Code and Concepts: Exploring Scientific Code with GICAT

Gabriele Gramelsberger, Daniel Wenz, Markus Pantsar, Dawid Kasprowicz, David Heyen

February 26, 2026

Whether for data-driven models or computer simulations, developing and using of scientific software has become an indispensable part of science and the humanities. Facing the complexity of scientific code, we developed a software tool (GICAT) to analyze data-structures, code layers and code genealogies. In this paper, we combine a description of the functionality of GICAT with a genealogical study of the transformation of scientific concepts into their encoded counterparts. of philosophy of science and science and technology studies (STS).

1 Introducing GICAT

"GICAT" is short for "General Isomorphic code analysis tool". In this first section we state the motivation for its development and demonstrate some of its key features with a simple example.

1.1 Motivation – Scientific code as research object

The rise of computer-based simulations and artificial intelligence (AI) tools, has transformed, and continues to transform, scientific research. New methods like big data analysis and machine learning (ML) have enabled applications that have brought important changes to scientific methodology. These include large-scale simulations, generative AI like ChatGPT, and the increasingly effective use of automated theorem proving software. This influence is not limited to the natural sciences, as also the humanities face

changing methodologies. In modern research, the computer can either substitute or heavily alter the classical work in the laboratory and the archive, even changing how field studies are done. The use of computers has expanded the three nuclei of every scientific endeavor: observation, experiment and theory building. This development has led to coining the terms "in-silico" and "drylab" research (Merz 2006).

Observation, as the empirical base of science, is something done by the trained scientist. Although augmented by technical devices since the dawn of modern science, it is thought to be a domain where the human scientist is irreplaceable. This can happen by "purely" observing things in a systematic way, but often it is augmented by tools like telescopes and measuring devices, or by practices like identifying the traces in a bubble chamber as specific particle tracks. In each case, however, the human plays a central role: making science without a human observer seem like an oxymoron akin to knowledge without an epistemic subject. Nevertheless, today particle tracks are computer-(re)-constructed via an analysis of automatically recorded data, where already this recording process heavily relies on machine learning (Falkenburg 2007, pp. 77). The use of CCD-chips in telescopes, beginning in the 1990s, transformed astronomy into a data-driven science that revolves around the analysis and creation of masses of photometric data (Hoeppe 2014). In the humanities, doing research in an archive today means in many cases to apply a search engine to a digitized library. The ability of computers to search for and to compare patterns in huge amounts of texts has transformed comparative studies in many fields, such as literature, religious studies, sociology and philosophy, as well as making new disciplines like corpus linguistics on a grand scale possible in the first place.[1]

Philosophically, one consequence of the rise of methods like numerical simulations has been calling into question the distinction between theory and experiment. Some authors even argue that this new way of doing science, rather than just changing the classical relationship between theory and experiment, actually begins to replace theory-based research altogether (Lazinger 2017). There is a vibrant discussion about the ontological status of computer-based simulations: is simulation "experimenting with theories" or is it a third and autonomous form of knowledge production (Dowling 1999, Winsberg 2010, Gramelsberger 2011, Gramelsberger 2010)? Can simulation- and ML-based knowledge production be transparent and reproducible, or is its epistemic status inescapably "opaque" (Humphreys 2004, Lenhard 2019)? In this context, simulation- and ML-based methods are sometimes compared metaphorically to telescopes and microscopes providing a new "type

[1] Starting with the collaboration of the Jesuit priest Roberto Busa with IBM in 1951 to create the Index Thomisticus (Winter 1999).

of empirical extension, augmentation, [...] beyond the reach of traditional mathematics" (Humphreys 2004, p. 5).

Industry-driven research, in particular, uses these new technical means. It is easier to construct and test new car designs trough computer modeling and simulations than to actually construct real life prototypes and test them in wind chambers or expensive crash tests. Most crash tests today, for example, are done to validate and fine-tune models and simulations, rather than to directly test a specific car model. Theorem provers, originally developed by and increasingly used as tools for mathematicians, are now an indispensable tool in software development. Without them, software validation (the test that a piece of software actually behaves as it is supposed to) of complex programs would not be possible.

Even scientific writing is impacted: parts or even whole papers in scientific research can be written, or at least drafted, by systems of generative AI like ChatGPT. This practice is quasi-sanctioned by the publication of handbooks how to do so by respected scientific publishing houses (Han, Qiu, and Lichtfouse 2024). However, the use of AI tools is difficult to detect and thus unrealistic to control, thus making it likely that in the near future an increasing amount of scientific writing is done by AI systems.

These developments change not only how science and scientific research is done, but also how and by which means we have to think about them. To be able to articulate (and possibly answer) epistemological questions concerning modern day science presupposes the ability to engage with computer programs in more than a superficial manner. It is safe to say that in order to do philosophy, or any kind of critical reflection, that involves computer-augmented sciences, one needs to grasp the workings of the code on some level, similarly to how one needs to find her way around the mathematical apparatus employed by, e.g., particle physics in order to say anything philosophical useful about electrons. Instead of writing about code from an external and often aloof point of view, it becomes more and more important to be able to engage with the actual code applied by scientists. Just like models and new forms of mathematical representations have shaped concepts and objects of scientific reasoning, programming languages and practices can change the very foundations of scientific methodology.

Finding inspiration in recognizing the importance of code for philosophy of science and science and technology studies (STS), we developed the General Isomorphic Code Analysis Tool (GICAT). GICAT visualizes different logical layers of a given software project. From

class and inheritances relations in object-oriented languages or complex inter-dependencies in functional programming to library imports, GICAT can generate diagrams of any given code structure. Other existing programs like Doxygen[2] are limited in the variety of structures they can depict, in addition to being limited to specific programming languages and to a specific time frame.[3] In contrast, GICAT allows the user to search every kind of code for a potentially unlimited number of structures due to its employment of regulars expressions. Due to its fluid nature, GICAT is thus not limited to any programming language. It works with different filters for the nodes and edges of the diagrams that, once pre-loaded, can be switched on and off in real time. GICAT comes with a set of standard filters for different programming languages (see section 2.3 for examples). It also includes a powerful filter generator, allowing the user to explore the targeted code freely. The integration of a code viewer enables the user to look directly into the code snippet that each node of the generated diagram represents via a simple click. Together, these features make GICAT a powerful tool that enables the user to freely explore the code in real time, switching between different logical layers and zooming in and out, from a structural overview of a whole project into its different parts and layers down to the corresponding strings of code.

In addition, other comparable programs are dependent on existing meta-data, and thus only work on newer code. GICAT, on the other hand, can be applied to any code, regardless of the syntax or age (as long as it is digitized). This is an important advantage not only for software historians, but for everybody interested in scientific developments documented in the evolution of software.

1.2 An introductory example

Already in the 1970s, the use of computers shifted astronomy from observing the sky with telescopes to data visualization analyzing images of the sky (Daston and Galison 1992). Since the 1990s, the use of CCD-chips in telescopes has shifted astronomy into a data-driven science by generating masses of photometric data (Hoeppe 2014). Effectively, this means that analyzing data sets has become central for today's astronomy. One of the huge databases used is the Reference Catalogue of Spectral Energy Distributions (RCSED; Chilingarian et al. 2017). It contains the photometric data on energy distributions of 800,299 Galaxies. Two of the main types of information that the RCSED includes concern

[2] https://www.doxygen.nl/.

[3] This means in many cases such programs cannot be used to analyze older code of programming languages they support. This severely limits their use for any kind of long-term developmental studies.

the light emitted directly by the stars and the light emitted by the gas content of galaxies. The latter is described by so-called emission lines. Some of these emission lines show a specific graphical feature, a double peak, which is an indicator that two or more galaxies are merging, i.e., colliding into each other. One of the interesting aspects about such merging galaxies is that they tend to be associated with a high star formation rate (SFR). The SFR of a galaxy, in turn, can be used to diagnose the evolutionary state of a galaxy. This makes the search for double peak emission lines part of an empirically based investigation into galaxy evolution theory. For this purpose, special software has been developed. Here we focus on xgaltool, a program devised by the astrophysicist Daniel Maschmann (Maschmann et al. 2020) that searches the RCSED for double peak emission line galaxies.

The following graphs were generated by analysing the code of xgaltool with GICAT. If we look at figure 1, we see the overall folder structure of xgaltool.

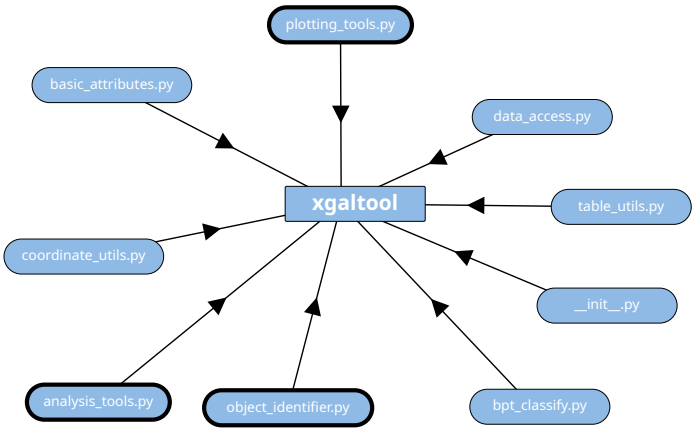

Figure 1: xgaltool simple folder structure

In this case, a look at this superficial level conveys some information: plotting_tools, analysis_tools, and object_identifier already hint at their role in the overall architecture of the program. We can also distinguish between scientific content in the folders and non-scientific content like, for instance, the plotting tools. As we examine a project written in the programming language Python, it seems to be promising to look at its class structure since classes are the place where objects are created, and languages like Python are object-oriented. Figure 2 depicts the same set of data under the class filter.

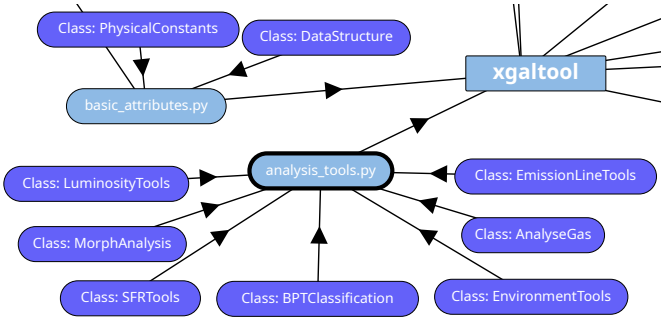

Figure 2: xgaltool class structure

By double-clicking on one of the nodes, the user can launch an integrated code viewer that shows us the relevant strings of code (figure 2). If the user wants to generate an image for publication or sharing, she can pause the simulation. Then she can drag the nodes to optimal positions and highlight individual nodes and edges by simply clicking on them. The resulting image on the screen can then be exported as a vector graphic file (.svg). Vector graphics can be zoomed in and out without loss of quality and are the standard in modern scientific publication.[4]

```
class AnalyseGas(EmissionLineTools, SFRTools):

    def __init__(self, **kwargs):
        r"""

        """
        super().__init__(**kwargs)

    def compute_co_luminosity(self, s_co_vel, luminosity_dist=None, redshift=None,
                              observed_co_frequency=None,
                              conversion_factor=1, index=None):
        r"""
        Compute the CO emission line luminosity. We use as default the CO(J1->0) line.
         following Solomon et al. 1997 APJ 478 : 144-161, doi:10.1086/303765
          Using EQ. (3) in Sect. 3.1 for the luminosity
          (see also Saintonge et al 2011  doi:10.1111/j.1365-2966.2011.18677.x EG. (4))
```

Figure 3: xgaltool code class structure

We will return to this example in section 3.3, but we have already gotten an idea of the kind of information that analysis with GICAT provide. Of course, such an analysis has to be guided by a certain knowledge of the subject matter (in this case astrophysics), the paper behind the project, and the programming language. But the analysis also needs to

[4] If you are reading a digital version of this article, the embedded pictures are .svg files.

be augmented by specific tools that allow the user to look at the different structural and logical layers of the code. This is what GICAT makes possible.[5]

2 Graphical representation of code

The overall framework that GICAT is build on is Electron (https://github.com/electron/electron), an open-source framework that allows for cross-platform development. This enables us to provide and update GICAT on a regular basis for all the major platforms (Windows, macOS and Linux). For the front end, we use Vue 3 (https://github.com/vuejs), an open-source JavaScript framework. To draw the actual graph from the information extracted by the regular expressions, we use the vector graphic based v-network-graph (https://dash14.github.io/v-network-graph/). The interactive behavior of the graph is determined by the physics engine d3-force (https://d3js.org/d3-force).

The method we use is a text search via *regular expressions* with the code as its base. On this foundation, filters are generated that determine the properties of nodes and edges in a graph which provides the user a dynamic visualization of the structure of the code. The adaptability of regular expressions is the reason why GICAT is so flexible. A regular expression, simply put, is a string of characters that fulfills certain criteria. When applied to a text, the regular expression filters the code and finds all patterns matching the criteria. This seemingly simple method is a powerful tool in the hand of a skilled user, but previously it has demanded considerable expertise on coding. GICATs filter generator and the instant feedback through its visualization opens up the potential of regular expressions to the non-expert. The user is able to generate her own filters according to her own needs by generating and modify regular expressions from a given code snippet, and then refine them in an ongoing and easy-to-handle feedback loop. Importantly, as mentioned above, this method is not restricted to any specific programming language or paradigm.

2.1 The baseline graph

If the user starts GICAT's visualization, the function startVisualization() will be executed. This generates a JSON object called "graph" according to the information generated by the function getGraph(). This baseline graph includes the information about the folders

[5] For a full version of this specific case study cf. (Gramelsberger, Wenz, and Kasprowicz 2024). See also section 3.3 below.

in the target directory and all the files nested in them.[6] If no further modifications are chosen (i.e., no filters are defined and activated by the user), the graph-object is committed to a store, a specific domain model that makes the data available to other components of the tool. One of these components is NetworkViewNew.vue, which utilizes v-network.graph to generate the visualization as the final output for the user. In effect, this results in a simple graph that depicts the folder and file structure of the target code in an isomorphic manner (as seen in figure 1 above). If the user chooses a filter package before starting the visualization, the search routines of the corresponding filter packages are run over the code. As explained above, these search routines are based on regular expressions. This process creates additional information that is then committed to the store which leads to new nodes, edges and labels in the resulting graphical representation (as seen in figures 2 and 10). In the following section, we describe in more detail how these search routines work, what regular expressions are, and how they can be applied to define filters for GICAT.

2.2 From regular expressions to dynamic graphs

A regular expression like $class\backslash s([A-Za-z]+)\backslash((.*)\backslash)\backslash:$ articulates a search pattern. It functions similar to a wildcard but is much more expressive. A wildcard is a symbol like ? or $*$ that stands in for a certain number of characters: ? stands in for exactly one, $*$ for an arbitrary number of characters (including zero). Wildcards are widely used in command-line commands. A command like $copy*.pdfto/home$ copies every pdf file (i.e., every file that ends with the suffix .pdf) in the current directory into a folder called "home". If the current directory entails the files $readme.pdf$, $licence.pdf$ and $a.pdf$ executing the first command would copy all three files into the home folder while the second would only copy the file $a.pdf$. If you search a body of text that entails the words "anger", "danger", "stranger", "ranger" with $*anger$, the search collects all four words. If you search the same text base with ?anger the result will be "danger" and "ranger". A text search is much more effective and flexible if it is possible to combine commands that collect strings with commands like ! that omit specific symbols. Searching the text base from above with $*[!r]anger$ gets the output "anger" and "danger". This example also shows how squared brackets [ ] can be used to nest search patterns. Roughly put, $*[!r]anger$ defines the set of strings that include the specific sequence "anger" that is preceded by a arbitrary sequence

[6] This process utilizes the dirTree library which creates a JavaScript object representing a directory tree (https://www.npmjs.com/package/directory-tree). Labels corresponding to the file names are assessed to the nodes.

of symbols that does not include the symbol "r".

The wildcard $*$ introduced above is called the Kleene star. The mathematician Stephen Cole Kleene is generally considered to be the inventor of regular expressions as a concise descriptive tool. The birth of what we call now regular expressions can be found in (Kleene 1951).[7] The paper is one of the first works that develop means to describe the "logic" of artificial neural networks (building on early ideas of McCulloch and Pitts 1943), here called "nerve nets". In chapter 7, Kleene introduces the concept of a "regular event" to describe dynamic and complex but regular behavior in such nerve nets. The key observation here is that applying these descriptions is not limited to nerve nets: Kleene showed that for every possible regular event there exists a finite automata and vice versa.[8] This fact enabled regular expressions to become one of the cornerstones of programming and how we interact with computers in general.

The regular expression $class\backslash s([A-Za-z]+)\backslash((.*)\backslash)\backslash:$ from above can be deciphered in the following way: $class$ means the actual string "class". $\backslash s$ means that this string must be followed by whitespace. The next section is a complex arrangement articulated with nested brackets, whereby the squared brackets define character sets, while the round brackets create capturing groups.[9] So $[A-Za-z]$ allows symbols from from the alphabet including upper and lower cases. The $+$ after the closing squared bracket in the capturing group $([A-Za-z]+)$ quantifies this from at least one to arbitrary many characters. $\backslash($, $\backslash)$ and $\backslash:$ mean the symbols "(", ")" and ":". The dot followed by the star in the capturing group $(.*)$ means zero or arbitrary many characters (except line breaks). A match for our regular expression would then be:
class AnalyseGas(EmissionLineTools, SFRTool): or

[7] This has been partially reprinted in Kleene 1956. We cite the original report which has now been made available online by the RAND Corporation.

[8] "Our objective is to show that all and only regular events can be represented by nerve nets or finite automata" (Kleene 1951, p. 46). An "event" is defined as the description of the pattern of the activation of the input layer of a nerve net over a period of time (ibid., p. 9). Consider a network that consists of two input neurons n1 and n2 that exhibit over the period t1-t4 the following activation pattern:

$t1:\quad n1=1\quad n2=0$
$t2:\quad n1=1\quad n2=1$
$t3:\quad n1=0\quad n2=1$
$t4:\quad n1=1\quad n2=1$

Then (among others) the following four sentences describe events:
(1) "n1 fired at t1";
(2) "One of n1 or n2 fired at time t1";
(3) "n2 fired at some time p";
(4) "n2 fired at every time except t1".
Regular events are then defined as a specific kind of events in what we now call neural networks.

[9] A capturing group unifies a part of the regular expression so that an operator like a quantifier can be applied to that unity as a whole.

class AnalyseGas(): but not
class (EmissionLineTools, SFRTool): or
classAnalyseGas(EmissionLineTools, SFRTool):.

Applied to xgaltool as in the example in section 1.2 we get as matches (among others) $classAnalyseGas(EmissionLineTools, SFRTool)$:, $classSFRTools(data\_access)$: and $EmissionLineTools(data\_access)$:. This is already enough information for a simple class filter (cf. figure 2). One can also already see from these expression that the first match (AnalyseGas) somehow "quotes" the other two (SFRTools and EmissionLineTool). This information is used in the class extension filter, as can be seen in figure 10. In this way, regular expressions can be used to mine the targeted code for information, which is then added to the store. Based on this information, additional nodes, edges and labels are created in the graphical representation.

2.3 Nodes and Edges – Examples of different filters

GICAT comes with a large set of filter packages for different programming language. In this section, we briefly introduce some of them and give short examples of their possible applications. A filter package consists of different filters that visualize different code structures of the same programming language. But more importantly, GICAT has an integrated and easy-to-use filter generator. The generator helps to articulate regular expressions from a given snippet of code and automatically transforms them into node and edge filters. The filter generator is highly flexible and works (as it is text-based) with code from any programming language. GICAT also provides many options for graphical customization (like different color schemes for nodes and edges etc.) and lets the user export and share her custom-made filter packages. In the examples shown below, basic folder structures of the base code are depicted as rectangles. The light blue nodes represent the corresponding code in these folders. Everything identified by a filter is colored either dark blue or red.

Fortran90 filter (submodules): Neural-fortran is a deep learning framework, written in Fortran90. Figure 4 shows every sub-module in dark blue. It also has the capacity to identify and represent recursive functions.

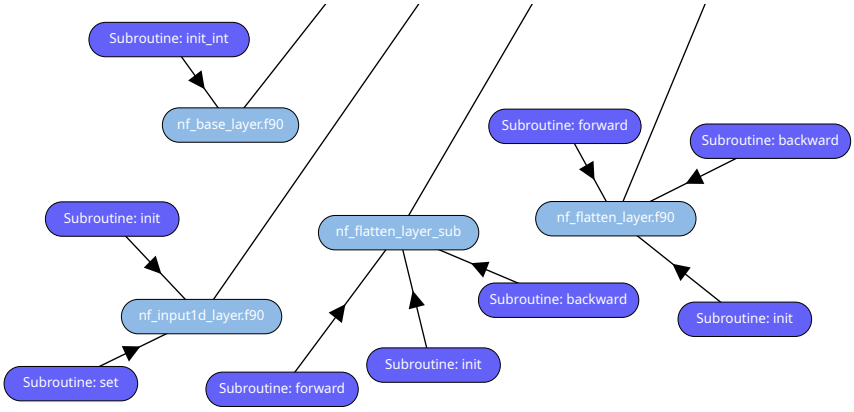

Figure 4: filter Fortran90

Python filter (class extension): https://gitlab.obspm.fr/dmaschmann/xgaltool Xgaltool was written by a former colleague of the authors, Daniel Maschmann. This project was our very first case study. It is a tool from astrophysics for detecting merging galaxies in the reference catalog of Spectral Energy Distributions (RCSED). Figure 5 shows class inheritance structures. This example was discussed in more detail in section 1.2.

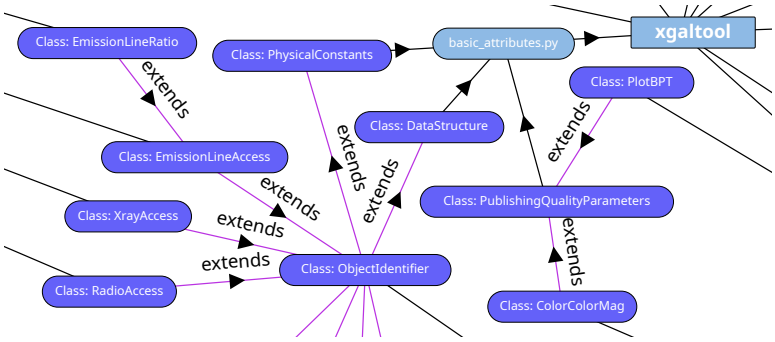

Figure 5: filter Python class extension

Java filter (class extension): GitHub Repository: https://github.com/apache/opennlp Apache OpenNLP is a toolkit for natural language processing (NLP) that applies methods from machine learning (ML). Figure 6 shows the implementation of the training formats. Classes and their inheritance structures are represented by the red edges between the red nodes.

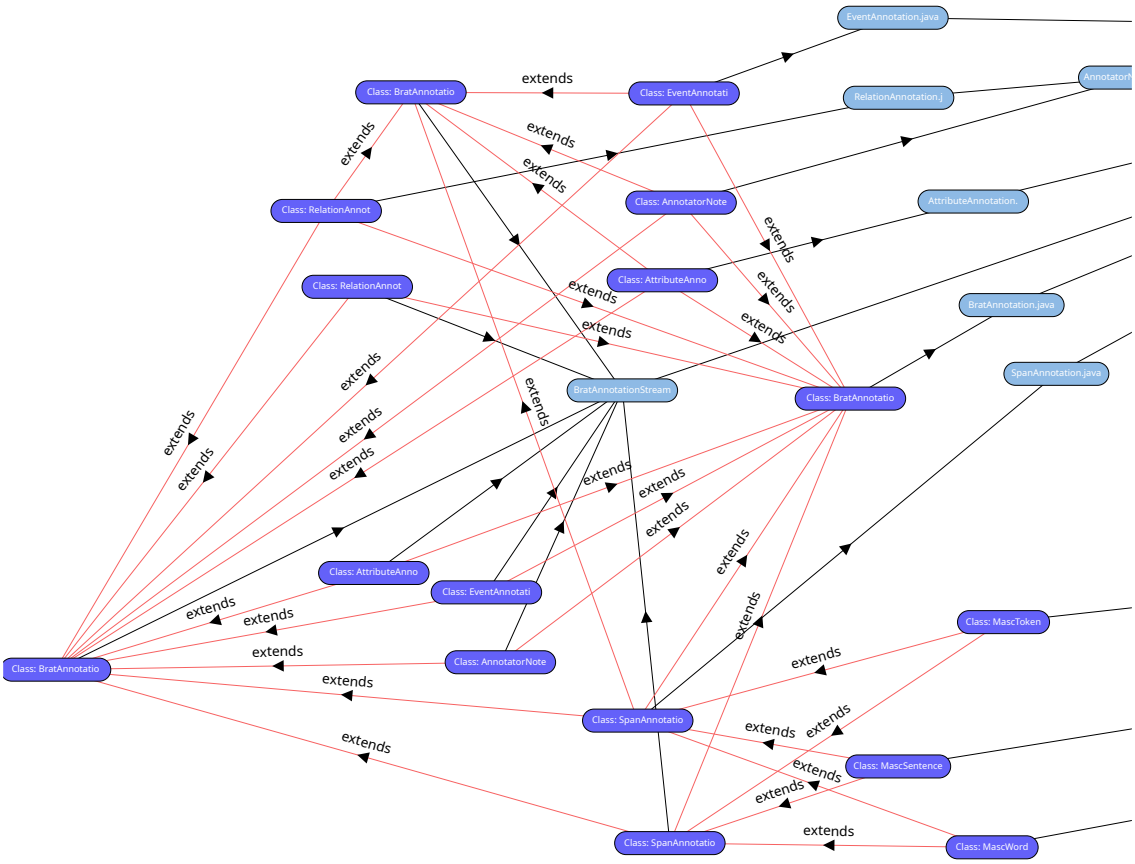

Figure 6: filter Java class extensions

Java filter (lambda functions): https://github.com/thingsboard/thingsboard/tree/master Thingsboard is an open source-device management and IoT platform written in Java and Javascript. Figure 7 identifies and visualizes a lambda function. (The relevant folder is monitoring/src/main/java/org/thingsboard/monitoring).

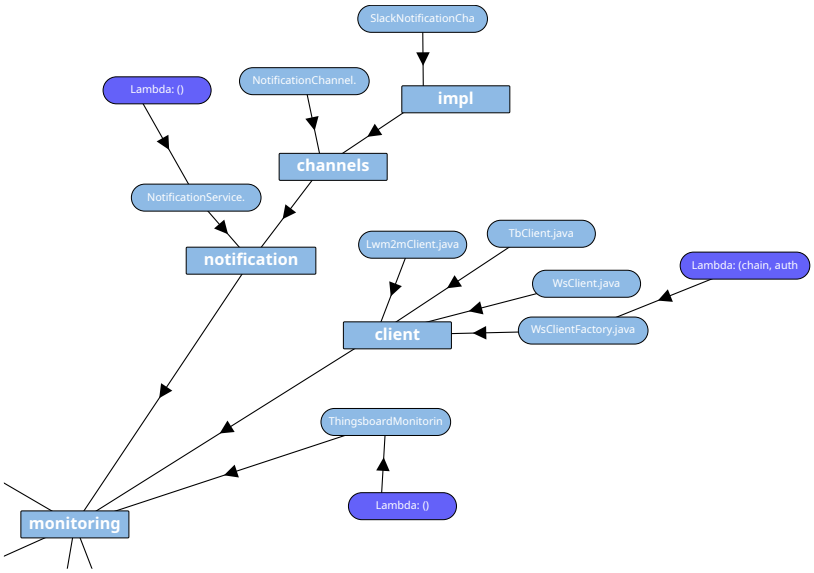

Figure 7: filter Java lambda function

Lean filter (imports; self-defined structures): https://github.com/leanprover/lean4/tree/master Lean is a theorem prover developed on an open source base by Microsoft. Lean is written partially in itself (i.e., the prover is also a programming language) which makes it an interesting object to study. We applied GICAT to the path that contains the compiler. Picture 5 depicts module imports in dark blue and every self-defined structure in red.

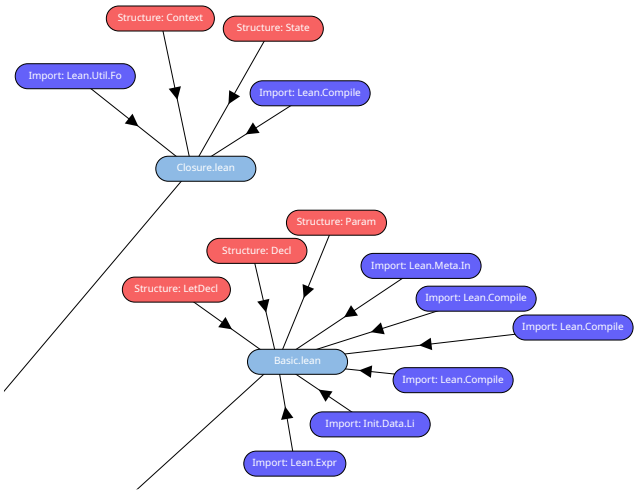

Figure 8: filter Lean imports and self-defined structures

3 GICAT as a means of discovery

GICAT provides the researcher with the means to navigate through scientific code. It is a tool that gives the user the ability to orient herself in the intricacies of the technical maze constructed by the programmer in her attempt to consciously embed a scientific theory in a piece of software. It is also a tool of discovery that can give access to 'hidden' layers of the code that emerge through the combination of technical conventions and necessities of programming and the actual scientific content that they are used to implement. These layers can encode new methods and concepts that are not thought up consciously either by the programmer or the scientist (or the scientific programmer) but that nevertheless somehow emerge in the code. It is not improbable that even novel scientific concepts (and methods) could emerge in this way. Especially in the exact sciences, it is a regular phenomenon that the mathematical apparatus that is employed in the articulation of a series of experiments had to be interpreted in a way that changes the theory those experiments rested upon. Sometimes this has lead to imposing a (often idealized) representational significance to parts of the mathematical apparatus originally only introduced for computational convenience. Prominent examples are the introduction of the concept of point mass as an interpretation of Newton's mathematical formulation of the laws of motion in his Pincipia (Newton 1687), the (diverse) interpretations of the mathematical apparatus employed in quantum mechanics (Heisenberg 1927, Reichenbach 1944, Bohm 1952, Bohr 1958) and the general tendency

to interpret statistical mechanics in a representational manner (associating statistical distributions with "real distributions" like distributions of particles etc, cf. (Galison 2011)). Of course the readiness to accept mathematics as a reliable guide to track down new scientific concepts rests on a presupposition that is deeply entrenched in modern science: that, as Galileo put it in a popular phrasing,[10] mathematics is the "language of nature". We would not be quite as ready to announce that the book of nature is written in a programming language - but certainly the heavy employment of computer programs in science and the fact that a lot of the mathematical apparatus is realized through these programs gives us confidence that something new and epistemologically interesting can be discovered also in the webs and layers of scientific code.

Before we can set out to discover something, however, we at least need a general preconception of what we are looking for. As scientific code somehow implements a scientific theory, and since every theory is at its core based on concepts, a good starting point seems to be the question how scientific concepts get into the code. This holds true no matter how much mathematics is involved in the actual articulation of a theory. That leads us to the general question: what are scientific concepts and how did they get into the code?

3.1 What are scientific concepts and how did they get into the code?

The history of scientific concepts goes back to the dawn of philosophy and science, and as such it predates Plato's and Aristotle's specific interpretation of the words "technē" and "epistēmē". Concerning the origin of scientific concepts, it is safe to assume that most are refinements (or combinations) of concepts that were already in circulation. An early example is the aforementioned "technē", which Homer uses as a general term for skill and competence before it was developed into a technical term by Aristotle (Löbl 1997–2002). From the beginning, scientific concepts have combined two very different and *prima facie* incompatible qualities: exactness and (systematic) ambiguity. Plato's complementary techniques of "dihairesis" (division) and "synagōgē" (bringing together) were methods of giving already existing concepts an exact meaning by generating a taxonomy defining their interrelations. Aristotle not only expanded and streamlined this approach, but also devised a general theory of deductive reasoning in such taxonomies. This gave rise to a long-lasting paradigm of a semantically grounded combination of systematic ontology and

[10] It is hard to actually trace this down to Galileo. A starting point would be Galilei Saggiatore 6 rep: Galilei 1960.

logic (cf. Cassirer 1910, 3ff). This constellation can be found in different forms throughout history – from antiquity through the scholastic thinkers of the middle ages up to modernity (Grant 1996). Our contemporary sciences that rest essentially on mathematical formulae and systems of derivation to articulate their core concepts can be seen as the pinnacle of this development. The other aspect, the ambiguity of scientific concepts, also preserved itself through all the modifications and transformations of the scientific project. This ambiguity is partially due to the origins of scientific concepts in ordinary language. The inherited semantic surplus allows bridging the gap between different theories that are meant to describe the same phenomenon and ensures the safeguarding of the conceptual core of a theory during different stages of its development. It also enables the discovery of new approaches and methods as a guiding thread and safeguard (cf. Wilson 2017, pp. 1–51). The other source of ambiguity of scientific concepts is actually co-dependent on their systematic reformation and clarification. There has always been fluctuation between concepts used in theories and concepts that are used to build and discuss theories. Aristotle's concepts "energeia", "dynamis" and "kinêsis" denote something he talks about in his Physics (and Metaphysics) but he also employs them to reflect on the theories put forward here. This kind of ambiguity enables us to discuss a theory without completely leaving it. This violates, it should be noted, the distinction between internal and external questions (restricting the use of concepts to the framework they are part of) put forward famously in Carnap 1950.[11] That distinction notwithstanding, it is hard to see how our modern concepts of "force" and "energy", for example, could have emerged without such transgressions. The tension between exactness and ambiguity of scientific concepts can be witnessed in action throughout the 20th century, a period that at first glance highlights the aspect of exactness.

3.2 Concepts, Theories and Code

In 1902, Henri Poincaré (Poincaré 2017) remarked that the meaning of a scientific concept is defined by its role in a theory. In this approach – which was refined by Duhem in 1914 (Duhem 1954) and famously applied by Quine[12] in the 1960s (Quine 1960, Quine 1969) – a scientific concept *per se* is not directly explainable. Instead, the approach

[11] Immanuel Kant gave similar warnings in an appendix to his *Critique of Pure Reason*: Von der Amphibolie der Reflexionsbegriffe, A260/B316 ff (Kant 1998); among the concepts he talks about are Aristotle's "hylē", matter and "morphē", form. His critique is mainly formulated as a critique of Leibniz rationalism.

[12] In a broader sense concerning a plurality of conceptual schemes and their respective objects as well as the possible mediating role of a formal mode of language after what Quine calls "semantic ascent" (Quine 1960, chapter 7).

addresses this question indirectly by asking the question concerning the origins of the meanings of concrete scientific concepts. The resulting picture in which the meaning of a scientific concept is defined through its role in a theory allows identifying two main sources for the change of content in a scientific concept. The first source consists of permanent modifications of a theory, the second of temporary modifications of some aspects of the theory, in order to make it applicable to a specific situation. The latter kind of modification typically concerns parametric modifications of parts of the theory in experiments and in real world applications. Here the concepts prove their worth by predicting or bringing about specific outcomes from a set of given starting conditions. However, the starting conditions and the outcomes are always interpreted and evaluated in the context of the respective theory.

In philosophy of science, three developments have undermined this classic perspective. In the first one, the clear distinction between theory and experiment according to which theory leads and the experiments follow (cf. Popper 1959) got blurred. This was not (only) done by an intricate philosophical argument, but also by analyzing the actual scientific practice (Hacking 1983, 149 pp.). The second development was that the propositional or syntactic view of theories (Carnap 1937, Hempel 1965) (viewing a theory as a set of axioms) was gradually replaced by the semantic view. According to the latter, scientific theories are first and foremost models (Suppes 1960, van Fraassen 1980). The idea is that instead of seeing a specific scientific concept determined by one specific theory (implicitly defined by a set of axioms), the content of such a concept can be grasped trough the sum of the models it figures in (i.e. the "family resemblance" of the operators that represent it in the respective models) (cf. ibid., 41 pp.). Based on this picture, scientific concepts, which were at the beginning of the 20th century conceived as paradigms of unambiguity and exactness, came to be seen as evolving entities that not only secure and handle accumulated knowledge, but through their flexibility open up paths for new investigations (Wilson 2006, Brandom 2010, Bloch-Mullins 2020). With the rise of the computer model in science, the content of scientific concepts is spread even further apart. This is due to translational reasons and their specific technicalities: a mathematical formula is not identical with its implementation into the code. One of the most pressing problems is the translation of mathematical models as used in the semantic view of theories into numerical (computable) models. In more complex cases it is not even clear if the numerical model really instantiates the mathematical model of the underlying theory (Gramelsberger 2011).

All this can be expected to lead to repercussions on the level of the scientific concepts expressed by the theory. Models get more and more independent from the theories they are associated with (Morrison and Morgan 1999). In extreme cases the development of the mathematical model and the development of the computer model can split up into different projects that only occasionally interact. The projects build something new but their novelty cannot be traced back to one specific feature in the practice of modeling. Rather, it seems to emerge through the combination of many different features. In this way, the technical aspects can come to the fore: modifications that are motivated by purely application-oriented considerations can tacitly infiltrate the core of the model. The emerging question is: with the complex interactions resulting from different modeling practices, can the distinction between the technical and the scientific aspects be made at all? Moreover, even if the answer to this question is yes, can this distinction be made solely with respect to the "original" theory? Viewed the other way around: how can we identify and track a genuinely *new* episteme that emerges from the code and gives rise to potentially new scientific concepts?

In (Lenhard 2019) and (Winsberg 2010), we find two different approaches to handling scientific code. While both Johannes Lenhard and Eric Winsberg focus on dynamic numerical simulations, the former is more mathematical, whereas the latter is more epistemologically oriented. Lenhard argues "that computer and simulation modeling really do form a new type of mathematical modeling" (Lenhard 2019, p. 2). He argues that simulations can really be seen as a new type of scientific investigation, and that reflection on this type can reveal relevant philosophical aspects. He tries to show that mathematical modeling has evolved into an "autonomous process that mediates between theory and intended applications" (ibid., p. 4). Although it is a kind of mathematical modeling, simulation modeling is not so much about complexity reduction (as classical models are), given that is has to balance high complexity with usability (ibid., p. 9). This seems to lead to a new logic of scientific discovery. In this sense, Lenhard opts to see simulation modeling as the cradle of a new kind of scientific rationality, which is in line with the thought above, according to which a new kind of episteme could emerge through the articulation of scientific code. Winsberg states that as simulations "more often involve the application rather than the testing of scientific theories", the corresponding epistemology is not directed at the justification of a theory. What has to be reevaluated is the "relationship between theories and local descriptions of phenomena" (Winsberg 2010, p. 3). His questions center around the kind of epistemological contribution numerical simulations provide: e.g., whether they can be read according to the scheme of

experimental practice provided by the likes of Hacking 1983 and Franklin 2016, i.e., with respect to the quasi-autonomy of simulations that start as applications of theories but that later decouple (to a degree) from the theories. Both Lenhard and Winsberg concentrate on modeling practices and simulations. As we have described in section 1.1, modeling and simulation are only two incarnations of scientific code. The list of relevant programs includes, among others, systems that manage and generate data structures (including traditional digital archives), as well as classical AI programs that are used for software validation and make big software projects possible in the first place. A conceptual analysis that aims at uncovering a new episteme should therefore be able to tackle all of these different kinds of programs. To illustrate how such a conceptual analysis of scientific code can be conducted with the help of GICAT, we return to the example outlined in section 1.2. As a case study, we will look at the concept "emission line" by first outlining its entangled historical genesis and then locating it in its specific function in the code of xgaltool.

3.3 Getting at the concepts?

Although the term "gas" was not coined before the 17th century[13], the interaction between gas and light has been studied at least since Aristotle explained the color and shape of the rainbow and related phenomena trough the interaction of "light" (sight and the sun) and vapor (raindrops in the clouds just before they begin to fall) via his theory of optics.[14] Here the rainbow is not explained directly through the substances-accident scheme (as demanded by Aristotle's theory of scientific explanation, cf. 3.1), but rather indirectly through the relationship between substances. Accordingly, Aristotle classifies the rainbow and its properties as *mere* phenomena, i.e., as something that seems to be a substance only *prima facie* but in reality lacks ontological autonomy. Consequently, his take on vapor ("gas") is not fully commensurable with his ontological presuppositions. As an in-between state of the element of water, it shifts between being a proper substance and being the state of an object.[15] A state is not just an accident but a possible mode of a substance that is enabled by specific circumstances. This aspect can also be found in our modern characterization of gas: what is now called an elemental gas (like hydrogen or

[13] The term "gas" presumably was invented by Johan Baptista van Helmont in the 17th century to name the dust that was generated when water is frozen (carbon dioxide) as a phonetic variation of the ancient Greek word "chaos" (at least according to de Vries 1859).

[14] Meteorologica 3, 2-5 (371b-377a); (Aristotle 1951).

[15] Aristotle needs the element of water to be in two states at once: in the state of vapor to explain its position (in the sky) and in the state of solid drops for the qualities that figure in his optical explanation of the rainbow and its colors.

helium) are chemical elements that behave gas-like in standard conditions (concerning temperature and pressure) – whereby these standard conditions are purely conventional specifications based on the conditions on Earth at a certain altitude.

However, the modern concept of gas is not just a slight modification of Aristotle's approach. Instead, it was only possible through an ongoing dissolution of the ontology of substance, starting in the 17th century through developments mainly in mathematics and its application in mechanics. These developments gave rise to what has in retrospect been called the shift to an ontology of relations (Cassirer 1910, Kreis 2010, 75 pp). This new way of thought is summed up in the following phrase of the 19th century logician and mathematician George Boole: "The object of science, properly so called, is the knowledge of laws and relations" (Boole 1854, p. 39). In the case of "gas" this transformation is linked to the development of the concepts "density", "pressure" and "heat". The development line from Boyle's law (1662), Charles's law (1787) and Dalton's law (1801) up to the initiation of (qualitative) thermodynamics in Carnot's *Reflections*[16] from 1824 brought about a concept of gas that was less and less semantically grounded in a phenomenon. Instead, it was implicitly defined through a network of other concepts related to each other by mathematical equations (cf. Müller 2007). In roughly the same time frame (and heavily influenced by the above developments), the science of chemistry emerged with its own systematic take on what a gas is, defining the aforementioned elemental gases inscribed in the periodic table (1869). Both lines of development came together in the transformation of thermodynamics from a qualitative to a purely quantitative discipline (Boltzmann 1896) and the rise of physical chemistry as the study of chemical systems as physical systems. In both cases, the underlying math began to be dominated by statistics in the form of statistical mechanics.

In roughly the same period, also optics as a scientific discipline underwent radical changes. Aristotle's approach had long been partly succeeded and partly rivaled by his second century commentator Ptolemaios. Although Ptolemaios' optics were seen in many ways as a refinement of Aristotle's approach, both differed fundamentally concerning the question of emission and intromission: do we see because our eye emits rays that bounce of from objects (like a visual sonar)[17], or is our eye affected by something that comes from or represents the objects? Both lines of thoughts where combined and thereby superseded

[16] "Reflections on the Motive Power of Fire and on Machines Fitted to Develop that Power" (1824).

[17] The theory of emission – as strange as it seems at first glance – had the advantage that mathematics could be applied much easier. Euclids treatise on optics was accordingly based on the theory of emission.

with the work of Ibn al-Haytham (Alhazen) (written approximately 1000). The translation of his work into Latin (De aspectibus) in the 13th century had great influence on the thinkers of the European Renaissance (Lindberg 1967). It was Johannes Kepler who in 1604 (via a critical discussion of Ibn al-Haytham) not only revolutionized optics again, but also emancipated light as a scientific concept that denotes a proper entity that could be described mathematically in the context of a mechanics.[18] Building on Kepler's new mechanical/mathematical approach, we find in Descartes' *Discourse on Method* from 1637 (in Optics 2) the (first published) introduction of the law of refraction (Descartes 2001, 75 ff). Based on this and his take on gas-like phenomena in Meteorology 8 (ibid., 332 ff), we also find a new explanation of the rainbow. In 1704, Newton used the concept of refraction to explain the colors of light, famously by splitting white light into a spectrum of colors.[19] Based on his experiments, Newton was the first who claimed that the colors where actually a property of light. He favored[20] a corpuscular theory of light that stood against the wave theory of light proposed by his contemporaries like Christiaan Huygens. Huygens conjectured against Kepler that the speed of light is not infinite, so that traveling light could be imagined like a spreading wave (Huygens 1912, Zubairy 2016, p. 12). The question about the material nature of light seemed to be settled in 1801, when Thomas Young's double slit experiment established the wave theory of light and rendered the corpuscular theory obsolete (for the time being, cf. below). Now it was possible to identify a specific color with a specific wavelength (ibid., p. 12). Already in this early stage, it was conjectured that specific spectra co-vary with specific chemical substances – but it was not before Joseph von Frauenhofer replaced the prisms used since Newton with defraction gratings that dispersive spectrometers could be build (Brand 1995, 37 ff). Only after that development was it possible to explore the co-variation between spectra and substances in a systematic way, opening up the path to spectrography as a method of materials analysis. In 1865, James Clerk Maxwell published his results on electromagnetism, which made it possible to understand light as electromagnetic waves[21] – a theory that was experimentally solidified by Heinrich Herz in 1888. This had immense repercussions for

[18] All approaches before tied "light" to our visual capacities. According to the 38 propositions of the first book of Kepler's *Ad Vitellionem paralipomena* light is descried as material but not corporal as the used mathematical apparatus rendered it two-dimensional (Kepler 2000).

[19] Different from the *Principia*, Newton's optics is not mainly build by mathematical modeling and derivation but proceeds mainly by the meticulous description of experiments (Opticks I,2, Theorem 2, Newton 1952.)

[20] Newton only favored the corpuscular theory of light without thinking that he had really established it. He was undecided wether the corpuscular theory or the wave theory is true because this question could in his time not be decided by experiment - the main criteria according to the experimental methods developed in his Opticks.

[21] This rendered Young's theory that there had to be a medium for the waves of light obsolete.

our understanding of what light is. Having emerged from an attempt to explain human sight and specific phenomena like the rainbow, the theory of light now was emancipated from its original phenomenon by showing that what we can see is actually only a small part of the electromagnetic spectrum.[22] This also significantly extended the range of application of spectrography.

Chemical elements and compounds of chemical elements emit a spectrum of frequencies of electromagnetic radiation that is unique to them. According to today's theories, this radiation is caused by electrons making a transition from a high energy state to a lower energy state. The wavelength is therefore associated with a specific material ingrained on the subatomic level. In this way, an *emission line*, i.e., the systematic color grading that a spectroscope catches, can be used to reconstruct the material base of the source of the radiation. Concepts like the emission line (and absorption line) combine different theories and experimental and measuring practices. This combination allows us to see, think about and act upon things that we could not see, think about or act upon before. As John Brand puts it: "Spectra are the idiom of atoms and molecules. As we express ourselves in words and phrases, molecules announce their presence by series of frequencies in the electromagnetic spectrum" (Brand 1995). In section 1.2, we have seen that light (radiation) that is emitted by the gas content of galaxies is an essential source of modern deep space telescopy. But this information is not directly available; instead, it is buried in the massive collections of photometric data generated by the CCD-chips found in nearly every telescope.

From this background, let us now return to the GICAT analysis of the code of xgaltool. We can see that one of the nodes of figure 9 (analysis_tools) comprises a class called EmissionLineTools.

[22] Of course, the story goes on in the 20th century, starting with Max Planck's presentation of black-body radiation spectra in 1900 at the German Physical Society, as well as a paper that Albert Einstein published in 1905 where he was forced to reintroduce a new version of the corpuscle theory of light, in the sense that he had to postulate that light comes in discrete bursts of energy (quanta). The two discoveries lead to the development of what is today known as quantum mechanics (Zubairy 2016, 15 ff).

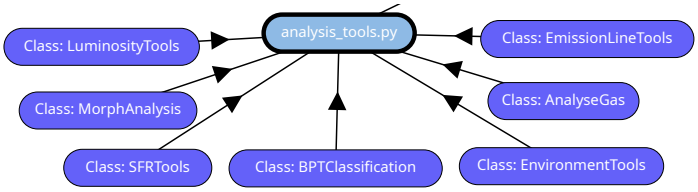

Figure 9: xgaltool class structure 2

This seems to be a promising starting point from where to look when we want to learn how the concept emission line is implemented in the code. Knowing that, we can dive deeper into the structure of the code and see how the classes that interest us are related to each other by activating the filter for class extensions.

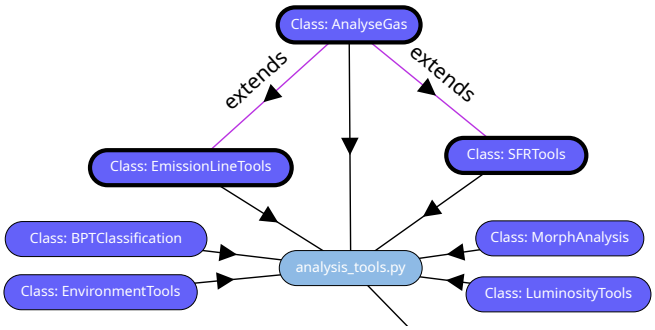

Figure 10: xgaltool 23.02.2022 class extension

This generates figure 10. Here we see that the class EmissionLineTools is part of a structure that also comprises the classes AnalyseGas and SFRTool. The name of latter class, in particular, gives us a hint, as it uses the standard abbreviation for star formation rate that (as we know) plays a central part in the scientific project that the code was written for. In figure 10, we can see that it is indeed a case of class extension, which means that what the program is executing works via the inheritance of properties. To keep it simple: in Python, properties and methods of objects define what data should be accessed by the program and how. So if there is an inheritance of the class AnalyseGas to the classes EmissionLineTools and SFRTools, we can assume that both tools are essential to examine how double peak emissions help to locate evolving galaxy formations.

This is already potentially interesting information, but with GICAT we can go further. A double click on EmissionLineTools shows us the code and we can see different definitions that are used to calculate the gas metallicity based on a specific emission line. If we now

deactivate the class extension filter and activate the import filter, the result is Figure 11.

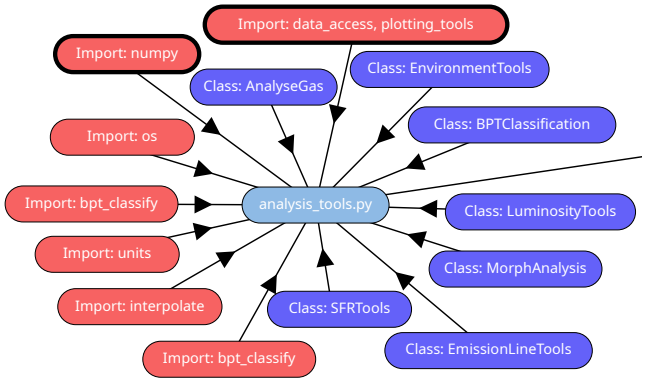

Figure 11: xgaltool import

Now we see two interesting imports: numpy is a Python library for mathematical calculations, mainly based on arrays and matrices. This gives us a hint about the mathematics used in xgaltool. The other interesting import is data access; plotting tools, as we can see that here data is drawn from another part of the program.

In addition to analysing the structure of the code, with GICAT we can also look at earlier versions of xgaltool to compare different stages of its development. Figure 12 shows us the same part of the program in an earlier stage of development.

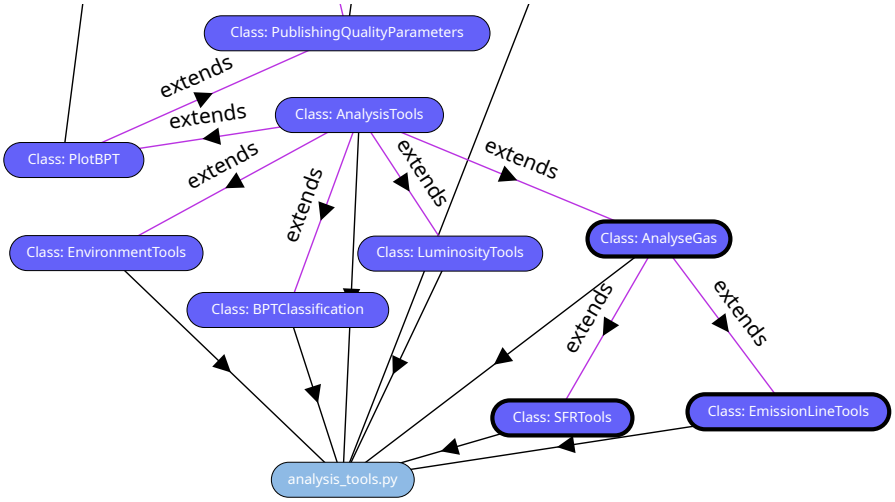

Figure 12: xgaltool 15.06.2021 class extension

We can see that the structure of the class extensions is more complex, and presumably messier, as every class is marked as an extension from the class AnalysisTools. What is interesting here is that there is a class extension which links this part of the program to another one. This link cannot be found in the newer version of xgaltool. The class PlottBPT that the central class AnalysisTools extends to is part of a section of xgaltool that computes how BPT plots are drawn. If we double click and look into the respective code we learn that in these specific plots lines that differentiate active galactic nuclei from other plot entries are drawn.

4 Conclusion

GICAT can be of great help with mundane problems like code-readability and navigation within the code. It is a powerful tool that enables the user to zoom in from the macro level of software architecture trough different layers and levels of interacting programs and subroutines down to individual lines of code. Its flexibility and modifiability allows for individual and experimental exploration. It is not dependent on any programming language or paradigm and can be used on any kind of code, independent of its age. This universality designates it as an important tool for science and technology studies, as well as for philosophy of science.

The features described can also be used to tackle a trivial but pressing problem: often the code produced by scientific researchers (although an essential part of their research is based upon it) is written just to work in the particular situation, without any consideration on its traceability or documentation for a third party.[23] In many instances, the result is something that is only traceable by the research team itself, or perhaps only one individuals within the team. Facing such situations, GICAT can be used in a straightforward manner to "reconstruct" the "meaning" of the code.

Finding and reconstructing the expected scientifically relevant structures of a given code and thereby identifying the scientific concepts that where consciously implemented in it is in itself no minor feat. However, the user of GICAT could could potentially do even more. Since it allows her to start directly in the code without any preconceptions, and due to its ability to visualize any kind of structure that can be captured by a regular expression, it enables her to discover structures "hidden" in the code – structures that where maybe not consciously implemented in it but that emerge through the interaction of its different

[23] So called "spaghetti-code", cf. (Gramelsberger, Wenz, and Kasprowicz 2024.)

layers. In other words: GICAT can be used to discover, explore and chart new epistemic structures and concepts that emerge in our digital scientific landscape.

References

Aristotle (1951). *Meteorlogica*. Trans. by Henry D. P. Lee. Harvard University Press.

Bloch-Mullins, Corinne L. (2020). "Scientific Concepts as Forward–Looking: How Taxonomic Structure Facilitates Conceptual Development". In: *Journal of the Philosophy of History* 14.2, pp. 205–231.

Bohm, David (1952). "A Suggested Interpretation of the Quantum Theory in Terms of Hidden Variables. II". In: *Physical Review* 85.2, pp. 180–193.

Bohr, Niels (1958). *Atomic Physics and Human Knowledge*. New York, Wiley.

Boltzmann, Ludwig (1896). *Vorlseungen über Gastheorie*. Ed. by J. A. Barth. Vol. 1. Leipzig.

Boole, George (1854). *An Investigation of the Laws of Thought on which are founded the Mathematical Theories of Logic and Probabilities.* London: Walton and Maberly.

Brand, John C. (1995). *Lines of Light: The Sources of Dispersive Spectroscopy, 1800-1930.* London: Routledge.

Brandom, Robert B. (2010). "Platforms, Patchworks, and Parking Garages: Wilsons Account of Conceptual Fine–Structure in Wandering Significance". In: *Philosophy and Phenomenological Research* 82.1, pp. 183–201.

Carnap, Rudolf (1937). *The Logical Syntax of Language (orig. Logische Syntax der Sprache 1934).* Trans. by Amethe Smeaton. Routledge.

– (1950). "Empiricism, Semantics and Ontology". In: *Revue Internationale de Philosophie* 4, pp. 20–40.

Cassirer, Ernst (1910). *Substanzbegriff und Funktionsbegriff. Untersuchungen über die Grundfragen der Erkenntniskritik*. Berlin.

Chilingarian, Igor V. et al. (Feb. 2017). "RCSED – A Value-added Reference Catalog of Spectral Energy Distributions of 800,299 Galaxies in 11 Ultraviolet, Optical, and Near-infrared Bands: Morphologies, Colors, Ionized Gas, and Stellar Population Properties". In: *The Astrophysical Journal Supplement Series* 228.2, p. 14.

Daston, Lorraine and Peter Galison (1992). "The Image of Objectivity". In: *Representations* 40, pp. 81–128.

De Vries, Matthias (1859). "Woordafleidingen". In: *De Taalgids* 1, pp. 247–282.

Descartes, Rene (2001). *Discourse on Method, Optics, Geometry, and Meteorology (1637).* Trans. by Paul J. Olscamp. Revised Edition. Hackett Publishing: Indianapolis.

Dowling, Deborah (1999). "Experimenting on Theories". In: *Science in Context* 12.2, pp. 261–273.

Duhem, Pierre (1954). *The Aim and Structure of Physical Theory (orig. La théorie physique son objet et sa structure 1914)*. Trans. by Philip P. Wiener. Princeton University Press.

Falkenburg, Brigitte (2007). *Particle Metaphysics. A Critical Account of Subatomic Reality*. Springer.

Franklin, Allan (2016). *What Makes a Good Experiment?: Reasons and Roles in Science*. University of Pittsburgh Press.

Galilei, Galileo (1960). "The Assayer: Translated from the Italian by Stillman Drake". In: *The Controversy on the Comets of 1618*. University of Pennsylvania Press, pp. 151–336.

Galison, Peter (2011). "Computer Simulations and the Trading Zone". In: *From Science to Computational Science*. Ed. by Gabriele Gramelsberger. Zürich: Diaphanes, pp. 118–157.

Gramelsberger, Gabriele (2010). *Computerexperimente. Zum Wandel der Wissenschaft im Zeitalter des Computers*. Transcript.

– ed. (2011). *From Science to Computational Sciences. Studies in the History of Computing and its Influence on Todays Science and Society*. Diaphanes.

Gramelsberger, Gabriele, Daniel Wenz, and Dawid Kasprowicz (2024). "Understanding and Analysing Sciences Algorithmic Regimes: A Primer in Computational Science Code Studies". In: *Algorithmic Regimes: Methods, Interactions, and Politics*. Ed. by Simon Egbert Yana Boeva Hendrik Heuer Juliane Jarke Bianca Prietl and Maike Arnold. Amsterdam University Press, pp. 57–78.

Grant, Edward (1996). *The Foundations of Modern Science in the Middle Ages. Their Religious, Institutional and Intellectual Contexts*. Cambridge University Press.

Hacking, Ian (1983). *Representing and Intervening: Introductory Topics in the Philosophy of Natural Science*. Cambridge University Press.

Han, Jie, Wei Qiu, and Eric Lichtfouse (2024). *ChatGPT in Scientific Research and Writing: A Beginners Guide*. Springer Nature Switzerland.

Heisenberg, Werner (1927). "Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik". In: *Zeitschrift für Physik* 43.3–4, pp. 172–198.

Hempel, Carl G. (1965). "Aspects of Scientific Explanation". In: *Aspects of Scientific Explanation and other Essays in the Philosophy of Science*. Free Press, pp. 331–496.

Hoeppe, Götz (2014). "Working data together: The accountability and reflexivity of digital astronomical practice". In: *Social Studies of Science* 44.2, pp. 243–270.

Humphreys, Paul (Aug. 2004). *Extending Ourselves: Computational Science, Empiricism, and Scientific Method*. Oxford University PressNew York.

Huygens, Christiaan (1912). *Treatise on Light: In Which Are Explained the Causes of That Which Occurs in Reflection and Refraction (1690)*. Trans. by Silvanus P. Thompson. London: Macmillan.

Kant, Immanuel (1998). *Critique of Pure Reason*. Trans. by Paul Guyer and Allen W. Wood. Cambridge University Press.

Kepler, Johannes (2000). *Optics: Paralipomena to Witelo & Optical Part of Astronomy (Ad Vitellionem paralipomena. Astronomiae pars optica 1604)*. New Mexico: Green Lion Press.

Kleene, Stephen C. (1951). *Representation of Events in Nerve Nets and Finite Automata*. Tech. rep. RM-704. The RAND Corporation. URL: `https://www.rand.org/content/dam/rand/pubs/research_memoranda/2008/RM704.pdf`.

Kleene, Stephen. C. (1956). "Representation of Events in Nerve Nets and Finite Automata". In: *Automata Studies*. Ed. by C. E. Shannon and J. McCarthy. Princeton: Princeton University Press, pp. 3–42.

Kreis, Guido (2010). *Cassirer und die Formen des Geistes*. suhrkamp.

Lazinger, Susan S. (2017). "Issues of Policy and Practice in Digital Preservation". In: *Digital Libraries*. Routledge, pp. 99–112.

Lenhard, Johannes (2019). *Calculated Surprises: A Philosophy of Computer Simulation*. Oxford University Press.

Lindberg, David (1967). "Alhazen's Theory of Vision and its Reception in the West". In: *Isis* 58.3, pp. 321–341.

Löbl, Rudolf (1997–2002). *TEXNH - TECHNE. Untersuchung zur Bedeutung dieses Wortes in der Zeit von Homer bis Aristoteles*. Vol. I + II. Königshausen und Neumann.

Maschmann, Daniel et al. (Sept. 2020). "Double-peak emission line galaxies in the SDSS catalogue: A minor merger sequence". In: *Astronomy & Astrophysics* 641, A171.

McCulloch, Warren S. and Walter Pitts (Dec. 1943). "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133.

Merz, Martina (2006). "Locating the Dry Lab on the Lab Map". In: *Simulation: Pragmatic Construction of Reality*. Ed. by Johannes Lenhard, Günter Küppers, and Terry Shinn. Dordrecht: Springer Netherlands, pp. 155–172.

Morrison, Margaret and Mary S. Morgan (1999). "Models as Mediating Instruments". In: *Models as Mediators: Perspectives on Natural and Social Science*. Ed. by Mary S. Morgan and Margaret Morrison. Cambridge University Press.

Müller, Ingo (2007). *A History of Thermodynamics: The Doctrine of Energy and Entropy*. Springer.

Newton, Isaac (1952). *Opticks or A Treatise of the Reflections, Refractions, Inflections & Colors of Light (1730)*. 4th Edition. New York: Dover Publications.

Poincaré, Henri (2017). *Science and Hypothesis (orig. La Science et l'Hypothèse 1902)*. Trans. by Mélanie Frappier, Andrea Smith, and David J. Stump. Bloomsbury Publishing.

Popper, Karl (1959). *The Logic of Scientific Discovery (orig. Logik der Forschung 1935)*. Trans. by by the author. second edition. Hutchinson.

Quine, Willard van Orman (1960). *Word and Object*. MIT Press.

– (1969). "Ontological Relativity". In: *Ontological Relativity and other essays*. Columbia University Press, pp. 26–68.

Reichenbach, Hans (1944). *Philosophic Foundations of Quantum Mechanics*. University of California Press.

Suppes, Patrick (1960). "A comparison of the meaning and uses of models in mathematics and the empirical sciences". In: *Synthese* 12.2–3, pp. 287–301.

Van Fraassen, Bas C. (1980). *The Scientific Image*. Oxford University Press.

Wilson, Mark (2006). *Wandering Significance: An Essay on Conceptual Behaviour*. Oxford University Press.

– (2017). *Physics Avoidance. Essays in Conceptual Strategy*. Oxford University Press.

Winsberg, Eric (2010). *Science in the Age of Computer Simulation*. University of Chicago Press.

Winter, Thomas Nelson (1999). "Roberto Busa, S.J., and the Invention of the Machine-Generated Concordance". In: *The Classical Bulletin* 75 (1), pp. 3–20.

Zubairy, M. Suhail (2016). "A Very Brief History of Light". In: *Optics in Our Time*. Ed. by Mohammad D. Al-Amri, Mohamed El-Gomati, and M. Suhail Zubairy. Springer International Publishing, pp. 3–24.