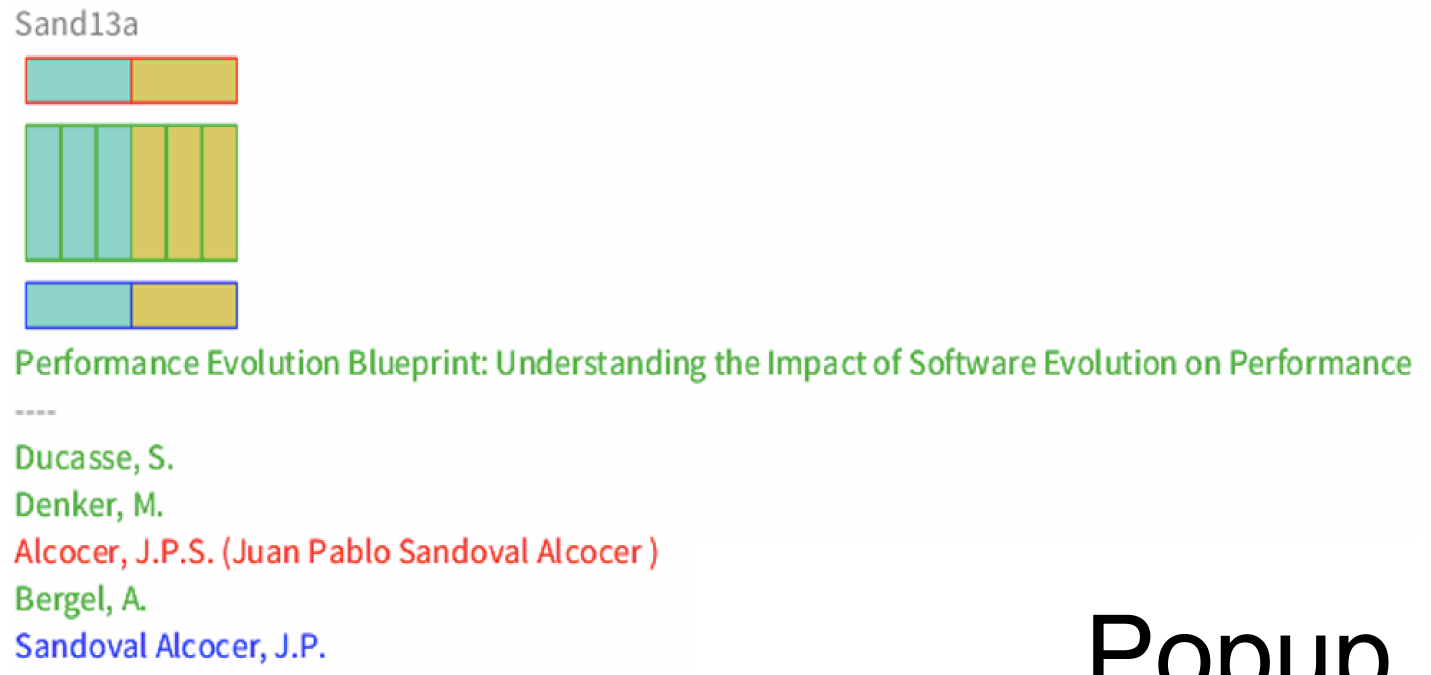
which software can be represented. As with existing techniques, PhysViz provides the means by which to represent entities in terms of their spacial coordinates, proximity, and visual properties such as colour, size and transparency. However, PhysViz also incorporates a basic implementation of Newtonian point-mass physics (a standard component of a games particle effects systems), which enables us to model entities in terms of physical attributes, such as their mass,

Title	Author 1	Author 2
Visualising Software as a Particle System	Computer Science	Neil Walkinshaw

Visualization



Assessment Grid



Sand13a

Performance Evolution Blueprint: Understanding the Impact of Software Evolution on Performance

----

Ducasse, S.

Denker, M.

Alcocer, J.P.S. (Juan Pablo Sandoval Alcocer)

Bergel, A.

Sandoval Alcocer, J.P.

Popup

Feature-centric Environment

David Rothlisberger, Orla Greevy and Adrian Lienhard

Software Composition Group
University of Berne, Switzerland
{roethlis, greevy, lienhard} @ iam.unibe.ch

1. Introduction

The task of locating the parts of the code that are relevant to a feature in object-oriented systems is widely recognized as a non-trivial task and a body of reverse engineering research collectively referred to as feature identification has emerged [1, 2]. A software engineer frequently needs to understand which parts of a system implement a feature to carry out maintenance activities, as change requests and bug reports are usually expressed in terms of features [4]. The main focus of feature identification research to date is in a reverse engineering context. Despite the fact that research has highlighted the usefulness of feature identification techniques for program comprehension, very little of this effort has found its way into the software engineer's development environment.

In this paper, we demonstrate a tool providing a perspective of a system that reflects how features are implemented to support maintenance activities. By integrating this tool in a development environment we support feature understanding while performing maintenance activities. This environment, called Feature-centric Environment, compares several features visually, provides a detailed view for a single feature and integrates a code browser focusing on a single feature of a software system. All these different views are enriched with metrics, they are interconnected and the user is able to interact with them.

In the following we introduce the Feature-centric Environment and its different views.

2. The Feature-centric Environment

The Feature-centric Environment provides three different views of features: The Feature Overview, the Feature Tree and the Feature Artifact Browser. All these three views are enriched with the Feature Affinity metric introduced in a previous work [3]. Applying this metric guides and supports the software engineer during the navigation and understanding of one or many features. We assign a color that represents its Feature Affinity value to the visual representation of a method used in a feature . Our choice of colors correspond to a heat map, e.g., colors from cyan to red.

Compact Feature Overview

The feature overview visualizes more than one feature. The software engineer can decide how many features she wants to visualize at the same time (see Figure 1 (1)). For every chosen feature, a list of all methods used in the current feature is provided. Every method is displayed as a small colored box where the color represents the feature affinity value, the list is sorted according to this metric value. Clicking on a method opens the feature tree where all occurrences of the selected method are highlighted.

Feature Tree

In the feature tree view we present the method call tree, captured as a result of exercising one feature (see Figure 1 (2)). The first method executed for a feature (e.g., the "main" method) forms the root of this tree. Methods invoked in this root node form the first level of the tree, hence the nodes represent methods and the edges are message sends from a sender to a receiver. As in the feature overview, the nodes are colored according to their feature affinity value. The tree is collapsed to the first two levels at the beginning, but every node can be expanded and collapsed again afterwards. Like this, the user can conveniently navigate even large call trees of a feature. Every node of the tree provides a button to look up the method of this node in the feature artifact browser.

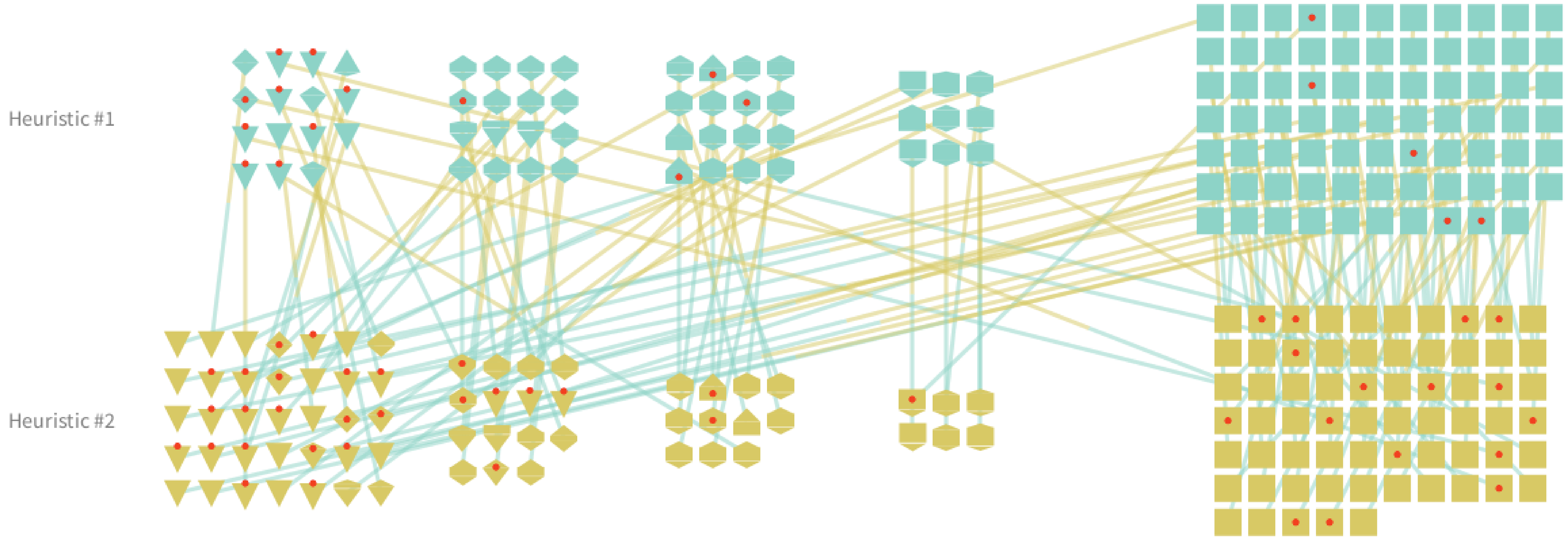
Feature Artifact Browser

The source artifacts of an individual feature are presented as text in the feature artifact browser (see Figure 1 (3)). It displays only the classes and methods actually used in the feature. Classes and methods not participating in the runtime behavior of a feature are not displayed. This makes it much easier for the user to focus on a single feature of the software. The feature ar-

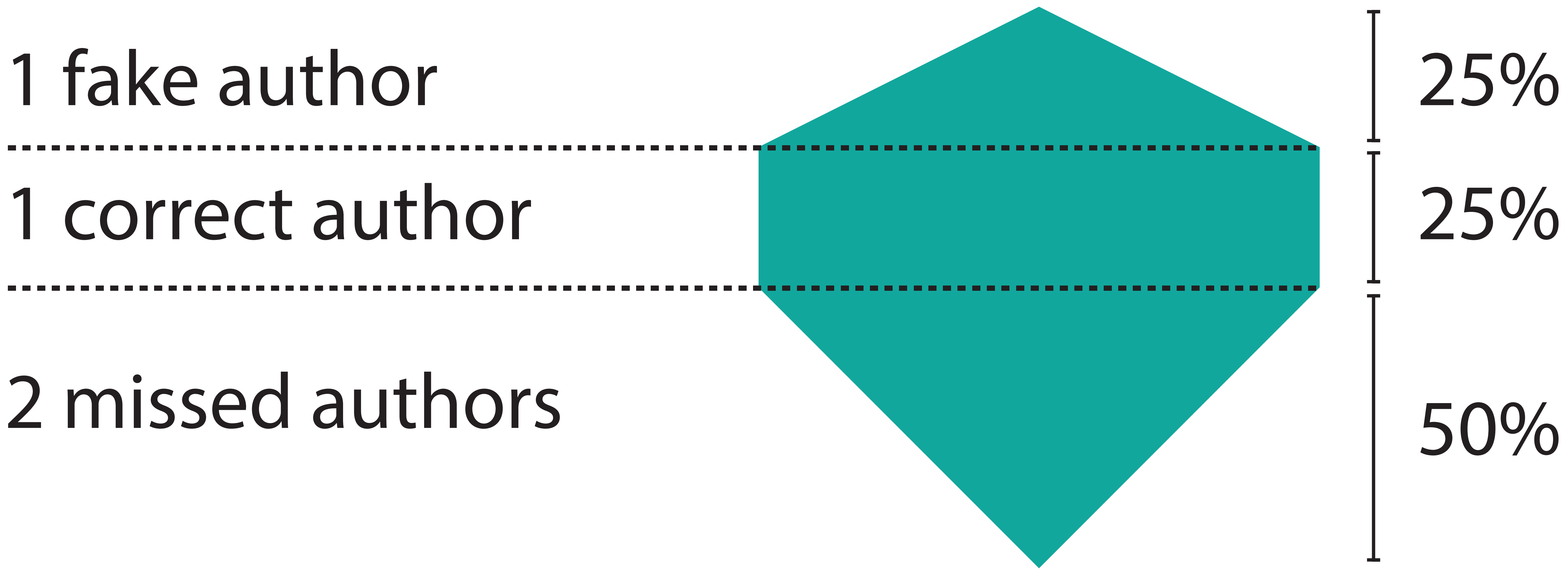
150

Document Preview

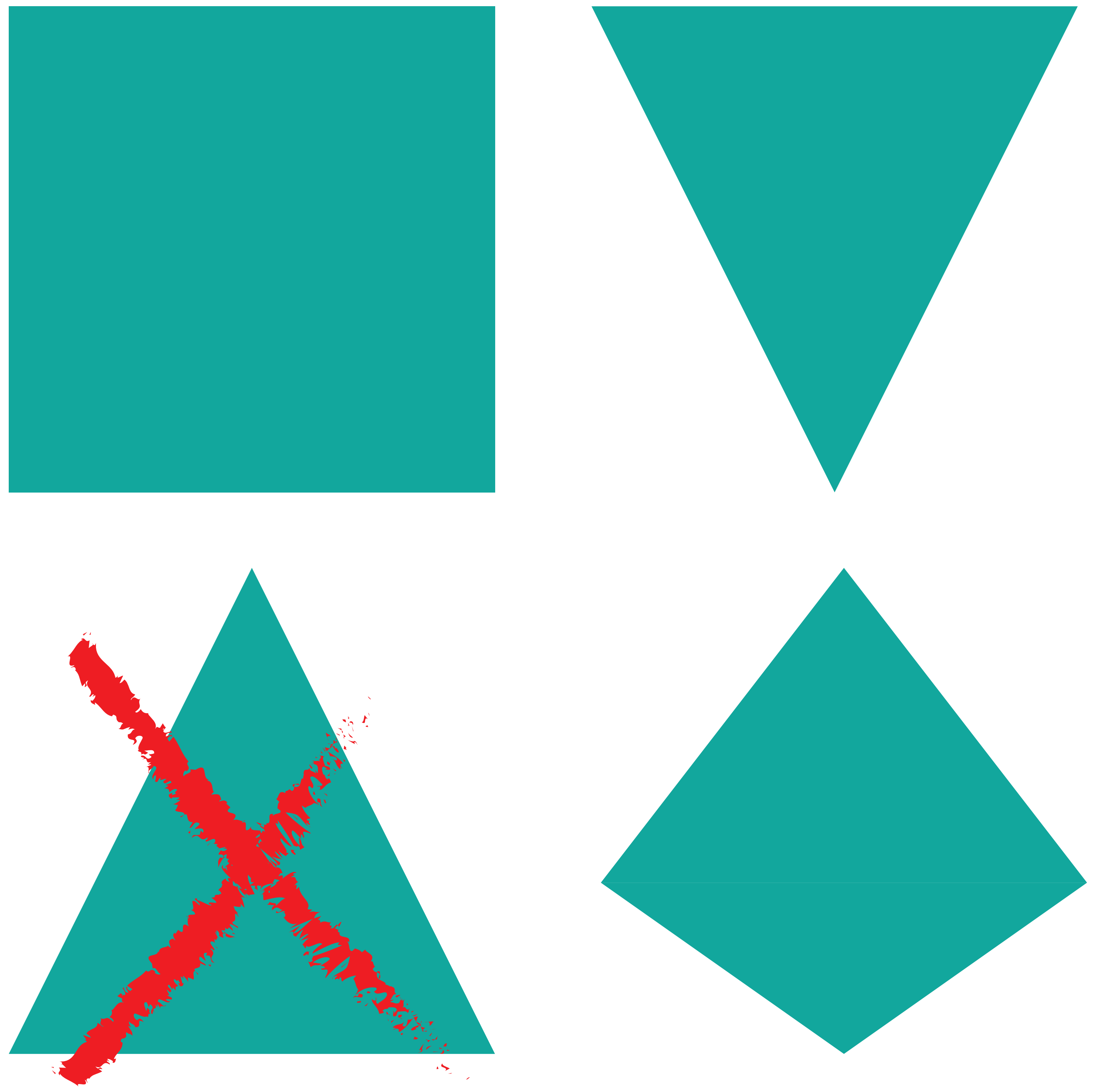
Assessment Grid



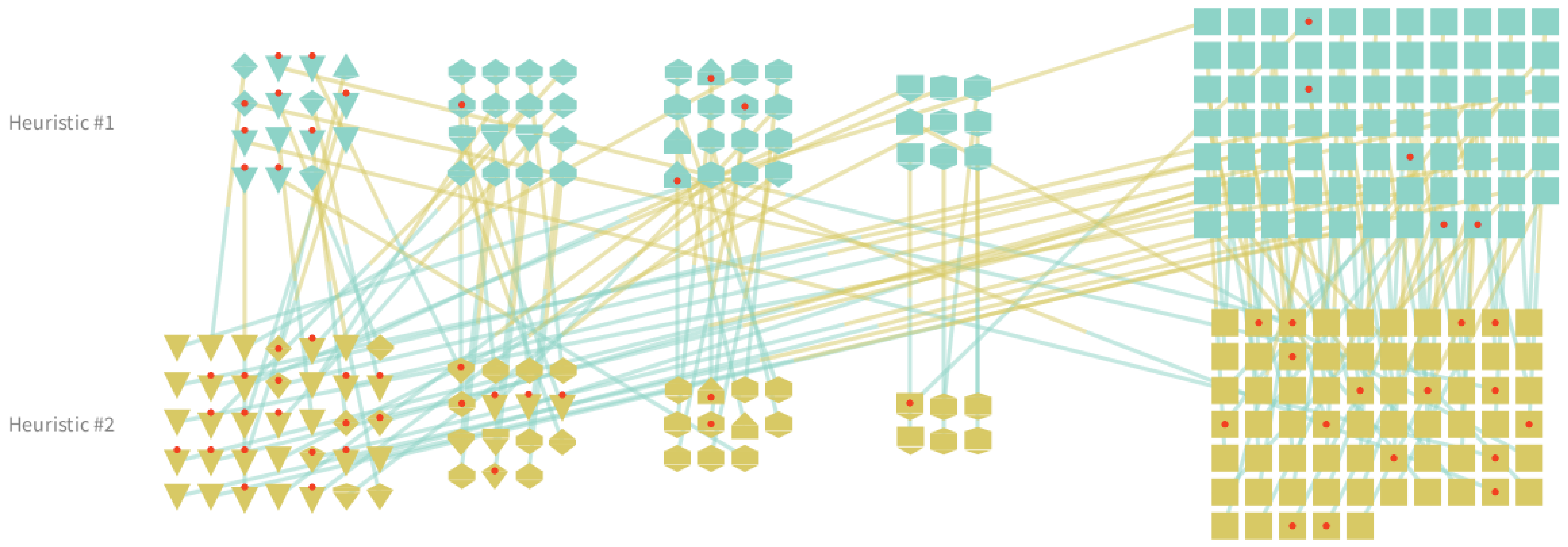
Element from the Assessment Grid (1)



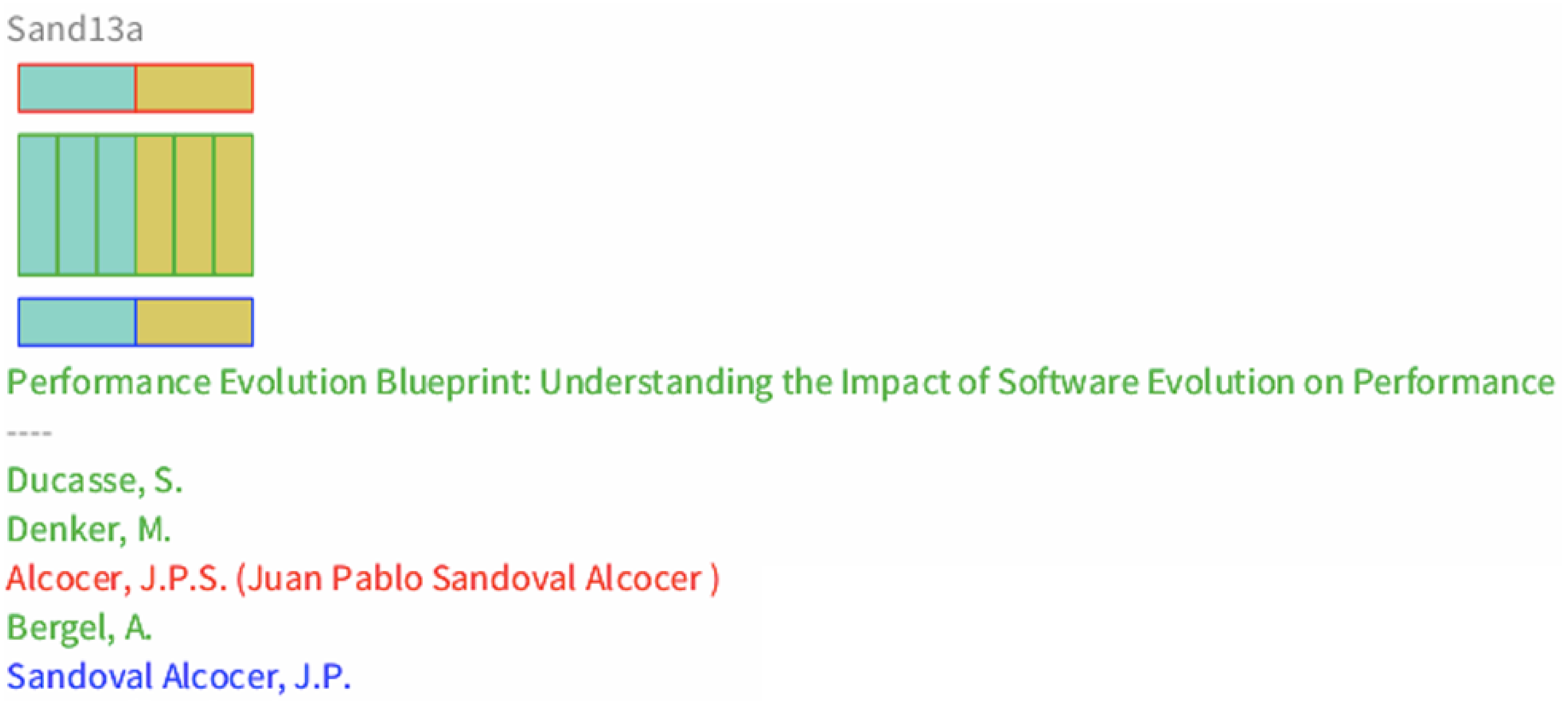
Element from the Assessment Grid (2)



Assessment Grid



Popup



Sand13a

Performance Evolution Blueprint: Understanding the Impact of Software Evolution on Performance

----

Ducasse, S.

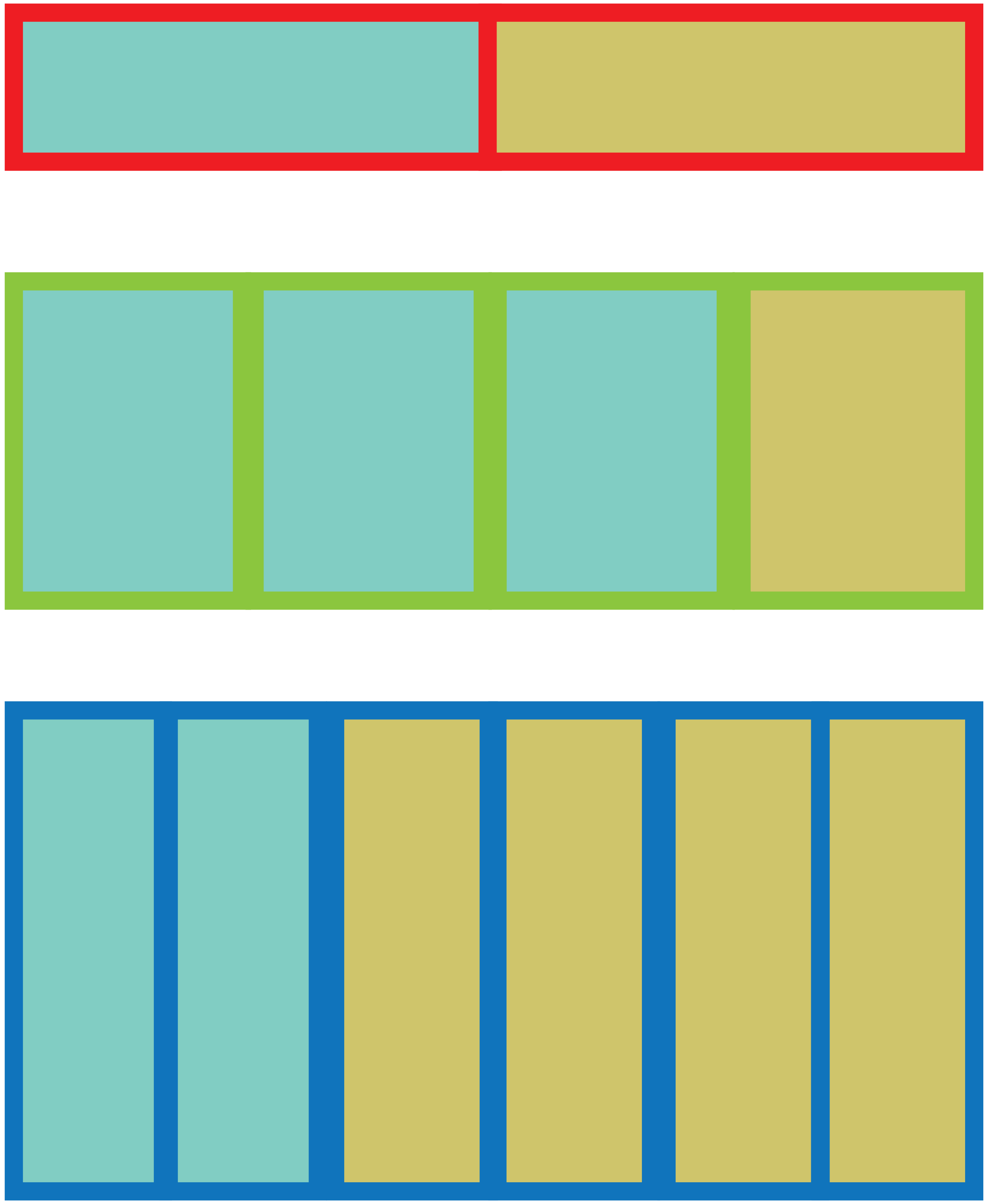
Denker, M.

Alcocer, J.P.S. (Juan Pablo Sandoval Alcocer)

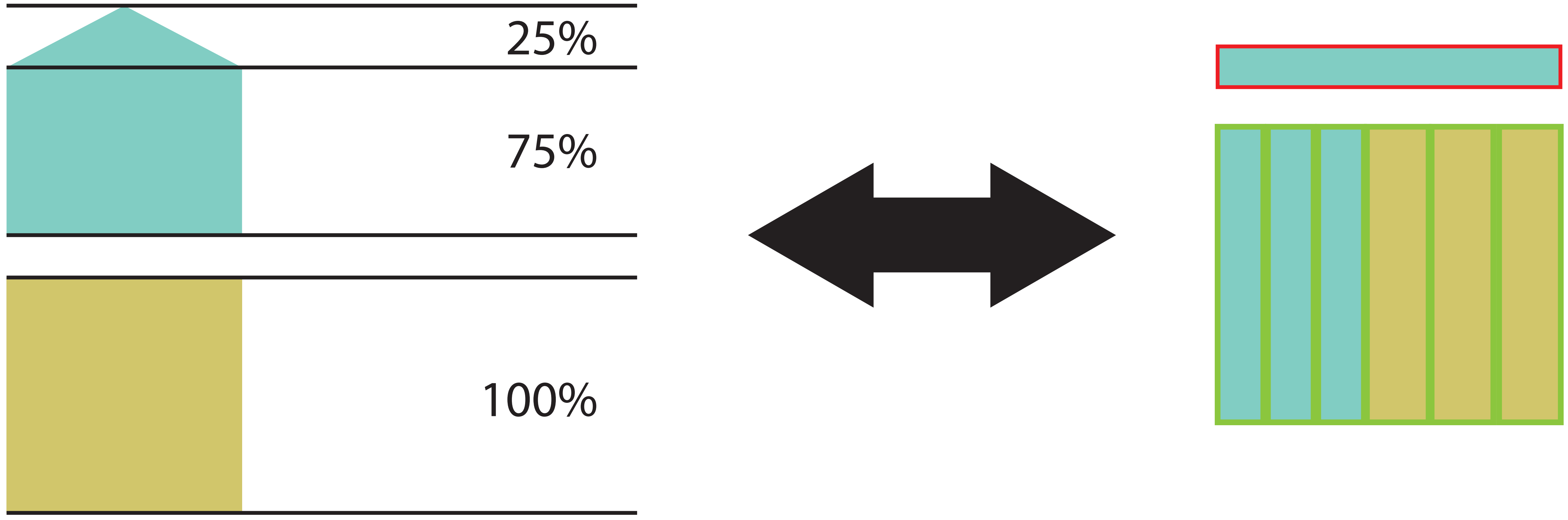
Bergel, A.

Sandoval Alcocer, J.P.

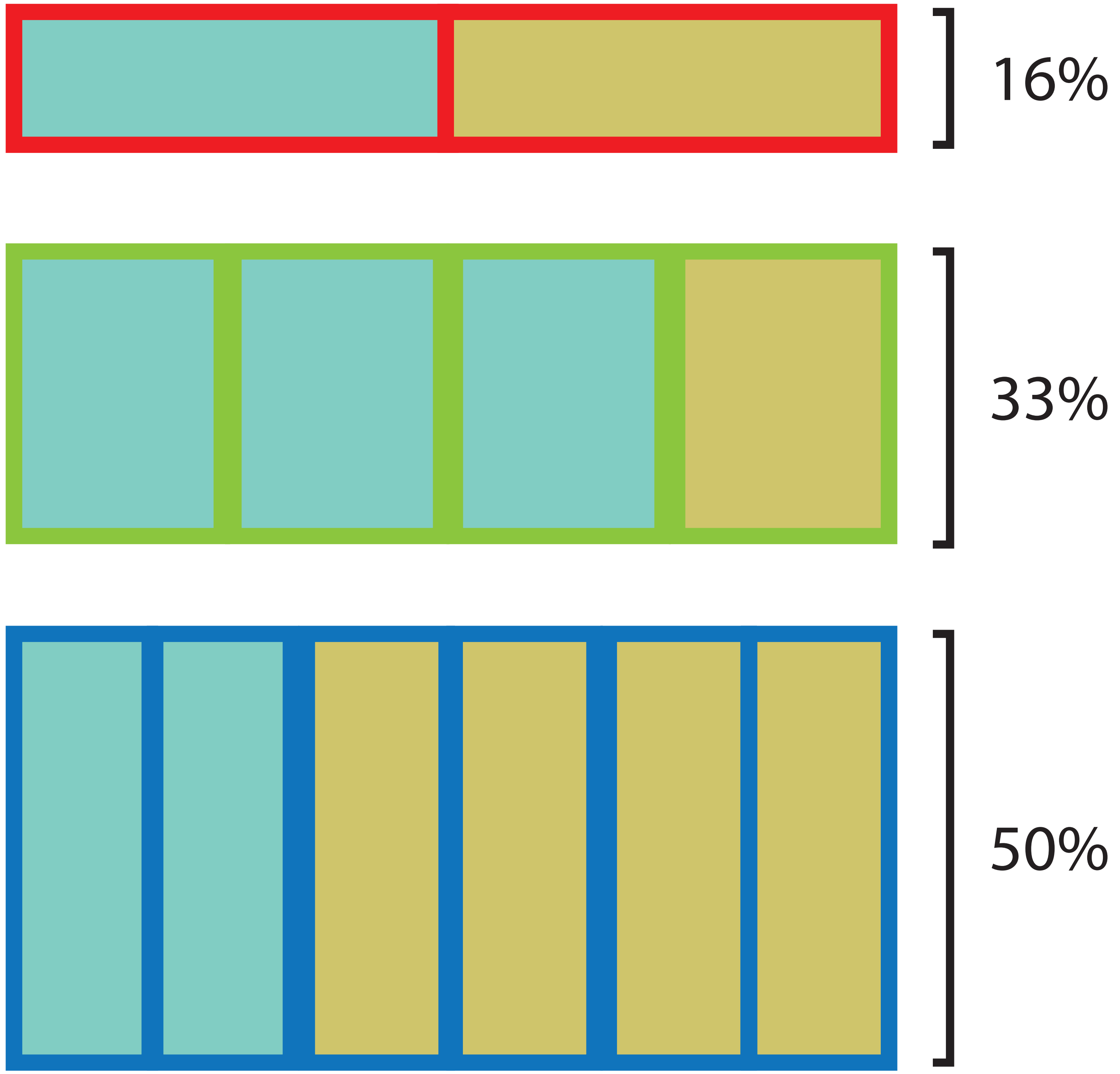
Popup Shape



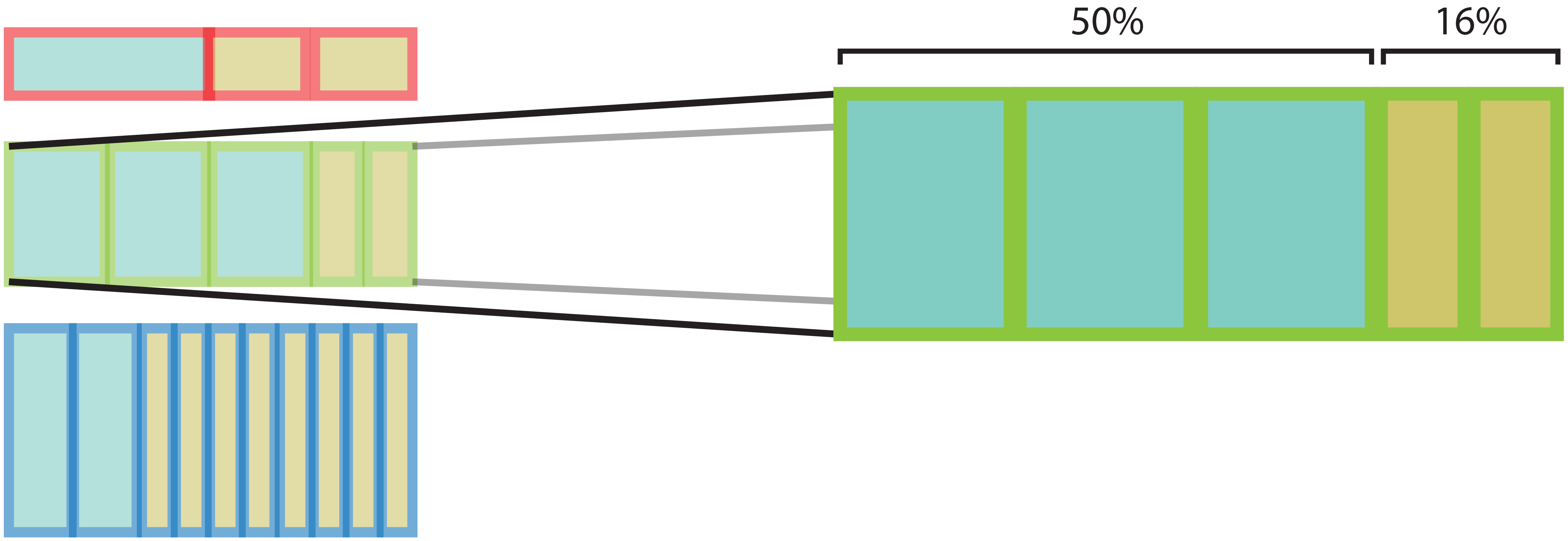
Popup Top (1)



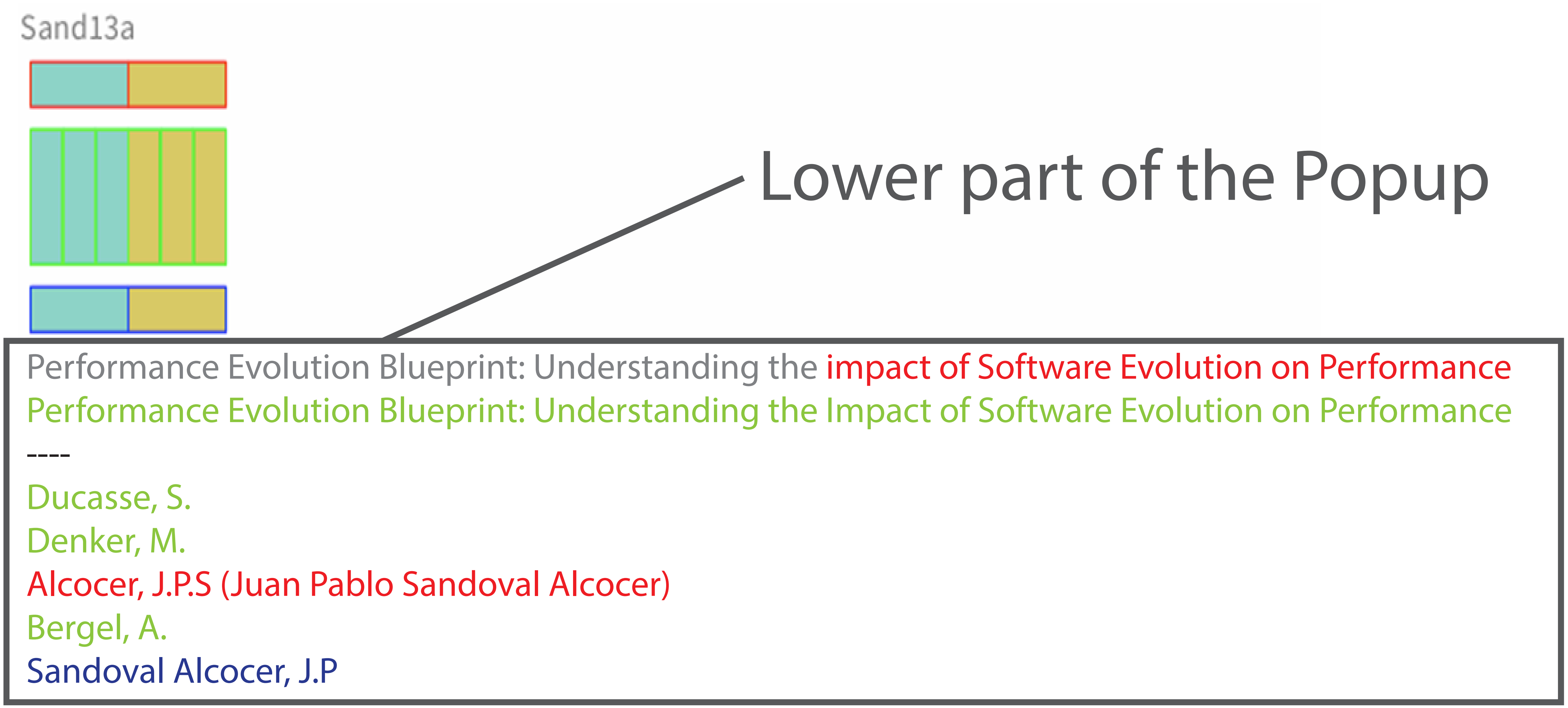
Popup Top (2)



Popup Top (3)



Popup Bottom



Lower part of the Popup

Performance Evolution Blueprint: Understanding the impact of Software Evolution on Performance
Performance Evolution Blueprint: Understanding the Impact of Software Evolution on Performance
----
Ducasse, S.
Denker, M.
Alcocer, J.P.S (Juan Pablo Sandoval Alcocer)
Bergel, A.
Sandoval Alcocer, J.P

Paper Preview

Feature-centric Environment

David Rothlisberger, Orla Greevy and Adrian Lienhard

Software Composition Group
University of Berne, Switzerland
{roethlis, greevy, lienhard} @ iam.unibe.ch

1. Introduction

The task of locating the parts of the code that are relevant to a feature in object-oriented systems is widely recognized as a non-trivial task and a body of reverse engineering research collectively referred to as feature identification has emerged [1, 2]. A software engineer frequently needs to understand which parts of a system implement a feature to carry out maintenance activities, as change requests and bug reports are usually expressed in terms of features [4]. The main focus of feature identification research to date is in a reverse engineering context. Despite the fact that research has highlighted the usefulness of feature identification techniques for program comprehension, very little of this effort has found its way into the software engineer's development environment.

In this paper, we demonstrate a tool providing a perspective of a system that reflects how features are implemented to support maintenance activities. By integrating this tool in a development environment we support feature understanding while performing maintenance activities. This environment, called Feature-centric Environment, compares several features visually, provides a detailed view for a single feature and integrates a code browser focusing on a single feature of a software system. All these different views are enriched with metrics, they are interconnected and the user is able to interact with them.

In the following we introduce the Feature-centric Environment and its different views.

2. The Feature-centric Environment

The Feature-centric Environment provides three different views of features: The Feature Overview, the Feature Tree and the Feature Artifact Browser. All these three views are enriched with the Feature Affinity metric introduced in a previous work [3]. Applying this metric guides and supports the software engineer during the navigation and understanding of one or many features. We assign a color that represents its Feature Affinity value to the visual representation of a method used in a feature . Our choice of colors correspond to a heat map, e.g., colors from cyan to red.

Compact Feature Overview

The feature overview visualizes more than one feature. The software engineer can decide how many features she wants to visualize at the same time (see Figure 1 (1)). For every chosen feature, a list of all methods used in the current feature is provided. Every method is displayed as a small colored box where the color represents the feature affinity value, the list is sorted according to this metric value. Clicking on a method opens the feature tree where all occurrences of the selected method are highlighted.

Feature Tree

In the feature tree view we present the method call tree, captured as a result of exercising one feature (see Figure 1 (2)). The first method executed for a feature (e.g., the "main" method) forms the root of this tree. Methods invoked in this root node form the first level of the tree, hence the nodes represent methods and the edges are message sends from a sender to a receiver. As in the feature overview, the nodes are colored according to their feature affinity value. The tree is collapsed to the first two levels at the beginning, but every node can be expanded and collapsed again afterwards. Like this, the user can conveniently navigate even large call trees of a feature. Every node of the tree provides a button to look up the method of this node in the feature artifact browser.

Feature Artifact Browser

The source artifacts of an individual feature are presented as text in the feature artifact browser (see Figure 1 (3)). It displays only the classes and methods actually used in the feature. Classes and methods not participating in the runtime behavior of a feature are not displayed. This makes it much easier for the user to focus on a single feature of the software. The feature ar-

150

Use Case

Conclusion

1. Simultaneously creating pipelines and the visualization
2. ~70% accuracy with help from the visualization
3. Visualization can be used to further improve the heuristics

Summary

1. Need for modeling communities
2. Pipeline for creating such models
3. Assessing the output of a pipline with the visualization

Questions