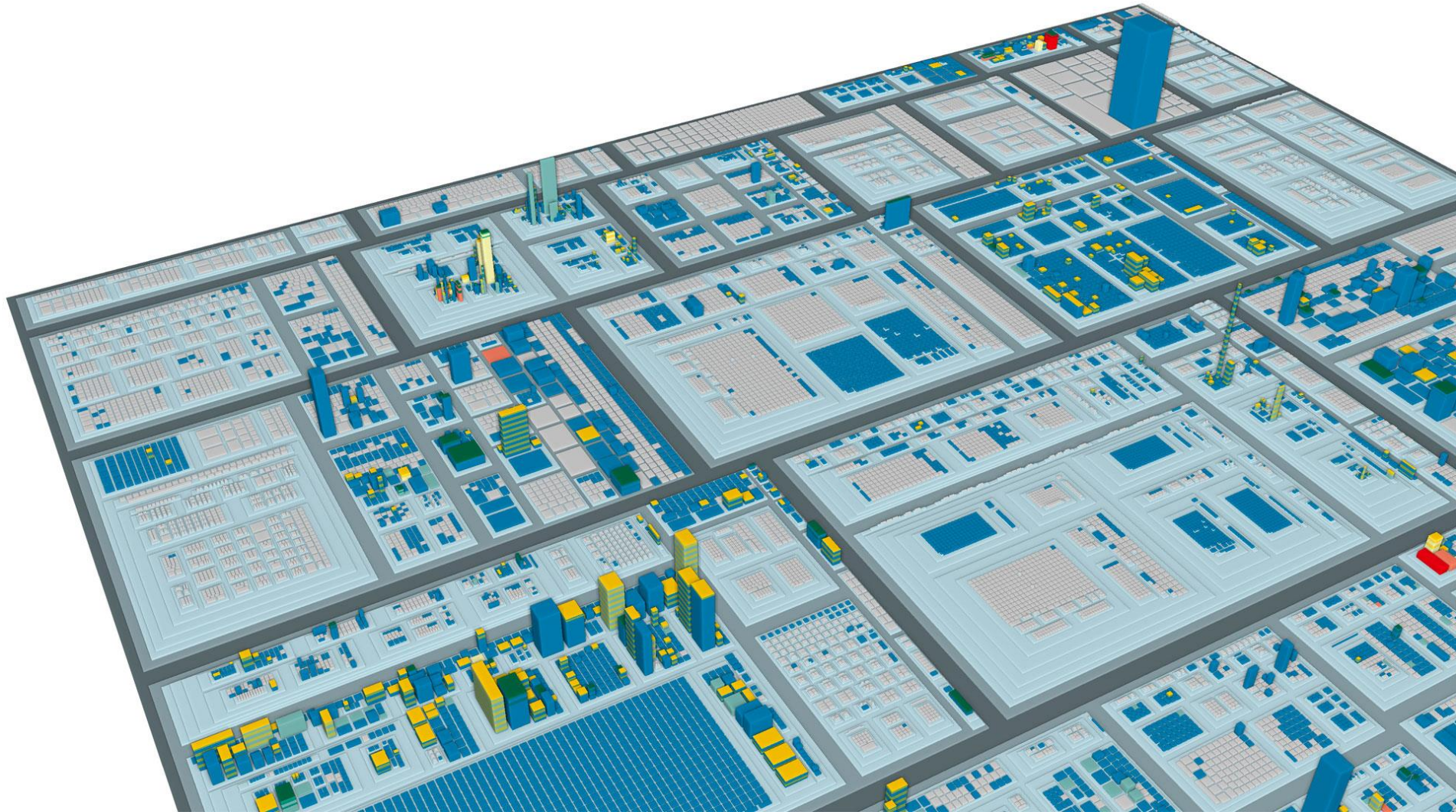
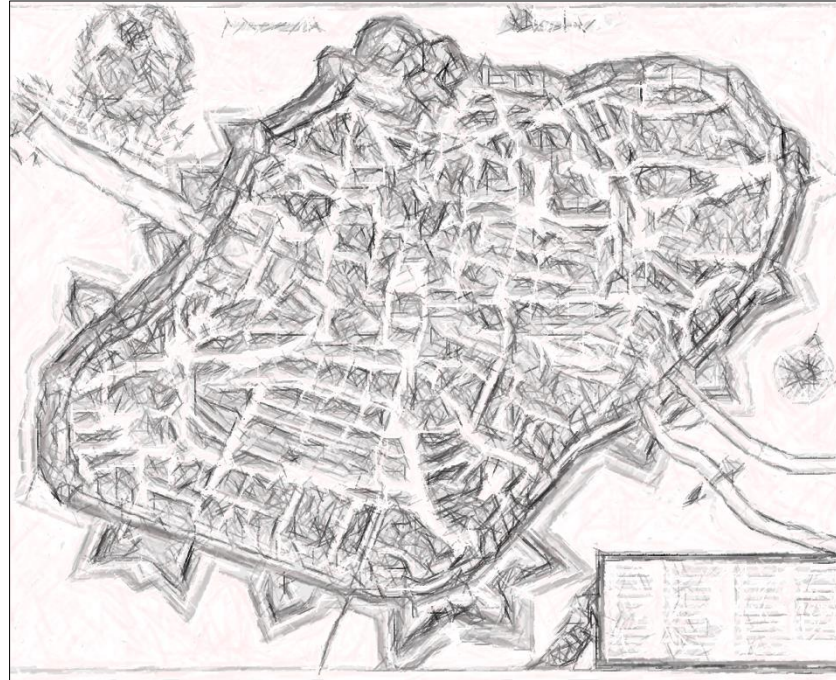


Automated Visual Software Analytics

OpenHPI – Prof. Dr. Jürgen Döllner – 2015

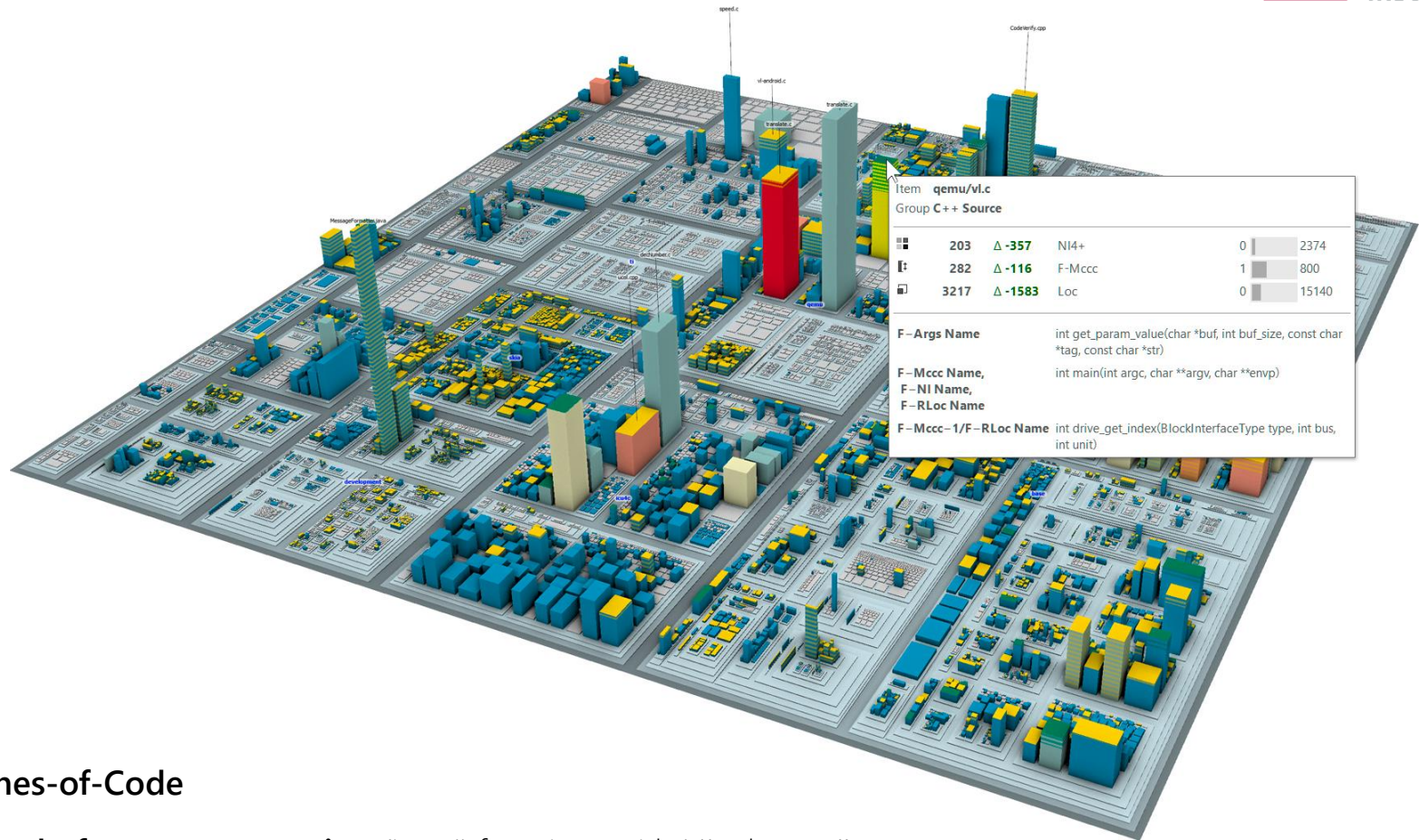


Software Map



Principles of Software Maps
Information Sources
Information Mapping

Principles of Software Maps



 **Lines-of-Code**

 **Level of statement nesting** (i.e., # functions with NL above 4)

 **McCabe complexity** (i.e., # paths of control flow; maximum McCabe complexity)

A software map depicting **complexity hotspots** in the implementation of the Android software system.

"A software map represents an adaptive, hierarchical representation of software implementation artifacts such as source code files being organized in a modular hierarchy. The maps can express and combine information about software development, software quality, and system dynamics; they can systematically be specified, automatically generated, and organized by templates. The maps aim at supporting decision-making processes."

– J. Bohnet, J. Döllner: "Monitoring Code Quality and Development Activity by Software Maps", IEEE ACM ICSE Workshop on Managing Technical Debt, 9-16, 2011

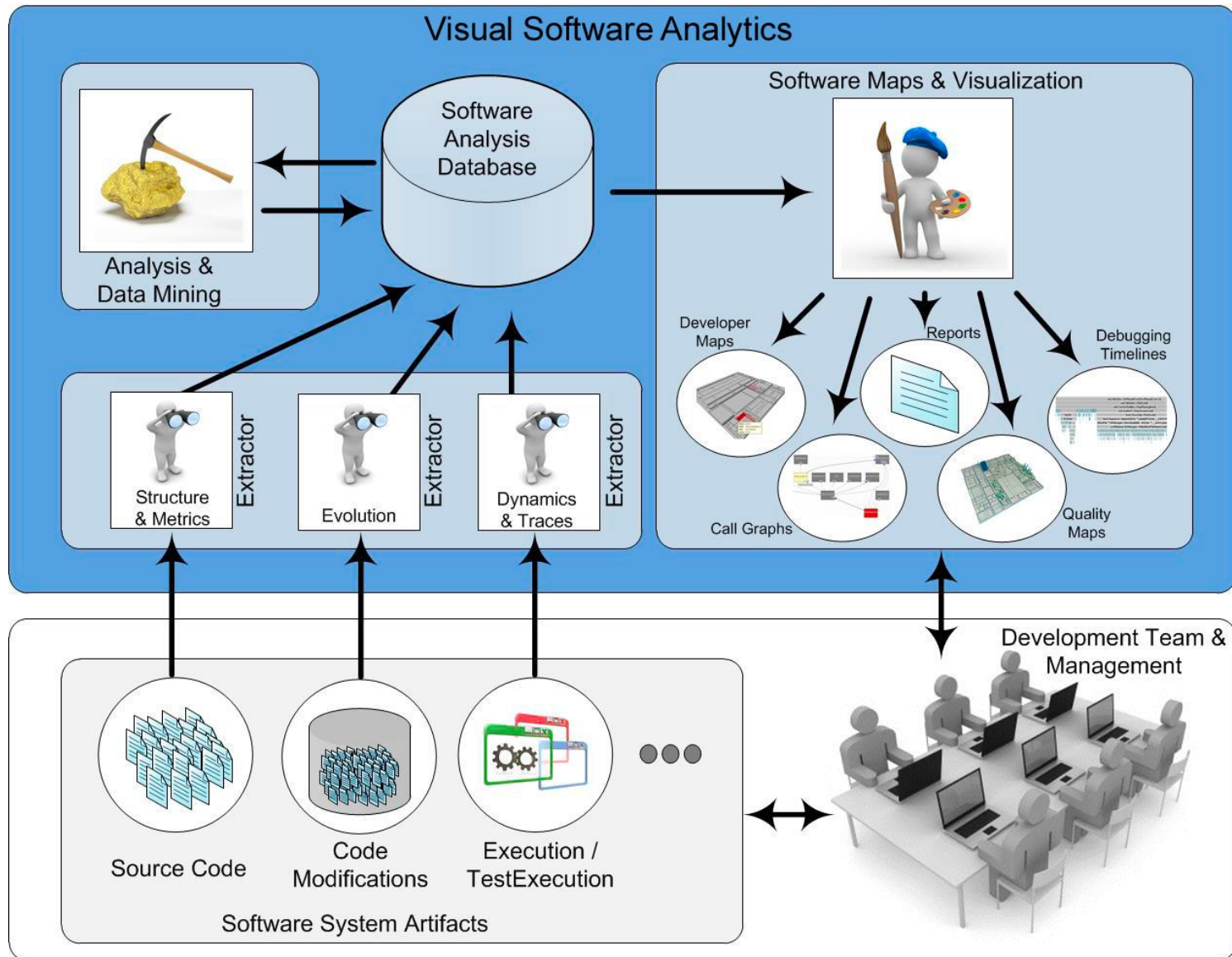
Software Map

A *software map* defines a visual mapping of a hierarchy of software artifacts and related information to a 2.5D treemap.

Software maps generally visualize information about:

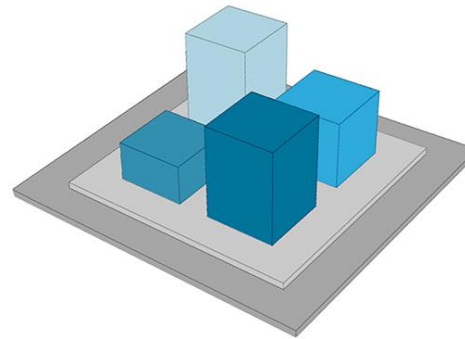
- source code
- software quality
- software dynamics and tests
- software evolution
- software development processes

Information Sources



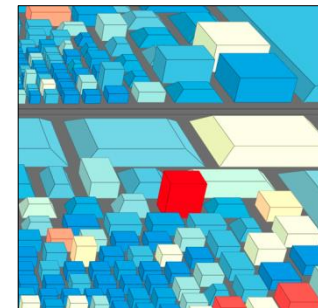
- **Primary Map Data:** Generally, three information dimensions can be mapped onto three primary visual variables of software maps:

- cuboid ground base
- cuboid color
- cuboid height



- **Secondary Map Data:** Additional information can be mapped onto secondary visual variables of software maps (i.e., having a less significant perceptual impact)

- side-face and top-face textures (e.g., hatching, patterns, ...)
- cuboid shape variations (rounded corners, conical shape, ...)

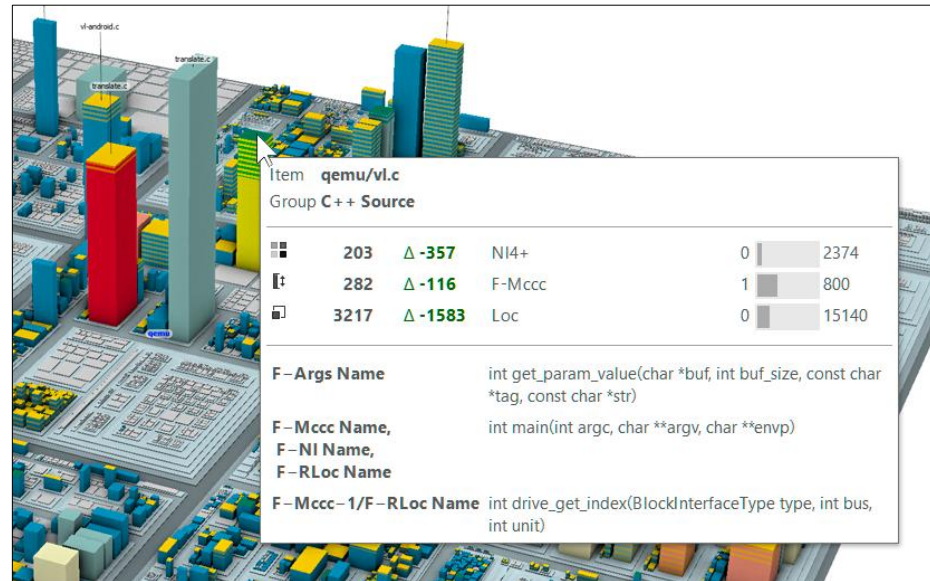


Map Theme

A concrete combination of primary map data and, optionally, secondary map data, that portrays specific aspects of software development and software engineering. Concrete combinations of map data are organized by software map templates.



Example Theme: "Implementation Complexity Hotspots". It combines the metrics LOC (ground base), McCabe code complexity (height), and code nesting level (color) as primary map data.



Software Map Interaction: Software maps are interactive interfaces to software analytics information and results. For example, the user can query the software artifact's name and the associated thematic values as well as details such as normal value ranges.

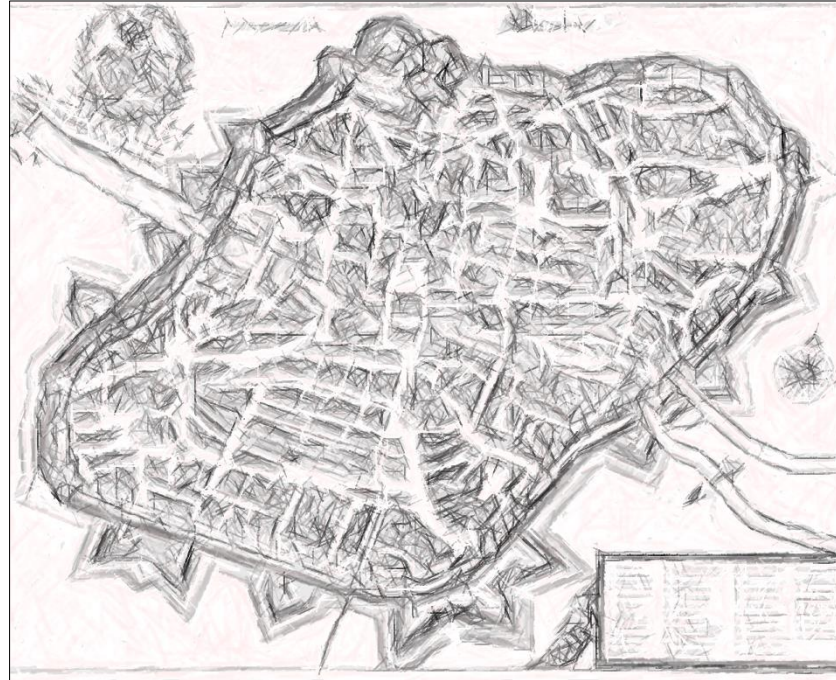
Software maps interactively visualize software analytics information such as source code, software quality, software dynamics and tests, software evolution, and development processes by means of 2.5D treemaps.

A **software map theme** represents a concrete combination of software analytics information sources and their visual mapping to primary and secondary visual variables. For different aspects in software development and software engineering, different themes for software maps are used.

Software maps support **different stakeholders** in software engineering in **data-driven decision making and finding actionable insights**.

Software maps serve as a **key tool for visual analytics within software analytics**.

Software Map



Principles of Software Maps
Information Sources
Information Mapping

Software Map

A *software map* defines a visual mapping of a hierarchy of software artifacts together with related information –about source code, software quality, software dynamics, software evolution, and development processes– to a 2.5D treemap.

Primary and Secondary Visual Variables

For software maps, the information can be mapped to the primary visual variables (i.e., cuboid ground base, height, and color) as well as to the secondary visual variables (e.g., cuboid texture and other shapes and shape variations).

Software Map Theme

A *software map theme* represents a concrete combination of software analytics information sources and their mapping to primary and secondary visual variables that portrays specific aspects of software development and software engineering issues.

HPI Hasso Plattner Institut

Monitoring Code Quality and Development Activity by Software Maps

Johannes Dörmann, Jürgen Götting
Hasso Plattner Institute for IT Systems Engineering, University of Potsdam
Germany
{joelnet, doerjann}@uni-potsdam.de

ABSTRACT

Software development is a difficult task to manage due to the diverse responsibilities within teams, and the need for communication and coordination. The use of software maps can help to overcome these difficulties. Software maps are graphical representations of the development process, which allow for a better understanding of the development process and the roles of the team members. This paper presents a software map tool, which allows for the automatic generation of software maps. The tool is based on the software map concept, which is a graphical representation of the development process. The tool is based on the software map concept, which is a graphical representation of the development process. The tool is based on the software map concept, which is a graphical representation of the development process.

1. INTRODUCTION

Software development is a complex task, which involves many different aspects. One of the most important aspects is the management of the development process. This is often done by using software maps. Software maps are graphical representations of the development process, which allow for a better understanding of the development process and the roles of the team members. This paper presents a software map tool, which allows for the automatic generation of software maps. The tool is based on the software map concept, which is a graphical representation of the development process. The tool is based on the software map concept, which is a graphical representation of the development process. The tool is based on the software map concept, which is a graphical representation of the development process.

2. SOFTWARE MAP CONCEPT

The software map concept is a graphical representation of the development process. It is based on the idea of a map, which shows the different parts of the development process and how they are connected. The software map concept is a graphical representation of the development process. It is based on the idea of a map, which shows the different parts of the development process and how they are connected. The software map concept is a graphical representation of the development process. It is based on the idea of a map, which shows the different parts of the development process and how they are connected.

3. SOFTWARE MAP TOOL

The software map tool is a software tool, which allows for the automatic generation of software maps. It is based on the software map concept, which is a graphical representation of the development process. The software map tool is a software tool, which allows for the automatic generation of software maps. It is based on the software map concept, which is a graphical representation of the development process. The software map tool is a software tool, which allows for the automatic generation of software maps. It is based on the software map concept, which is a graphical representation of the development process.

4. CONCLUSION

The software map tool is a software tool, which allows for the automatic generation of software maps. It is based on the software map concept, which is a graphical representation of the development process. The software map tool is a software tool, which allows for the automatic generation of software maps. It is based on the software map concept, which is a graphical representation of the development process. The software map tool is a software tool, which allows for the automatic generation of software maps. It is based on the software map concept, which is a graphical representation of the development process.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques
D.2.3 [Software Engineering]: Testing Tools and Techniques
D.2.4 [Software Engineering]: Program Development Tools

General Terms

Design, Languages, Verification

Keywords

Software Maps, Software Quality, Software Development, Software Engineering, Software Development

Research supported by the German Research Foundation (DFG) under the Special Collaborative Program SFB 375/B1. The authors would like to thank the members of the Hasso Plattner Institute for their support and advice.

9

Software maps provide a unique and general-purpose interactive "interface" to software analytics. The underlying 2.5D treemap allows for mapping two or more information dimensions onto the hierarchy of software system artifacts.

In particular, this allows us to explore expected and to uncover unexpected correlations between information dimensions in a straightforward way.

Software maps can be cartographically enhanced by using secondary visual variables to provide a higher level of detailed information in the depicted map.

A number of "useful combinations" have already emerged, that is, a catalogue of software map themes simplifies the practical application of software maps in software development and software engineering (subject to the parts about "software map use cases").

Software Analytics in Practice



Motivation and Problem Statement
Creating Software Maps
Exploration and Analysis

Example Scenario: “The Shipwrecked System XY”

- The XY system has been developed over the last 10 years;
 - *Topic*: The software system provides an industry solution for object inquiry, presentation, modification, distribution, and storage.
 - *Functionality*: The system is based on complex functionality (e.g., computational geometry, 3D computer graphics, data management algorithms), including different user interfaces and APIs.
 - *Implementation*: The major parts of the system have been implemented in C++; the system’s source code has been managed with a standard version control system.
- This legacy system represents the **key asset** of the company and its services and products.

Example Scenario: “The Shipwrecked System XY”

- The project was shipwrecked and stopped recently.
- Reasons include for example:
 - New features appeared in one release but disappeared in the next release;
 - features remained or became instable;
 - most required key features could not be implemented in a reasonable time frame;
 - even minor changes resulted in major new subsystem developments.
- In summary, the team hardly achieved their goals, in particular in the last two years, in time and within given budgets.
- The investors realized that further investments require fundamental changes to the software and its team.



Software Due Diligence

A new management wants to **explore and analyze the as-is situation** and understand the system structure and quality, its development history, and identify its principle weaknesses.

To decide whether to invest new money, the investors and the management decided for a data-driven approach based on software analytics, after receiving many subjectively shaped stories and explanations from different team members, different consultants, and the former CEO – all these stories did not converge. Furthermore, they did not provide an overview of the entire system (in terms of software quality measures) nor about its evolution.

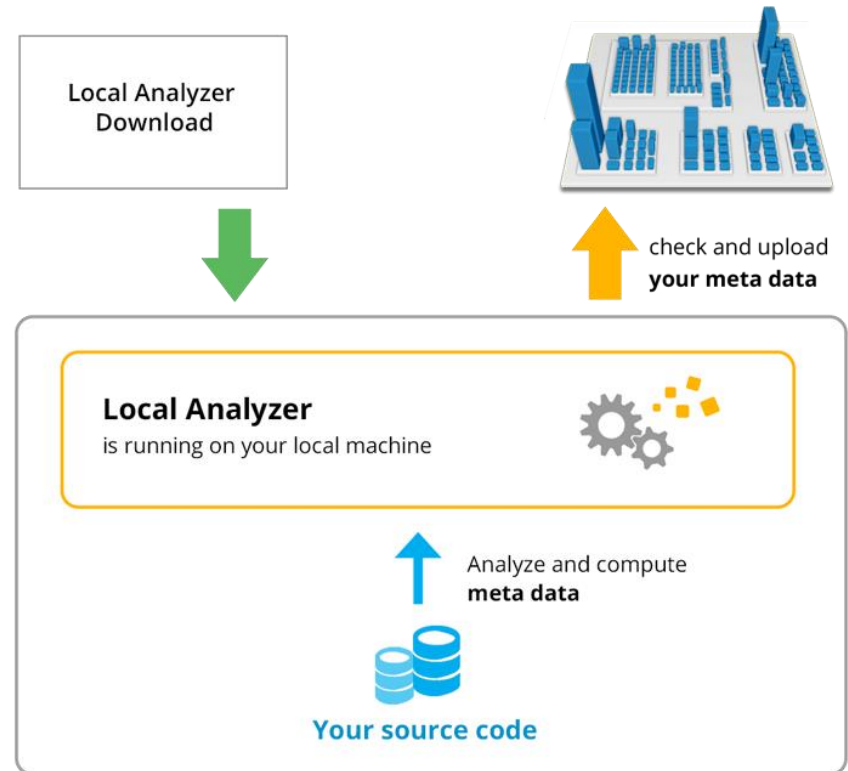
The investors and the new management ask for decision support, for example, regarding a better team setting, reusing options for existing system components, and maintenance strategies for the existing system and estimations of cost by applying methods and techniques of software analytics.

- As software analytics tool, in this example, we apply the **Effectiveness Platform** that analyzes and evaluates software systems and their development processes based on predictive and prescriptive software analytics.
- The Effectiveness Platform can be used to automatically integrate all relevant software engineering data of a software project and to “translate” the results into different types of actionable insights.
- The Effectiveness platform supports software metrics for a large number of programming languages, revision control systems, systems for issue and bug tracking, and collaborative software development services.
- *Short Tutorial by Dr. Felix Salfner from Software Diagnostics*

Creating Software Maps

Creating the analytics database:

1. Creating an analytics project in the Effectiveness Platform
2. Downloading the 'LocalAnalyzer' tool
3. Configuring and running the automatic data analysis on your local machine/environment
4. Uploading the derived analytics data to the Effectiveness Platform

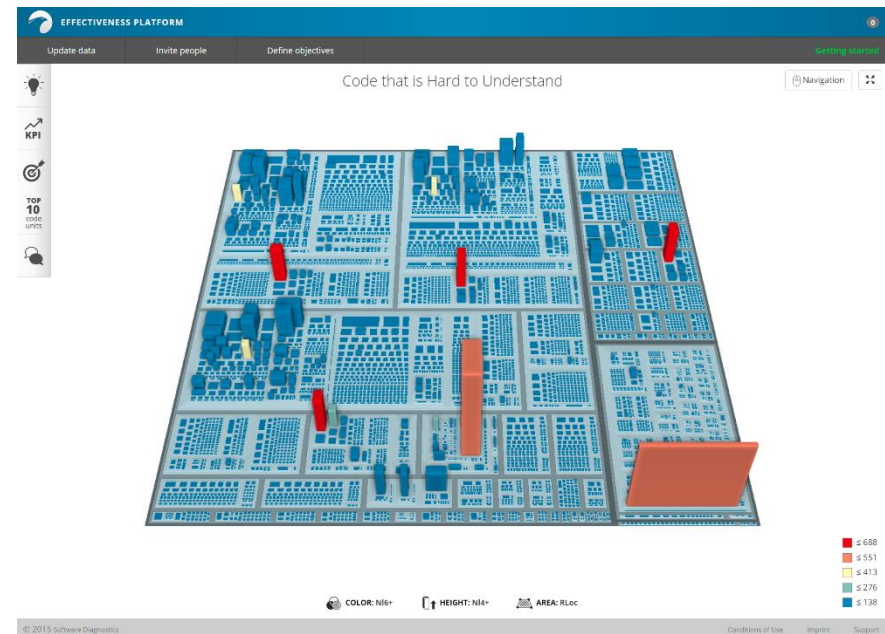


URL: <https://tp.softwareiagnostics.com>

Exploration and Analysis

Start with Analytics Tasks on the Platform

- Explore interactive software maps
- Inspect hotspots and outliers
- Invite other people to share your insights
- Reason and discuss about phenomena and their possible root causes



Exploration and Analysis

Visual Variable: Ground Area



LLOC

Visual Variable: Height



McCabe

Visual Variable: Color



Developers



Exploration and Analysis

Visual Variable: Ground Area



LLOC

Visual Variable: Height



McCabe

Visual Variable: Color

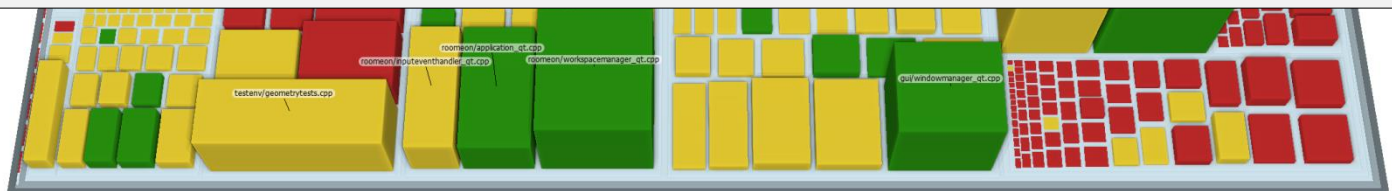


Developers



Mapping:

■ one, ■ two, ■ more than two developer(s) touched the code in that year.



Exploration and Analysis

Visual Variable: Ground Area



LLOC

Visual Variable: Height



McCabe

Visual Variable: Color



Developers



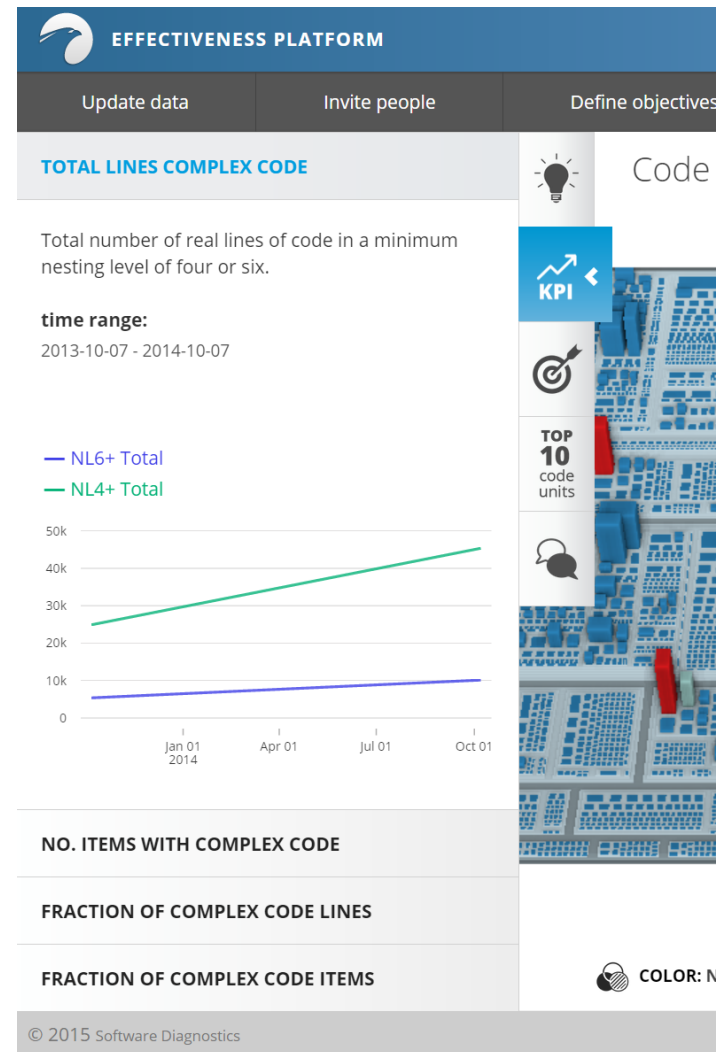
Main Insight:

Most code is owned/known by only a few/only a single developer ...

Exploration and Analysis

Effectively Managing Software Development

- Monitor key performance indicators
- Define *objectives*
 - Specify goals (e.g., target values)
 - Specify a timeframe in which you want to achieve the goals
- Get informed about trends in advance
- Establish a proactive risk and problem management



Summary

Software analytics tools such as the Effectiveness Platform provide instruments to put software analytics in practice. Most processes that are required by software analytics can be fully automated to provide an up-to-date information basis and decision-support platform at any time.

Software analytics tools can be applied to projects of any size, ranging from small first-time developments to the management of large legacy systems.

Software analytics is also key to enable a cross-project benchmarking that will help project teams to take advantage of predictive and prescriptive software analytics: Findings distilled from a large collection of projects serve as the information basis for a software-analytics early warning system.

Software Analytics in Practice



Motivation and Problem Statement
Creating Software Maps
Exploration and Analysis

Map Theme "Discovering Error-Prone Code"



Error-Prone Code

Software Maps Depicting "Error-Prone Code"

Application and Actionable Insights

Error-prone code is one grave quality issue
in software engineering and software maintenance

Negative effects:

- High bug fixing costs for deployed or operational systems
- Potential damage on the reputation of a product or company
- Conflicts of bug fixing activities with scheduled development activities
- Negative impact on developer team motivation
- ...

Factors for error-prone code

- **Software Complexity:** Error-prone code is correlated with software complexity, measured for example by software metrics such as NL (Nesting Level), McCabe (Cyclomatic Complexity), Halstead metrics, etc.
- **Software Size:** Generally, error-prone code is an inherent property of all large-scale systems, that is, error-prone code fractions are proportional to the size of a module (measured by LOC metrics).
- **Software Development:** Error-prone code also correlates with development activities. Roughly speaking, if *many* things in a source code unit are changed, most likely *many* errors are introduced as well.

Error Prone Code - How to identify such code?

- There is no single efficient strategy to identify or catch error-prone code or even errors as they have manifold reasons and appearance.
- The software analytics approach to catch error-prone code takes into account software quality metrics and development activity – in correlation they provide an effective detection of code that is likely to be error-prone.
- Software maps with a corresponding map theme facilitate identifying the most error-prone suspected modules.

Software Maps Depicting Error-Prone Code

Visual Variable: Ground Area



LLOC

Visual Variable: Height



McCabe

Visual Variable: Color



Developers

■ > 11

■ = 2-11

■ = 1



Composition of the Software Map Theme “Error-Prone Code”

- LLoC, mapped to cuboids’ footprint size, is used to expose monolithic code as a first indicator to code that is hard to manage
- McCabe complexity, mapped to the cuboids’ height, is used to indicate complex, error-prone coding
- Number of involved developers that changed a given source code unit, mapped by a color scheme, as a measure for bug fixing effort
- Combination of McCabe complexity and bug fixing effort guide awareness towards modules that are highly complex and often touched in the course of bug fixing.
- **Red tall cuboids** can be identified easily as bug-fix hotspots.

Typical use-cases and applications

- Error-prone source code units should be identified and revised as early as possible to reduce the number of error-related incidents in the future
- Focus software development activities on highly error-prone code to maximize the impact of invested effort
- Precisely adapt testing activities to ensure that all highly error-prone source code units are actually covered

Potential conclusions and way of proceeding

- First, identify the most error-prone code items
- Then try to reduce their complexity as well as to ensure proper testing of those items

The **software map theme “error-prone code”** can be applied to get an overview of error-prone source code units. It uses software analytics, in particular, to assist to manage one root cause of low software quality by revealing the unfortunate correlation of complex code and code changes.

As actionable insights, the software map guides responsible stakeholders to prioritize those modules that deserve complexity reduction and better testing to increase code stability.

The approach underlying this software map theme applies the “yesterday’s weather” rule: Code items that caused errors in the recent past will most likely be the root cause of future errors. Typically refactoring measures are adopted for those modules.

Map Theme "Discovering Error-Prone Code"



Error-Prone Code

Software Maps Depicting "Error-Prone Code"

Application and Actionable Insights

Map Theme "Technical Debts"

Bank of Cyprus				
Account 00711 11			IBAN CY0000100111000004711 11	
Bank of Cyprus *** Statement ***				
Statement Nr. 3-272011			*** International Service - Statement German Language ***	
Tag	WERT	Text	Versuchung	Saldo
15.03.	15.03.	1001	Alter Kontostand Auszug b-3/2011	54.000,00 EUR
15.03.	15.03.	1000	Abhebung Geldautomat DE-90120122	1.000,00 -
15.03.	15.03.	1004Q	Abhebung Geldautomat CH-80322311-1	1.000,00 -
15.03.	15.03.	1010	Überweisung „Überweisung beibehalten“	40.000,00 -
17.03.	15.03.	9001	Storno WERT Summe Wert 15.03.	50.000,00 +
17.03.	15.03.	1001-CY	*** EU Sanierungsbeitrag *** 5,75 % aus Wert 15.03.	1.375,00 -
18.03.	16.03.	9001-CY	Korrektur	50.000,00 -
18.03.	15.03.	1002	Neuer Kontostand	1.375,00 -
Mr. Max Mustermann			Credit 00000,00	Saldo 1.375,00 - EUR
Musterstr. 1			Debit 1.375,00 - *** BITTE ÜBERWEISEN ***	
DE - 47111 Musterkredit			Date 2013.03.19	

Technical Debt

Software Maps Depicting "Technical Debts"

Application and Actionable Insights

Technical Debts in Software Engineering

- “Technical debt is the debt that accrues when you knowingly or unknowingly make wrong or non-optimal design decision.”

–G. Suryanarayana et al. "Refactoring for Software Design Smells, 2014
- Coined originally by Ward Cunningham 1992, technical debts are introduced whenever developers choose to have a quick or ad hoc implementation rather than a well-designed implementation. The accrued “interests” accumulate (like for financial debts). Refactoring is one way to dissolve technical debts.
- Technical debts
 - aggravate program comprehension
 - increase the risk of introducing bugs in the code
 - decrease precision of estimations
 - ...

Software Maps Depicting “Technical Debts”

Visual Variable: Ground Area



LLOC

Visual Variable: Height



McCabe

Visual Variable: Color

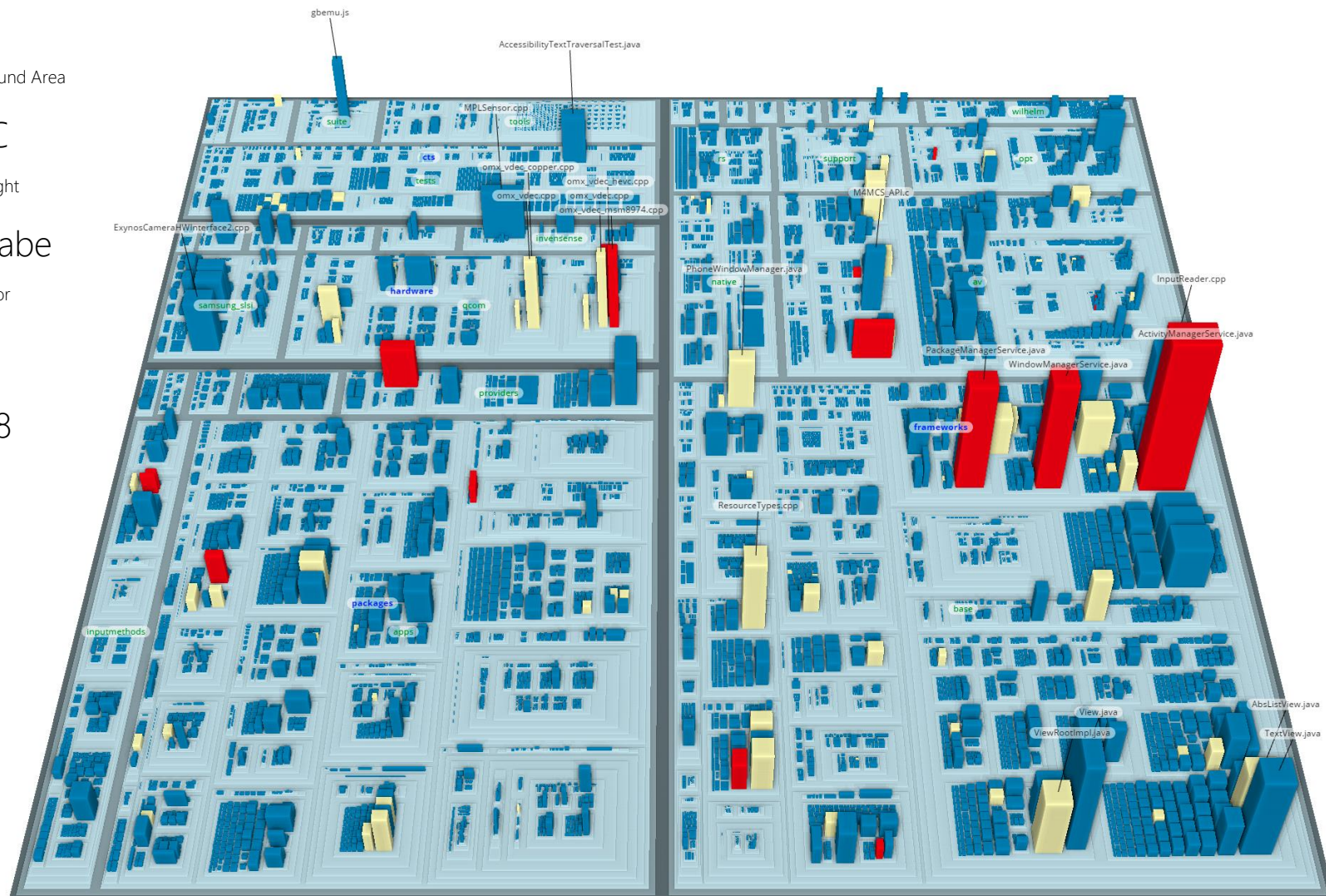


NL+

< 148

< 16

< 1



Composition of the Software Map Theme “Technical Debts”

- LLoC, mapped to cuboids' footprint size, is used to expose monolithic code as a first indicator to code that is hard to manage
- McCabe complexity, mapped to the cuboids' height, is used to indicate complex, error-prone coding, focusing on control flow complexity
- Nesting level metric (NL), mapped by a color scheme, as indicator for software complexity, focusing on the complexity due to the depth of logical nesting

Composition of the Software Map Theme “Technical Debts”

- Combination of McCabe metric and Nesting Level metric in the software map guides awareness towards modules that are highly affected by both software complexity indicators.
- High nesting-level complexity is experienced as a stronger productivity drain than high McCabe complexity. Hence, we use color as a “prioritization function” for complexity hotspots: Where both high McCabe complexity and high NL meet, program comprehension is severely aggravated.

Typical use-cases and applications

- Technical debts can be identified by this software map theme; it highlights those source code units that most likely will have a lot technical debts estimated by correlating McCabe complexity measures and nesting level measures
- Focus software development activities on highly technical debts code to reduce implied future cost due to technical debts
- Precisely apply code refactoring to reduce complexity as an investment in future maintainability of the code

Potential conclusions and way of proceeding

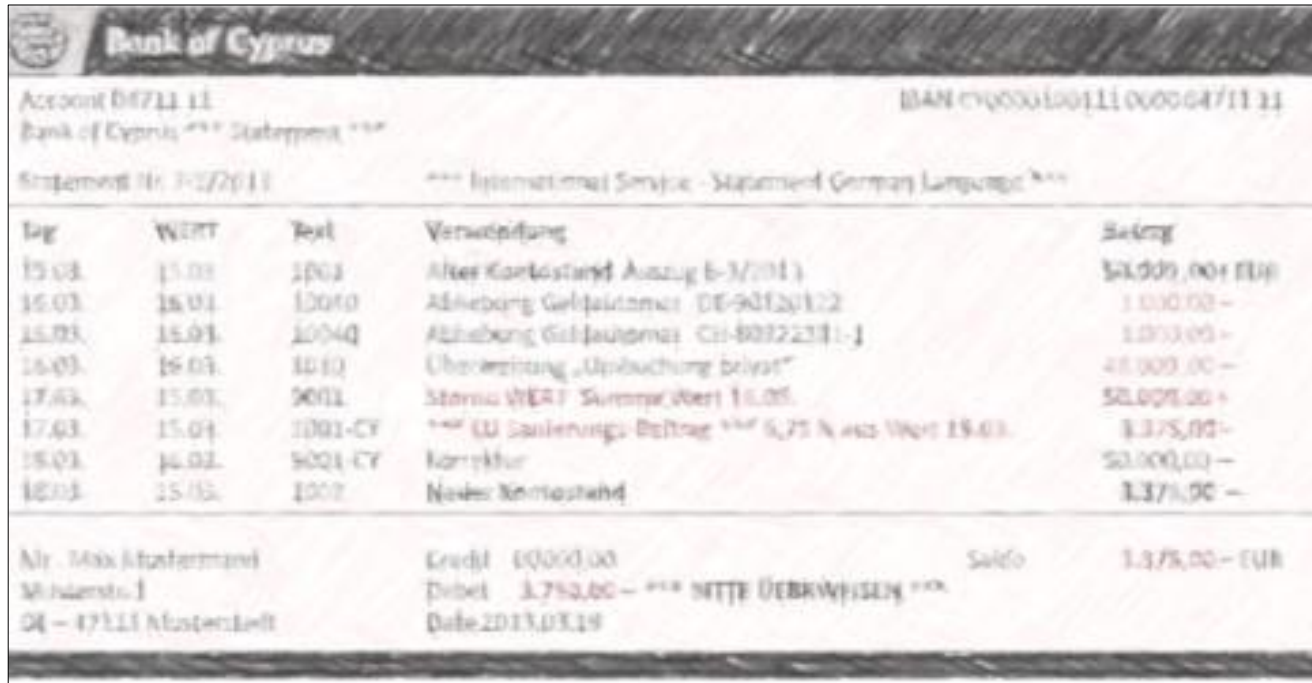
- First, reduce nesting level. Then, try to reduce McCabe complexity.

The **software map theme “technical debts”** can be applied to reveal and monitor the “technical debts” inherent to a software system’s implementation. Technical debts is among the strongest factors that influence the long-term development and its cost.

As a starting point for exploring technical debts, McCabe complexity correlated with nesting level measures are commonly a good first estimate of the technical debts.

While reducing, e.g., the lines of code in NL4+ to zero without an impact on feature completeness, one cannot reduce McCabe in the same manner – the functionality would have to be reduced. That is, the technical debts can be reduced to a certain lower bound that is determined by the complexity of the software system.

Map Theme "Technical Debts"



The image shows a screenshot of a Bank of Cyprus statement for account 00711 11, dated 3/2/2011. The statement is in German and lists various transactions. At the bottom, it shows a credit of 10,000.00 and a debit of 1,750.00, resulting in a balance of 8,250.00. The debit is labeled 'SITTE ÜBERWEISEN' (Please transfer), which is a technical debt.

Dg	WERT	Text	Versandung	Saldo
15.03.	15.03.	1001	Alter Kontostand Auszug b-3/2/11	10.000,00 EUR
15.03.	15.03.	1001	Abhebung Geldautomat DE-90120122	1.000,00 -
15.03.	15.03.	1004	Abhebung Geldautomat CH-80322311-1	1.000,00 -
15.03.	15.03.	1010	Überweisung „Überweisung beibehalten“	40.000,00 -
17.03.	15.03.	9001	Storno WERT Summe Wert 15.03.	50.000,00 +
17.03.	15.03.	1001-CY	*** EU Sanierungsbeitrag *** 5,75 % aus Wert 15.03.	1.375,00 -
18.03.	16.03.	9001-CY	Korrektur	50.000,00 -
18.03.	15.03.	1002	Neuer Kontostand	1.375,00 -
Mr. Max Musterfirma				Credit 10.000,00
Musterstr. 1				Debit 1.750,00 - *** SITTE ÜBERWEISEN ***
DE - 47111 Musterfeld				Saldo 8.250,00 - EUR
				Date 2011.03.19

Technical Debt

Software Maps Depicting "Technical Debts"

Application and Actionable Insights

Map Theme “Risk of Knowledge Drain”



Knowledge Drain
Software Maps Depicting “Risk of Knowledge Drain”
Application and Actionable Insights

Knowledge Drain Phenomenon in Software Engineering

- Knowledge drain risks are created if code is known only by a few developers.

- **Phenomenon known as “Truck Factor”**

„How many or few would have to be hit by a truck (or quit)
before the project is incapacitated?”

– John Coplien, in Pair Programming Illuminated, p. 41

- If the original developers become unavailable (e.g., leaving the company, retirement, ...), others eventually need to understand the code from scratch – a time-consuming and error-prone task.
- **Low knowledge distribution** regarding code can lead to code known to no active developer in the long run.

Revealing Knowledge Drain Potentials

- Knowledge drain in the scope of software analytics can be measured based on knowledge distribution among source code.
- **“Collective code ownership”** is one principle that ensures knowledge distribution.
- The following rules of thumb apply to measuring knowledge drain potentials:
 - Generally, small units are easier to handle (and to understand) than large units, so lines-of-code is a base weight.
 - Generally, low complex modules are easier to handle than those with high complexity, so software complexity is a effective measure.
 - Based on the development history, the number of developers active on modules should be taken into account.

Software Maps for “Risk of Knowledge Drain”

Visual Variable: Ground Area



LLOC

Visual Variable: Height



NL4+

Visual Variable: Color

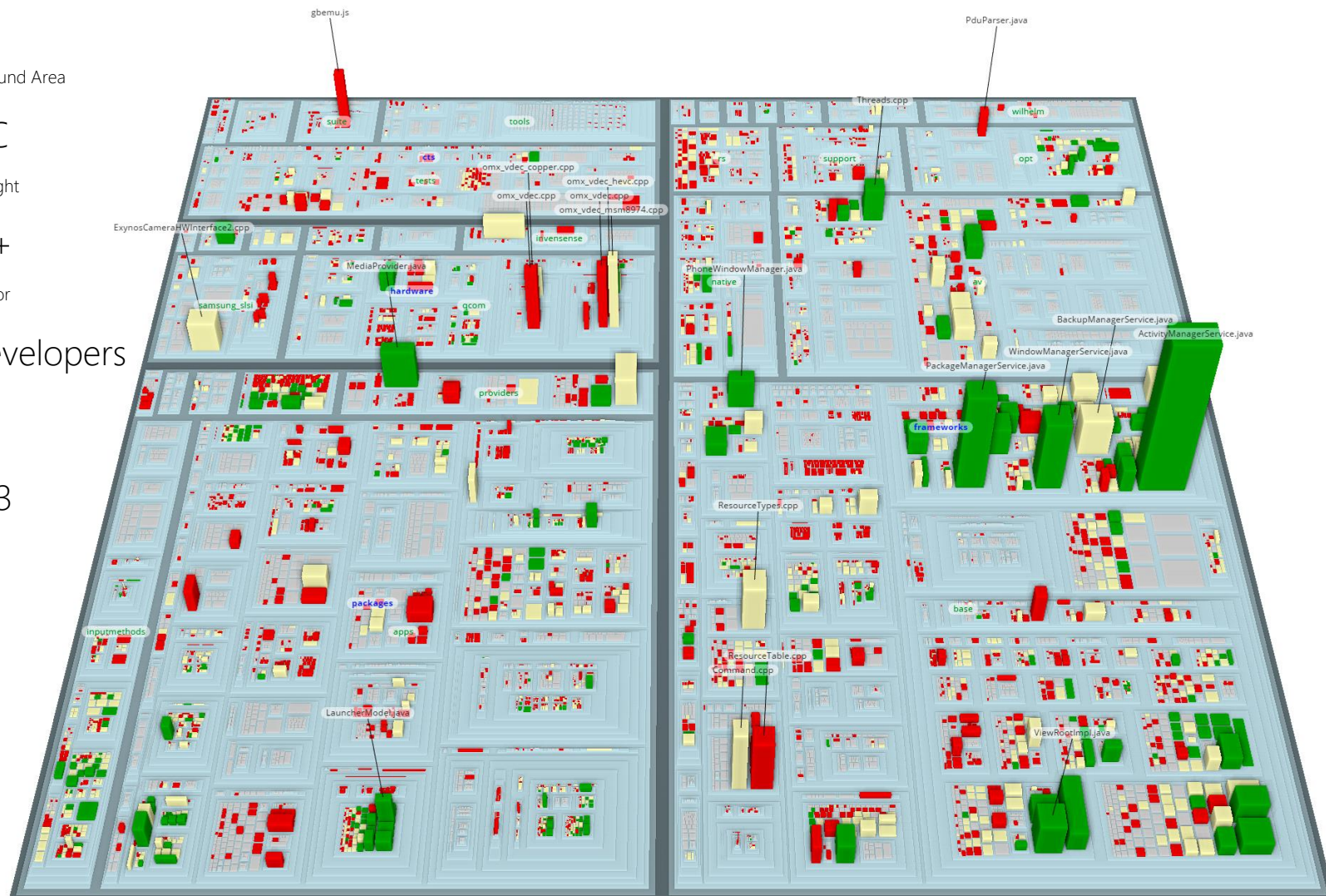


Developers

■ > 3

■ = 2-3

■ = 1



Composition of the Software Map Theme “Risk of Knowledge Drain”

- LLoC, mapped to cuboids' footprint size, is used to expose monolithic code as a first indicator to code that is hard to manage
- LoC in nesting level 4+, mapped to the cuboids' height, is used to indicate complex, difficult-to-comprehend code
- Number of developers that have been active in a given period of time, e.g., last three months, as indicator for knowledge distribution, mapped by a color scheme
- Colors are used as classification related to number of developers during a specific time frame:
 - Green: More than 3 developers have recently worked with the code
 - Yellow: Warning – only 2 or 3 developers have recently worked with the code
 - Red: Strong warning – Only one developer has recently worked with the code

Typical use-cases and applications

- Knowledge drain is a general risk in software engineering; the corresponding software maps help to identify those source code units for which a different team setting should be found.
- Knowledge drain software maps assist to effectively manage the handover before a developer leaves. As a common practice, the leaving developers should transfer knowledge to their successors at an early stage.
- “Orphan source code units” can be systematically identified and reintegrate them in the team’s knowledge pool.

Potential conclusions and way of proceeding

- Ensure sufficient knowledge distribution especially on complex code
- Introduce and apply pair programming
- Implement peer reviews
- Assign coding tasks to developers without knowledge about the code with occasional help from developers who know the code well

The **software map theme “risk of knowledge drain”** can be applied to identify complex code units known only by few developers and reveal knowledge distribution in the software system’s implementation.

As actionable insights, the software map can outline those parts of a source code that are most affected by a “bad truck factor” (factor close to 1) and, thereby, to take actions that ensure better knowledge distribution. The long-term cost for the development can be reduced.

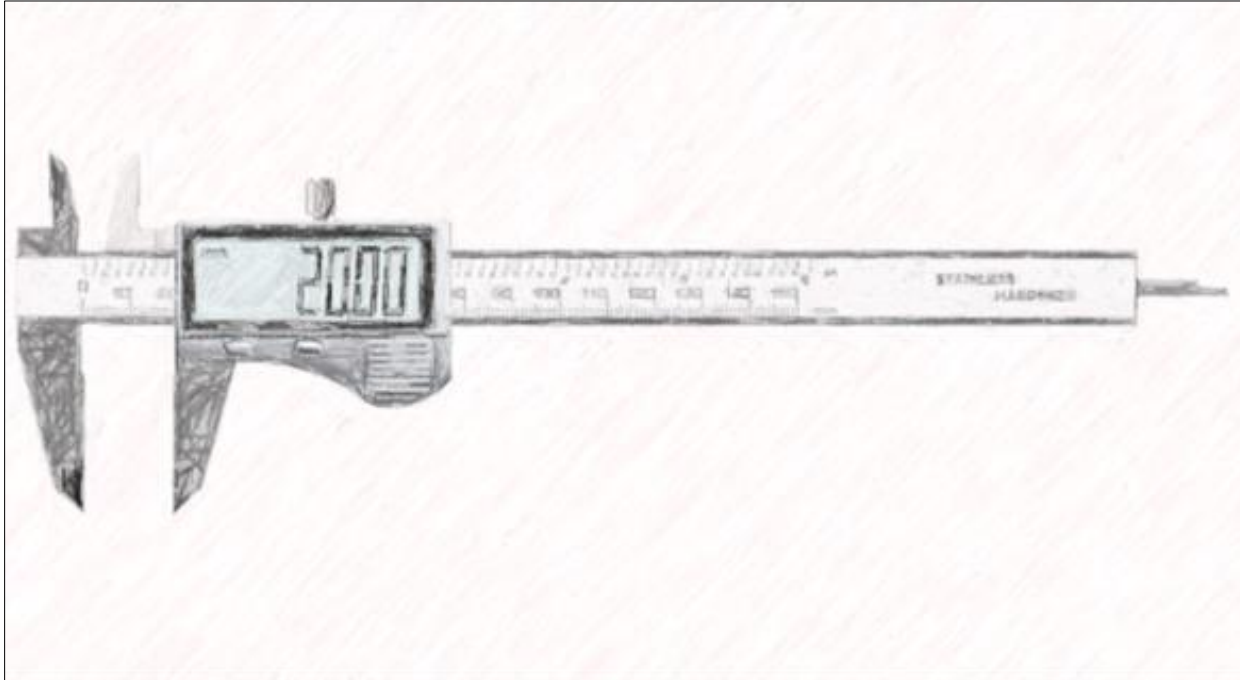
This application of software maps in the scope of software analytics also shows that the correlation of technical-structural information such as software complexity metrics with evolution-based developer metrics is a fundamental approach to cope with subtle phenomena such as knowledge-drain risk.

Map Theme “Risk of Knowledge Drain”



Knowledge Drain
Software Maps Depicting “Risk of Knowledge Drain”
Application and Actionable Insights

Map Theme "Quality Improvements"



Quality Improvements
Software Maps Depicting "Quality Improvements"
Application and Actionable Insights

Characteristics of Quality Improvements

- **Quality-improvement measures** are largely invisible to non-technical stakeholders and thus difficult to communicate and discuss.
- **Internal quality of code** is largely invisible and intangible. For that reason, it is difficult to convince the management and create awareness.
- **Effective instruments** are required that support and facilitate, for example,
 - to monitor the progress of quality-improvements
 - to assess their actual success in practice
 - ...

Quality Improvements versus Feature Delivery

- Efforts spent by developers on improving the internal quality of code is hardly recognized, and developers do not receive appreciation for their efforts.
- This leads to various consequences, for example, shifting the focus on feature delivery with decreasing care on internal code quality.
- If quality improvement cannot be communicated or is not recognized, then it will be difficult to motivate work on quality improvements.

Software Maps Depicting “Quality Improvements”

Visual Variable: Ground Area



LLOC

Visual Variable: Height



NL4+

Visual Variable: Color



Weeks with
Changes

■ = 14-21

■ = 7-13

■ = 4-6

■ = 0-3



The color encodes the number of weeks with changes during a time frame of one year.

Software Maps Depicting “Quality Improvements”



The top face color of a cuboid and the projected stripes indicate the change regarding the nesting level complexity. Yellow denotes an increase, whereas green denotes a decrease of NL measures.

Composition of the Software Map Theme “Quality Improvements”

- LLoC, mapped to cuboids' footprint size, is used to expose monolithic code as a first indicator to code that is hard to manage
- LoC in nesting level 4+, mapped to the cuboids' height, is used to indicate complex, error-prone code
- Number of weeks with changes related to bug fixes to show how frequently bug fixes have been applied to source code units, mapped by a color scheme
- Difference in height resp. nesting level to a previous code snapshot are mapped to **colored top faces and horizontal stripes**
 - Green indicates change for the better, while yellow indicates change for the worse

Composition of the Software Map Theme “Quality Improvements”

- Mapping metric value differences between two versions of the source code by means of horizontal stripes allows for assessing the progress in terms of reducing/increasing a software quality measure.
- Both the relative as well as the absolute differences can be assessed.

Typical use-cases and applications

- Monitoring quality improvement over time
- Communicating the result of efforts spent on improving the internal quality to the management

Potential conclusions and way of proceeding

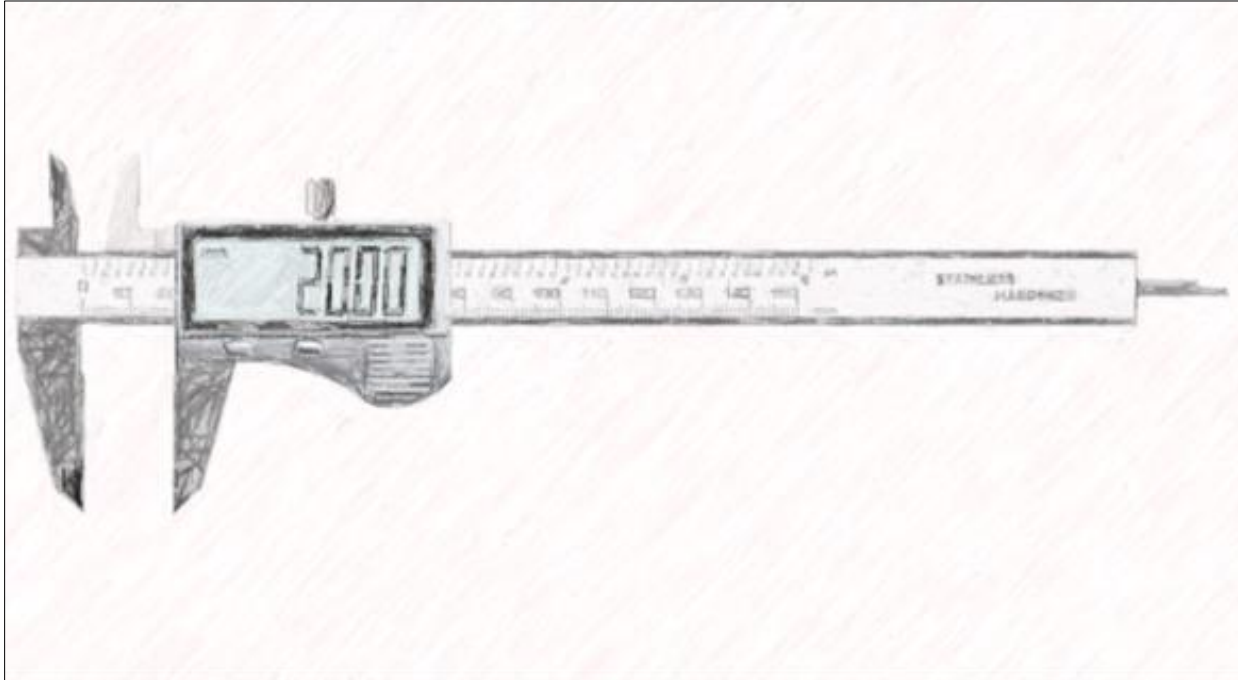
- Support the management of long-term software quality improvements
- Assess progress and control whether the development trends are still aligned to the software quality goals
- Document effects of investment in software quality
- Communicate success and progress of quality improvement initiatives to the management

The **software map theme “quality improvements”** can be applied to monitor, evaluate and communicate the status and trend of quality improvements as well as the effect of efforts spent to quality improvements.

With this software map theme, in particular, we can precisely show the progress of implementing measures to improve quality with respect to appropriate key indicators such as different software quality metrics.

As actionable insights, this type of software map allows us to intervene in the case adopted measures to quality improvements are not on track. We can also effectively control that teams actually make progress. This software map theme furthermore helps to decide whether the overall software quality of a system is up to standard.

Map Theme "Quality Improvements"



Quality Improvements
Software Maps Depicting "Quality Improvements"
Application and Actionable Insights