Variability in Data Visualization: a Software Product Line Approach

Jose-Miguel Horcas
University of Seville
Seville, Spain
jhorcas@us.es

Jose A. Galindo
University of Seville
Seville, Spain
jagalindo@us.es

David Benavides
University of Seville
Seville, Spain
benavides@us.es

ABSTRACT

Data visualization aims to effectively communicate quantitative information by understanding which techniques and displays work better for different circumstances and why. There are a variety of software solutions capable of generating a multitude of different visualizations of the same dataset. However, data visualization exposes a large space of visual configurations depending on the type of data to be visualized, the different displays (e.g., scatter plots, line graphs, pie charts), the visual components to encode the data (e.g., lines, dots, bars), or the specific visual attributes of those components (e.g., color, shape, size, length). Researchers and developers are not usually aware about best practices in data visualization, and they are required to learn about both the design practices that make communication effective and the low level details of the specific software tool used to generate the visualization. This paper proposes a software product line approach to model and materialize the variability of the visualization design process, guided by feature models. We encode the visualization knowledge regarding the best design practices, resolve the variability following a step-wise configuration approach, and then evaluate our proposal for a specific software visualization tool. Our solution helps researchers and developers communicate their quantitative results effectively by assisting them in the selection and generation of the visualizations that work best for each case. We open a new window of research where data visualization and variability meet each other.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; •

Human-centered computing → Information visualization; *Graph drawings.*

KEYWORDS

effective communication,

feature model,

graph,

quantitative data,

software product line,

variability,

visualization

1 INTRODUCTION

In digital transformation scenarios in which we are now immersed, such as Cloud Computing, Internet of Things, or Cyber-Physical Systems, a large amount of data is produced, stored, and analyzed [63]. Quantitative information forms the core data of what organizations and practitioners (e.g., researchers, scientists) must know to exploit and communicate their results and advances. The quantity and diversity of the data make it difficult to extract and present relevant information for end-user consumption. *Data visualization* [31] is an interdisciplinary field that deals with visual representation of data to effectively communicate information.

However, data visualization exposes a large space of possible visual configurations depending on the type of data to be visualized, the data relationships, and the message we want to transmit [17, 32]. The use of graphs and tables is a common practice in organizations today, becoming the fundamental vehicle to represent quantitative information [19]. There exist considerable variations in graphs and tables (e.g., scatter plots, line graphs, bidirectional tables) that correspond to different quantitative relationships of the data (e.g., correlation, ratio, ranking, hierarchical), and each of these variations can be paired with the visual components and techniques (e.g., points, lines, bars, boxes) that present them most effectively. These visual components can be further configured to improve their visual perception through multiple visual attributes such as form attributes (e.g., length, width, orientation, shape, size, enclosure), color attributes (e.g., hue, intensity), or position attributes (e.g., 2D, 3D position), among others. Both graphs and tables have been developed over time to the point that visualization experts now thoroughly understand which works better for different circumstances and why [17, 61]. Nevertheless, few practitioners have yet learned the design practices that make the use of graphs and tables effective [12, 43]. Evidence of this fact in the form of countless poorly designed graphs and tables is visible in the literature (Section 2). For example, pie charts are widely used for visual representation, despite the fact that they are not recommended for data that represent a relationship between a part and a whole, because there exists evidence of cognitive perception that indicates that the human eye is not well prepared to detect differences in the size of the angles; and in this case, bar graphs are better options for easy reading and comparing data [19].

Moreover, there are a variety of software solutions capable of generating a multitude of different visualizations of the same data. Examples of languages and tools used in data science activities and industry digital transformation are *Data-Driven Documents (D3.js)* [7] for web engineering, *Grafana* [10] for cloud computing,

Pgfplots and *TikZ* [57] in LaTeX for research, *Ggplot2* [65] in R and *Matplotlib* [26] in Python for data analysis, just to mention a few. These options offer little support to guide users or developers in determining the most correct way to visualize the data, that is, how to decide which visualization is the most appropriate for a dataset with certain properties. The problem increases with the huge number of configurations regarding the visual components and attributes that those libraries offer and that may overwhelm the user, requiring considerable technical skill to know the low-level details of the software library used to generate the visualization.

In this paper, we propose a *software product line* (SPL) [51] approach to model and materialize the variability of the visualization design process (Section 3). The *visualization design process* [63] is concerned with the design practices, development, and application of visual representation of data assisted by computers to display the information for accurate and efficient interpretation by the reader. We make the following contributions:

- We synthesize the best practices of the visualization design process that have been learned through many years of research and real-world trial and error by trailblazers [17, 19]. To do so, we encode the variability of the visualization knowledge regarding the best design practices in feature models and follow a *step-wise configuration* [14] approach that allows configuring different displays, visual components, and attributes in a specific software visualization library or tool, so that non-experts in visualization can easily decide the best representation for their data (Section 4).
- We materialize the variability of visualization designs by automatically generating effective visualization implementations using existing software solutions. To do so, we rely on *template-based code generation* [56] to resolve the variability of the visualization at different levels of abstraction, so that users and developers can build effective graphs without a deep understanding of the low-level details of each visualization tool (Section 5).
- We provide an implementation of our SPL approach and apply it to four practical scenarios to generate visualizations that take into account the best design practices to communicate a quantitative message (Section 6).

Although there are several works discussing the best visualizations for SPL artifacts [38] (e.g., visualization of feature models), to the best of our knowledge, there is no approach that manages the variability of the visualization design process (Section 7). Our solution helps practitioners communicate their quantitative data effectively by assisting them in the selection and generation of the visualizations that work best for each case. We envision that this contribution can open a new line of research where data visualization and variability management, analysis, and implementation can benefit of each other (Section 8).

2 UPBRINGING DATA VISUALIZATION

This section presents the main concepts of the visualization design process and the best design practices, motivating our approach.

2.1 The visualization design process

The visualization design process [31] is concerned with the design practices, development and application of visual representation of data to effectively communicate information. A formal model of the visualization design process is proposed by Walny et al. [63] and includes five main stages (Figure 1): (1) conceptualization of the vision and goals of the project, as well as the dataset; (2) analysis and characterization of the data to be visualized including understanding the data types, amounts, and relationships; (3) abstraction and encoding design steps that encompass creative design and data mapping with the required representation, resulting in a visualization design concept (or visualization design documentation); (4) implementation of the visualization according to the design performed in the previous phases and using a specific software visualization library; and (5) deployment of the visualization for public use, including its maintenance and data updates. In this paper, we focus on stages 3 (Visualization Design) and 4 (Visualization Development) highlighted in Figure 1 which are the most interesting phases due to their complexity in variability.

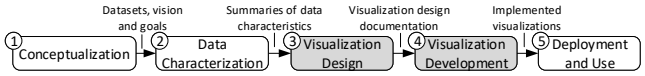


Figure 1: Stages of the data visualization design process [63].

Informally, data visualization maps values to visuals, turning numbers into graphs to convey a story or an idea as efficiently as possible. According to [19], tables and graphs are the main visualizations (aka displays) to structure and communicate quantitative information. Due to the visual nature of graphs, graphs expose a high degree of variability, requiring a number of unique design practices in contrast to tables. Therefore, in this paper, we focus mainly on the visualization of graphs.

Visualization design of graphs. Graphs display relationships (e.g., time series, ranking, part-to-whole, distribution) in quantitative information by giving shape to those relationships. Multiple structural forms of graphs can be used to display each relationship by encoding quantitative values as points (e.g., dot plots, scatter plots, strip plots, bubbles), lines (e.g., line graphs that may also include points, frequency polygon graphs), bars (e.g., histograms, table lens), boxes (e.g., box plots with vertical or horizontal orientation), and 2D and 3D shapes (e.g., pie charts, donut charts, radar charts). Deciding the best value-encoding object and graph type for each kind of relationship requires to understand the specific types of relationships that graphs can display [19]. For instance, data with a correlation relationship should be displayed by encoding values with points as a scatter plot [19], but the user needs to be able to identify that the message she wants to transmit is a correlation in the data (e.g., a causal relationship) in contrast to a deviation or a distribution relationship. Furthermore, several visual and textual components work together in graphs to present quantitative information. Design involves not only choosing the primary components that are used to construct the graphs (i.e., points, lines, bars, plots), but also designing the secondary components and non-data components to display the information (e.g., axes, trend lines, reference lines, annotations, scales, tick marks, grid lines, legends, etc.), as well as determining the visual attributes of each component such as its form (e.g., length, width, orientation, size, shape, enclosure), its color (e.g., hue, intensity), or its position (e.g., 2D position).

Visualization development of graphs. There are a variety of software solutions to develop and generate a visualization of data such as *Matplotlib* [26], *Ggplot2* [65], *Pgfplots* [57], *Grafana* [10]. Each tool requires considerable technical skill by the users to know

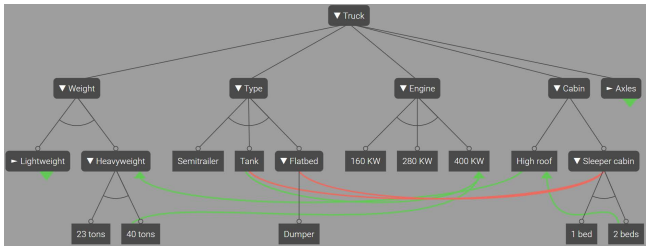
its low-level details in order to generate an efficient visualization. Moreover, these tools expose a colossal number of options regarding the visual components and attributes to develop the visualization, but they offer little support to guide users or developers regarding the most correct way to visualize the data.

2.2 Best design practices in data visualization

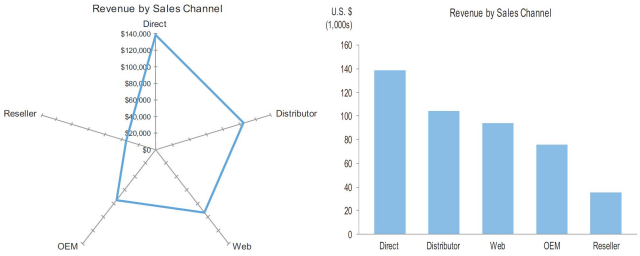
Despite the high number of possible options in the design and implementation of visualization to represent data, only a few of the configurations are appropriate to effectively communicate information. For example, only graphs based on points, lines, bars, and boxes are recommended (with rare exceptions) [19] because they rely on visual attributes that can be easily and accurately perceived; while 2D and 3D shapes (e.g., pie charts, donut charts) fail at data representation, mainly due to perceptual reasons, even though their high popularity [19]. In this regard, many fields of scientific study have contributed to the understanding of visual perception and have applied it to visual design [6, 9, 12, 19, 44, 47, 61]. A set of patterns known as the *Gestalt Principles of Visual Perception* [23] uncovered how people perceive patterns, forms, and organizations, revealing that people tend to group objects in particular ways. Many of these findings have been applied in the design of tables and graphs [9, 19, 47]. For instance, the *principle of proximity* [23] establishes that objects that are close to each other are perceived as belonging to a group, and thus, tables can be arranged better in a particular direction without the need of using horizontal or vertical rules to delineate rows or columns, improving the *data-ink ratio* [19] (i.e., the amount of ink that presents important data compared to the total amount of ink used to present the display). Another example is the *principle of similarity* [23] that establishes that similar objects in color, size, shape, or orientation tend to be grouped together; and thus, shapes with symbols keeping their interiors empty of color (circles, squares, and triangles) are easier to distinguish in graphs. Moreover, some visual attributes considerably affect the visual perception of the display [12], such as the color, where there are a list of nine hues (i.e., gray, blue, orange, green, pink, brown, purple, yellow, red) that meet the requirement of distinctness, and therefore they are easy to recognize and distinct enough to work well together; but its selection is challenging because red and green in combination can be difficult to differentiate for colorblindness people.

Deciding whether to use a table or a graph as a display, the best components to build such a display, and the visual attributes or those components, requires knowing the principles of visual perception behavior and its application to graphical communication [43, 58], but also understanding the specific types of relationships that each visualization can display [19]. However, yet few of the practitioners have learned the design practices that make the use of tables and graphs effective. Evidence of this fact in the form of countless poorly designed tables and graphs is visible in the literature. Concretely, in Figure 2 and Figure 10 (in Section 6), we illustrate some of the design problems in visual representations extracted from the literature. A summary of the best design practices for visualization to decide the best display according to the data relationships and communicate the information effectively is shown in Tables 3 and 4 in Appendix A.

[1]The content information inside these graphs and tables are not relevant in our paper.



(a) Feature model displayed in the *Glencoe online tool* [2]. The dark hues used do not meet the requirement of distinctness with features' colors fading more into the background. Also, green and red colors for "requires" and "excludes" constraints are difficult to distinguish for colorblindness people. In contrast, black features over a white background and different line styles for constraints would be easier to read.



(b) The radar chart at the left (taken from [19]) is a graph type often available in visualization tools, but it does not present information clearly, accurately, and efficiently, mostly because of perceptual reasons. The same nominal comparison of sales channels can be better represented with the bar graph at the right-hand side (from [19]). The bar graph is easier to read since positions along a quantitative scale are much easier to compare when they are laid out linearly along a single vertical or horizontal axis. Also, a bar graph supports additional meanings to be displayed such as ranking the items, but the radar char does not because it is not clear where the information begins and ends, or whether it should be read clockwise or counterclokwise.

Type	Model	Z3	Clafer	BBPF
FSE2015	Dune	26.20s	11s	0.01s
	HSMGP	40.70s	14s	0.01s
	HiPAcc	458s	33s	0.01s
	Trimesh	Time-out	2s	0.01s
KConfig	axTLS	Time-out	Time-out	0.01s
	Fiasco	Time-out	Time-out	0.01s
	uClibc-ng	Time-out	Time-out	0.01s

Type	Model	Z3	Clafer	*BBPF*
FSE2015	Dune	26.20s	11s	0.01s
	HSMGP	40.70s	14s	0.01s
	HiPAcc	458s	33s	0.01s
	Trimesh	Time-out	2s	0.01s
KConfig	axTLS	Time-out	Time-out	0.01s
	Fiasco	Time-out	Time-out	0.01s
	uClibc-ng	Time-out	Time-out	0.01s

(c) Comparison of counting time solutions of three solvers [45]. The table at the left (from [45]) contains a high ratio of non-data ink hindering the important data. The same table (at the right) considers the *principle of proximity* [23] to reduce the unnecessary rules (non-data ink) and enhance the relevant information.

Figure 2: Examples of visualizations, extracted from the literature, that suffer from design problems[1].

In the next section, we propose an approach to model and manage the variability of the visualization design process.

3 AN SPL FOR DATA VISUALIZATION

Our approach consists of an SPL (Figure 3) to model and manage the variability and encoding the best practices of the visualization design process. We follow the classical SPL framework [51] that separates the domain and the application engineering processes, and distinguishes between the problem and the solution spaces.

In the domain engineering process, we analyze the knowledge and variability of the visualization design process encoding them in feature models (top left of Figure 3), and prepare the reusable and variable artifacts to be used in multiple visualizations in terms

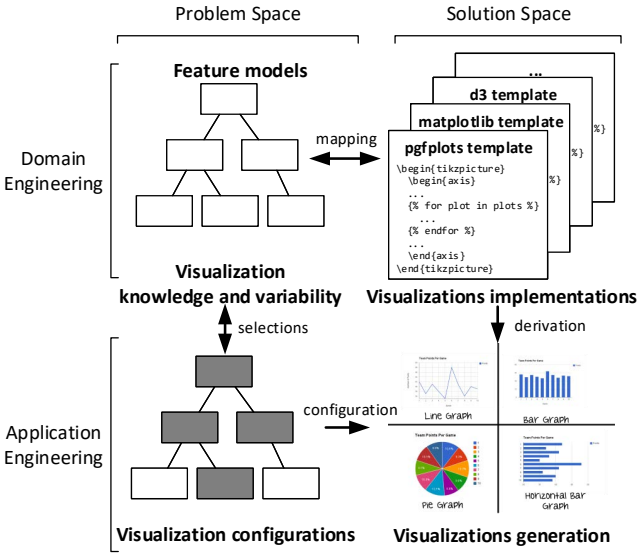


Figure 3: SPL for the visualization design process.

of templates for the different visualization software libraries (top right of Figure 3). In the application engineering process, we specify configurations for the visualization according to the stakeholder's requirements (bottom left of Figure 3) and generate a final display for a specific software library (bottom right of Figure 3).

Since we are interested in the separation of the visualization knowledge and variability from the implementation in a specific software library, we will distinguish the perspectives of the problem and the solution space in the following sections. The problem space takes the perspective of stakeholders (e.g., researchers, data analysts) and the problematic of designing visualizations that effectively communicate quantitative information about their data (Section 4). In contrast, the solution space represents the developer's perspective, including the generation of the visualization in a particular visualization software library (Section 5).

4 VARIABILITY MODELING AND CONFIGURATION IN VISUALIZATION

To address the problem space of our SPL, we first analyze and extract the knowledge and best practices of the visualization design process that have been learned through many years of research and real-world trial and error by trailblazers. To do so, we analyze the literature on visualization design techniques and best practices [9, 17, 19, 32, 47, 58], and synthesize a set of feature models that encode this knowledge. Then we configure the feature models following a step-wise configuration technique [14].

Figure 4 illustrates our approach to model and configure the variability of the visualization design process. Following the visualization design and development phases presented in Section 2, we refine the modeling of variability [52, 53] into three fundamental steps in the design of a visualization identified by Few [19] and an additional step for the development of the visualization: (1) determine the message to display; (2) select the best means to display the information; (3) design the display to show the information; and (4) select the implementation for the visualization. Each step has a particular goal in the design and development of the visualization, and thus, we encode the knowledge and variability of each step in

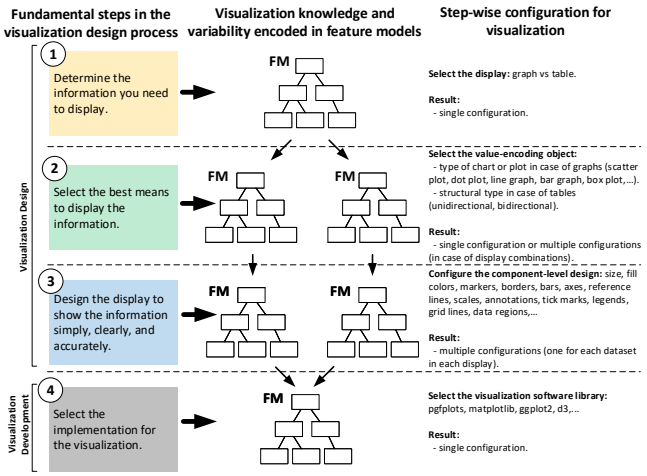


Figure 4: Our approach to model and configure the variability of the visualization design process.

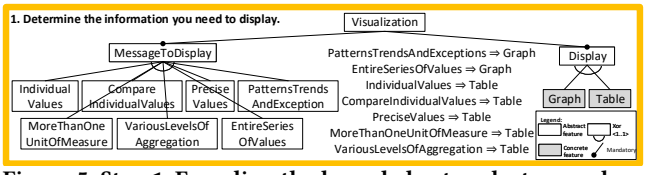


Figure 5: Step 1. Encoding the knowledge to select a graph or a table to communicate the information.

separate feature models. Then, feature models are configured step by step [14].

Step 1. Determine the information you need to display (Figure 5). The first step is to determine the message to be displayed from the data, and consequently choose the best medium of communication (i.e., a table or a graph). This requires knowing the different roles of graphs and tables when presenting quantitative information. As stated by Few [19]: *"tables make it easy to look up individual values"*, while *"graphs are used to display relationships among and between sets of quantitative values by giving them shape"*. Complete knowledge of whether a graph or a table is more appropriate is available in Appendix A and has been encoded in the feature model of Figure 5. This feature model allows deciding whether to use a table or a graph according to the message you want to transmit. The feature model encodes the different types of message as abstract features, and the relations between the message and the best concrete medium of communication to show that message (a table or a graph) as cross-tree constraints. To configure this feature model, the user only needs to select the message she wants to display, resulting in the best medium to display the message: a graph or a table. As a result, a single configuration is obtained that will guide the next step in the visualization design process.

Step 2. Select the best means to display the information (Figure 6). In the second step of the visualization design process, the goal is to select the best means to display the information in terms of the specific chart type (in the case of graphs) or structural type (in the case of tables) according to the relationships of the data to communicate. As in step 1, the available knowledge extracted from the literature is summarized in Appendix A and has been encoded in the feature model of Figure 6 for the graphs. Figure 6

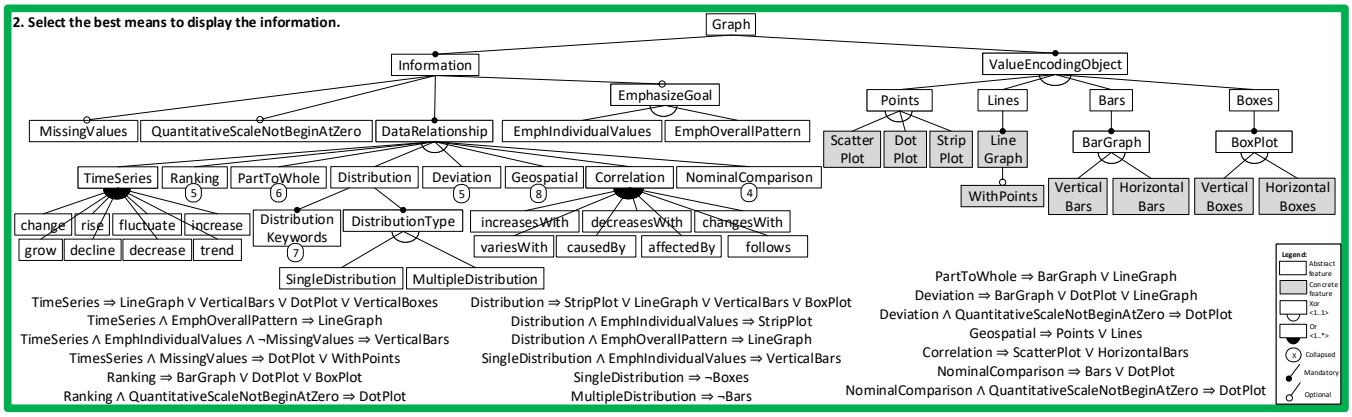


Figure 6: Step 2. Feature model to determine the best graph type according to the data relationship.

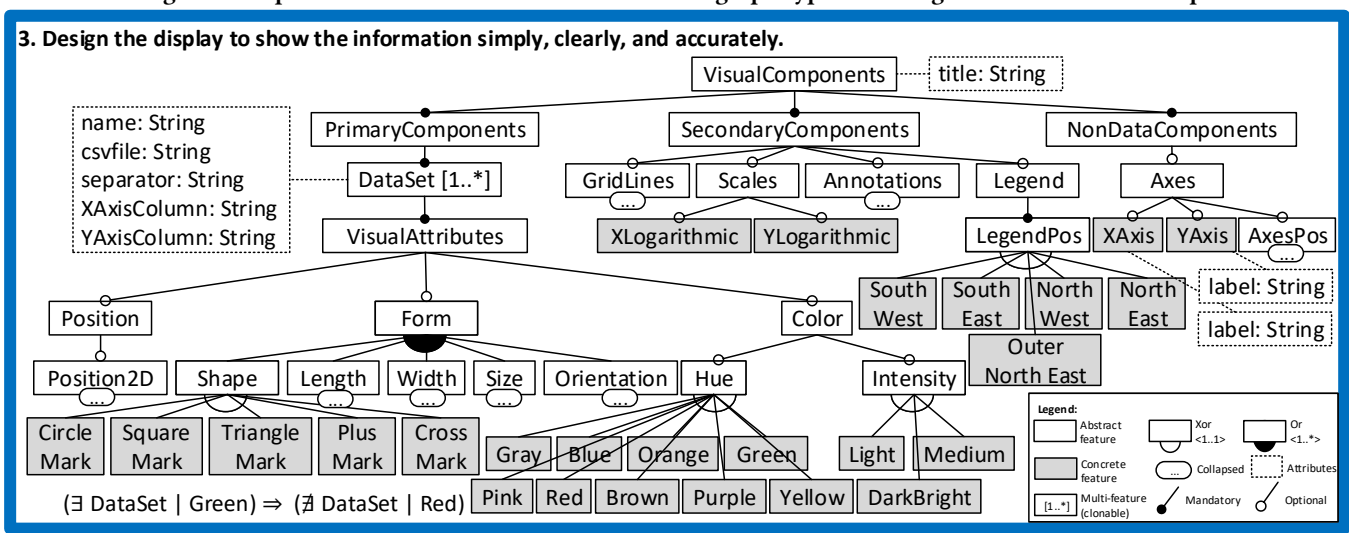


Figure 7: Step 3. Variability of the visual component-level graph design.

specifies the fundamental variations of graphs in terms of chart types (concrete features in gray) that correspond to different quantitative relationships (abstract features in white). Knowledge about the type of graph that presents each quantitative relationship more effectively is encoded in the cross-tree constraints. For example, to represent a nominal comparison in the data, it is preferable to use bars by means of either a vertical or a horizontal bar graph; but when then quantitative values of the scale do not begin at zero, using dot plots is recommended [19]. To help the user determine whether her quantitative message involves a particular data relationship or another, we also encode a set of keywords and phrases that are commonly used to identify each relationship [19]. So, the user can first state the message in writing and then look to see whether she used any of the keywords to transmit the message. For example, the message likely involves a time series if it includes any of the following words: *change*, *rise*, *increase*, *fluctuate*, *grow*, *decline*, *decrease*, *trend*; while the following words suggest a correlation relationship: *increases with*, *decreases with*, *changes with*, *varies with*, *caused by*, *affected by*, *follows*. The feature model of this second step is normally configured once for a visualization, but can be configured multiple times to represent a combination of displays (e.g., a bar graphs combined with a time series). Here, we have illustrated how to model the variation on graphs, while the variations on tables shown in Appendix A are modeled similarly.

Step 3. Design the display to show the information simply, clearly, and accurately (Figure 7). In this step, we model the variability of the visual component-level graph design. The goal is to make the visual objects that encode data prominent, accurate, and clear to communicate the information. Since tables are slightly more forgiving of visual design flaws because tables encode data through the use of verbal language (i.e., text) visually displayed [19], in this step we focus on the design of graphs. Graphs are constructed from components that can be divided into three groups: (i) primary components, that is, the points, bars, lines, and boxes that encode the quantitative data; (ii) components that serve secondary roles like the scales, trend lines, tick marks, and so on; and (iii) non-data components like the axis lines. Figure 7 shows an excerpt from the feature model that encodes the variability in the design of the visual components. On the left-hand side of Figure 7, the visual attributes of the primary components need to be configured for each set of data, and thus we model the `DataSet[1..*]` feature as a *multi-feature* [1, 13] (aka *clonable feature*) that allows one to configure the entire subtree as many times as necessary. For instance, consider a graph representing a comparison of algorithms; each

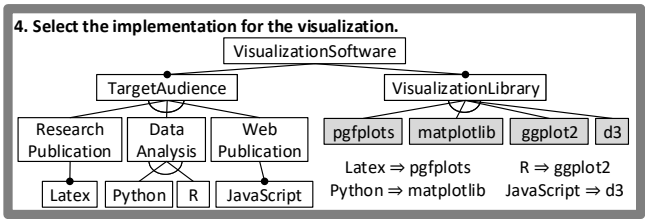


Figure 8: Step 4. Development alternatives for visualization.

series of data (or dataset) will represent an algorithm with different visual attributes (position, form, color) to easily distinguish them. Note that the variability of the modeled visual attributes includes only those values that are recommended by visualization experts in the literature [17, 19, 58], in contrast to modeling all possibilities of a specific software tool. This allows us to reduce the configuration space and avoid the selection of features exposed by the tool that may result in poor design practices undermining the final visualization. For example, we model only those shapes that are easy to distinguish from one another like circle (◯), square (□), triangle (△), plus (+), and times (×) marks; or those color hues that meet the requirement of distinctness [19]. Some best practices can be encoded as complex cross-tree constraints such as the fact that green and red colors should not be used in combination because they can be difficult to differentiate for colorblindness people; therefore, we explicitly avoid the selection of both colors in different dataset of the same visualization: $(\exists\ DataSet \mid Green) \Rightarrow (\nexists\ DataSet \mid Red)$. In addition, each dataset requires to provide a set of attributes such as a name to identify the data, the file path to the (.csv) file containing the data, and the column name for the X and Y axes inside the data file. This is modeled as *feature attributes* [5] associated with the `DataSet` feature. The secondary components (middle of Figure 7) include several other components of graphs that also provide information and can be configured such as the scales (linear or logarithmic), grid lines, annotations, or the legend including its position, among others [19]. We collapsed some of those features (...) and omitted others, such as trend lines or reference lines, due to lack of space. Finally, non-data components (right-hand side of Figure 7) are axis lines that give graphs dimension, serve as a container for the data and also provide various possibilities for configurations, such as the number of axes and their positions. For instance, under most circumstances, graphs include a single axis either vertical or horizontal for 1D graphs, or two perpendicular axes for 2D graphs. The configuration process of the feature model in this third step results in multiple configurations, one for each dataset due to the presence of the `DataSet[1..*]` multi-feature, as well as due to the possibility of building a combination of displays from step 2 (e.g., a dot plot with a trend line).

Step 4. Select the implementation for the visualization (Figure 8). The last step consists of choosing the software library or tool to produce the visualization designed in the previous steps. The goal here is to decide the visualization tool depending on the target audience or the publicity method for the visualization (e.g., research publication, data analysis, web publication), without the need to know about the different software solutions available. Figure 8 shows an example of a feature model that exposes four well-known visualization libraries: `pgfplots` [57] for research publications in LaTeX, `matplotlib` [26] in Python and `ggplot2` [65] in R for data analysis, and `D3` [7] in JavaScript for publication on the web. The configuration of this feature model will guide the development of the visualization, as explained in the following section.

Listing 1: Excerpt of the *Jinja2* template for graphs visualizations in LaTeX using the *pgfplots* and *tikz* packages.

```
\begin{tikzpicture}
  \begin{axis}[
    title={{ title }},
    xlabel={{ xaxis_label }}, ylabel={{ yaxis_label }},
    ...
    {{ plot_type }},
    {% if x_log_scale %} xmode=log, {% endif %}
    {% if y_log_scale %} ymode=log, {% endif %}
    {% if legend_pos %} legend pos={{ legend_pos }}, {% endif %}
  ]

  {% for plot in plots %}
  \addplot+[
    mark={{ plot.mark }},
    color={{ plot.color }},
    ...
  ]
  table[
    x={{ plot.xaxis_column }},
    y={{ plot.yaxis_column }},
    col sep={{ plot.separator }},
  ]
  { {{ plot.csv_file }} };
  {% if legend_pos %}\addlegendentry{ {{ plot.name }} }{% endif %}
  {% endfor %}
  \end{axis}
\end{tikzpicture}
```

5 VARIABILITY IMPLEMENTATION AND RESOLUTION IN VISUALIZATION

To address the solution space of our SPL, we first implement the variable visualization artifacts and then resolve the variability of those artifacts according to the configurations provided, generating the final visualization in a particular software library.

In the domain engineering of our SPL (top-right of Figure 3), we implement and resolve the variability using a *template-based code generation* [56] approach. In a template-based approach, a text-based language (e.g., JavaScript, Python, LaTeX) is enriched with directives to automatically generate custom content. A template system allows us to reuse static elements (e.g., web pages) while defining dynamic elements based on parameters. In visualization, templates support static non-variable content, providing basic structure and appearance, while the dynamic variable content (the visual components and attributes) are specified as parameters. Listing 1 shows an example of a variable artifact for visualization using a template-based approach, concretely using the *Jinja2*[2] template engine. Variable elements are specified between curly braces and have been highlighted in bold. These variable elements represent variation points and will be replaced by the selected variants in the configurations. We choose a template-based approach in contrast to a pure annotative-based approach [33] for implementing the variability, because the visualization domain exposes a large set of variants for each variation point, and therefore, a pure annotative-based approach (e.g., C preprocessors with *#ifdef* directives) requires introducing a high number of annotations to represent each possible variant. In addition, a template-based approach supports more powerful directives that allow the use of composition-based

[2] *Jinja2*: https://palletsprojects.com/p/jinja/

	Variation point		Variants (V)	
	Feature (f)	**Handler (h)**	**Feature/Attribute (ρ)**	**Value (v)**

Features	ValueEncodingObject	`plot_type`	ScatterPlot
			LineGraph
			VerticalBars
			HorizontalBars
			…
				`only marks`
				`smooth`
				`ybar`
				`xbar`
				…

	XLogarithmic	`x_log_scale`	-	–
	YLogarithmic	`y_log_scale`	-	–

	LegendPos	`legend_pos`	SouthWest
			SouthEast
			NorthWest
			NorthEast
			OuterNorthEast
				`south west`
				`south east`
				`north west`
				`north east`
				`outer north east`

Attributes	VisualComponents	`title`	VisualComponents.title	`(provided)`
	XAxis	`xaxis_label`	XAxis.label	`(provided)`
	YAxis	`yaxis_label`	YAxis.label	`(provided)`

Features of Multi-features	Shape	`plot.mark`	CircleMark
			SquareMark
			TriangleMark
			PlusMark
			CrossMark
				`o`
				`square`
				`triangle`
				`+`
				`x`

	Color	`plot.color`	Gray
			Blue
			Orange
			…
				`gray`
				`blue`
				`orange`
				…

Multi-feature Attributes	DataSet	`plot.name`	DataSet.name	`(provided)`
	DataSet	`plot.csv_file`	DataSet.csvfile	`(provided)`
	DataSet	`plot.xaxis_column`	DataSet.XAxisColumn	`(provided)`
	…	…	…	…

Table 1: Mapping model of variation points and variants.

mechanisms [3] such as the substitution and/or replacement of elements, the assignment of values, or the use of control structures such as *if/elif/else*, *for-loops*, *macros*, and *blocks*. For example, the for-loop in Listing 1 (lines 12–25) iterates over the plots that will represent each dataset modeled in the feature model of Step 3 (Figure 7) as the `DataSet[1..*]` multi-feature. The template engine will generate a copy of the code between lines 12–25 for each instance of `DataSet[1..*]` provided in the configurations.

To map the variability specified in the feature models with the implementation of the visualizations, we define a mapping model that relates the features in the feature models with the variation points and variants in the parameterizable templates.

Mapping features to variation points and variants in visualization artifacts. Pohl et al. [51] define a variation point in the context of an SPL as *"a representation of a variability subject within domain artifacts enriched by contextual information"*, and a variant as *"a representation of a variability object within domain artifacts"*. In our case, the domain artifacts are the code templates in charge of generating the visualization (e.g., LaTeX or Python source code), and the contextual information are the directives (e.g., *Jinja2* directives, *#ifdef* directives) that specify the variability subjects (e.g., plot type, color, etc.). The associated variability objects will be every variant (e.g., scatter plot or bar chart for the plot type, or the different color variants). To simplify the formalization of a variation point, we rely on the definition of Jacobson et al. [29] who define a variation point as *"one or more locations at which the variation will occur"*, and we formally define a variation point and its variants as follows:

Definition 5.1 (Variation point, variant). A *variation point* is defined as a 3-tuple (f, h, V):

- f is the feature in the feature model associated with the variation point.
- h is the handler (or identifier) of the location in the artifact where the variation takes place.
- V is the set of possible variants for this variation point. If no variants are specified for the variation point (i.e., $V = \emptyset$), the handler h is processed as a boolean flag (i.e., like a *#ifdef* directive).

A *variant* $v \in V$ is defined as a tuple (ρ, v):

- ρ is the feature or attribute in the feature model associated with the variant.
- v is the concrete value to be substituted/replaced/assigned to the handler h of the variation point.

With these definitions, we define a mapping to establish a relation between the feature models specified in Section 4 and the template-based variable artifacts. Table 1 shows the mapping model of variation points and variants for the *Jinja2* template of Listing 1. For instance, when the feature (e.g., `ValueEncodingObject`) associated with a variation point is selected in a configuration of the feature models, the variant chosen in the configuration (e.g., `ScatterPlot`) will be considered to replace its associated value (e.g., `only marks`) in the handler of the template (e.g., `plot_type`). When the feature is not selected in a configuration, the variation point identified by the handler is removed from the template. The same behavior is applied for those variation points without variants (e.g., `XLogarithmic`). The *Jinja2* template engine receives a map structure of $(handler, value)$ pairs for the variation points whose associated features have been selected in the configurations of the feature models. In case of a multi-feature, the *value* is a list of maps with $(handler, value)$ pairs that can be traversed with the for-loop directive in the template (lines 12–25 in Listing 1).

6 VALIDATION

This section illustrates the feasibility and applicability of our proposal by providing an implementation of our SPL for visualization and applies it to four practical scenarios to generate a visualization that takes into account the best design practices to communicate a quantitative message.

To develop our SPL, we rely on the following technologies: (i) the Universal Variability Language (UVL) [55] to model the variability of the visualization design process; (ii) FeatureIDE [41] to create the configurations of the feature models; (iii) the Python framework for automated analysis of feature models proposed in [21] to manage and deal with the feature models and configurations; and (iv) the *Jinja2* template engine for Python to implement and resolve the variability. All artifacts of our SPL, including the feature models in *UVL*, the configurations, the *Jinja2* templates for the *pgfplots* library in LaTeX, the Python scripts to manage and orchestrate the feature models and configurations, as well as the resources and visualizations to replicate these experiments are available online[3]. In the following, we apply our approach to four scenarios with different quantitative data and messages to be visualized.

Scenario 1. Let us consider the quantitative data (Table 2) that contain the number of articles published in the proceedings of the SPLC, VaMoS, and ConfWS conferences available in the *dblp*

[3] https://doi.org/10.5281/zenodo.6474762

Table 2: Quantitative data for the visualization Scenario 1: Number of papers in proceedings for the SPLC, VaMoS, and ConfWS conferences (extracted from *dblp* [59]).

Year	SPLC	VaMoS	ConfWS	Year	SPLC	VaMoS	ConfWS	Year	SPLC	VaMoS	ConfWS
2000	27	NA	NA	2008	64	17	NA	2016	51	14	NA
2001	NA	NA	NA	2009	48	23	NA	2017	37	15	NA
2002	24	NA	NA	2010	67	30	NA	2018	49	17	19
2003	NA	NA	NA	2011	58	21	7	2019	54	17	13
2004	42	NA	NA	2012	41	22	9	2020	41	25	NA
2005	24	NA	NA	2013	49	21	16	2021	37	19	13
2006	43	NA	NA	2014	52	23	13				
2007	27	21	NA	2015	65	16	21				

NA: Not available in *dblp* [59]. SPLC: https://dblp.org/db/conf/splc, VaMoS: https://dblp.org/db/conf/vamos, ConfWS: https://dblp.org/db/conf/confws.

computer science bibliography [59]. Following the four configuration steps of our approach (see Section 4), we can generate a visualization for these quantitative data:

(1) **Determine the information you need to display.** We are interested in displaying the entire series of values to show how the number of articles have changed through time or identify trends. Therefore, we create a configuration of the feature model of Figure 5 by just selecting the feature `EntireSeriesOfValues`, which results in the following configuration: Config_FM1 = {`Visualization`, `MessageToDisplay`, `EntireSeriesOfValues`, `Display`, **`Graph`**}. As expected the best communication vehicle for our data is a graph. Indeed, from Table 2 is difficult to take a overall pattern of the data. In the configuration, we have highlighted the concrete features automatically selected (**`Graph`**) and underlined the feature manually selected by the user (`EntireSeriesOfValues`). Others abstract features are also automatically selected due to the feature model relations.

(2) **Select the best means to display the information.** We want to display "how the number of articles changes through time", so we can easily identify that the data relationship we want to visualize is a time series. In fact, our message contains the "change" keyword that help us to identify the time series. To create a configuration of the feature model of Figure 6, we select the `change` feature which results in the selection of the `TimeSeries` feature. At this point, to display a time series, a line graph or vertical bars are preferable (see best practices in Appendix A). However, our data (Table 2) contain missing values at some years either because the conference was not celebrated or because the data is not available in the *dblp* [59]. Thus, we select the `MissingValues` feature, suggesting us that a dot plot or a line graph with points are the most preferable representation. As we are interested in the overall pattern, we also select the `EmphOverallPattern` feature, resulting in the configuration: Config_FM2 = {`Graph`, `Information`, `MissingValues`, `EmphasizeGoal`, `EmphOverallPattern`, `DataRelationship`, `TimeSeries`, `change`, `ValueEncodingObject`, `Lines`, **`LineGraph`**, **`WithPoints`**}; that gives us the line graph with points as the recommended encoding objects to display the information.

(3) **Design the display to show the information simply, clearly, and accurately.** In the third step, we configure the visual components for our display (feature model of Figure 7). We have three datasets that implies to instantiate the multi-feature `DataSet [1..*]` three times, one for each conference (SPLC, VaMoS, and ConfWS), configuring the visual attributes for each dataset differently: DataSet1_SPLC = {`VisualAttributes`, `Form`, `Shape`, `CircleMark`, `Color`, `Hue`, `Blue`}; DataSet2_VaMoS = {`VisualAttributes`, `Form`, `Shape`, `SquareMark`, `Color`, `Hue`, `Red`}; DataSet3_ConfWS = {`VisualAttributes`, `Form`, `Shape`, `TriangleMark`, `Color`, `Hue`, `Brown`}. We also set the `name`, `datafile`, `separator` `XAxisColumn`, and `YAxisColumn` attributes for each dataset accordingly; and select the secondary and non-data components such as the `NorthWest` position for the legend, and set the `title` and `label`(s) for the axes; resulting in the configuration: Config_FM3 = {`VisualComponents.title="Number of publications in proceedings"`, `PrimaryComponents`, `DataSet1_SPLC`, `DataSet2_VaMoS`, `DataSet3_ConfWS`, `SecondaryComponents`, `Legend`, `LegendPos`, `NorthWest`, `NonDataComponents`, `Axes`, `XAxis.title="Year"`, `YAxis.title="Papers"`}. Since the feature model only exposes the recommended variations extracted from the visualization literature, any configuration of the visual attributes will ensure the best design practices for visualization.

(4) **Select the implementation for the visualization.** In the last step, we decide to generate the visualization for a research publication in LaTeX. We create a configuration of the feature model of Figure 8: Config_FM4 = {`VisualizationSoftware`, `TargetAudience`, `ResearchPublication`, `Latex`, **`pgfplots`**}, which uses the *pgfplots* library, and thus, results in the use of the *pgfplots* template to generate the final visualization.

With the four configurations provided and the *pgfplots* template (Listing 1), we can resolve the variability as explained in Section 5 to generate the visualization of Figure 9: a time-series graph using a combination of lines and points to effectively encode the quantitative information of Table 2. The lines give the sense of continuity that is required for displays of time, while the points explicitly identify the exact position along the line whether the values are available because our data contain several missing values. We may remove the points and use only the lines in the case of collecting all intermediate values (e.g., years 2001 and 2003 for SPLC, and years 2016, 2017, and 2020 for ConfWS). We may also show individual values as points without connecting them with a line only if the time-series values were not collected at regular intervals of time (e.g., every year).

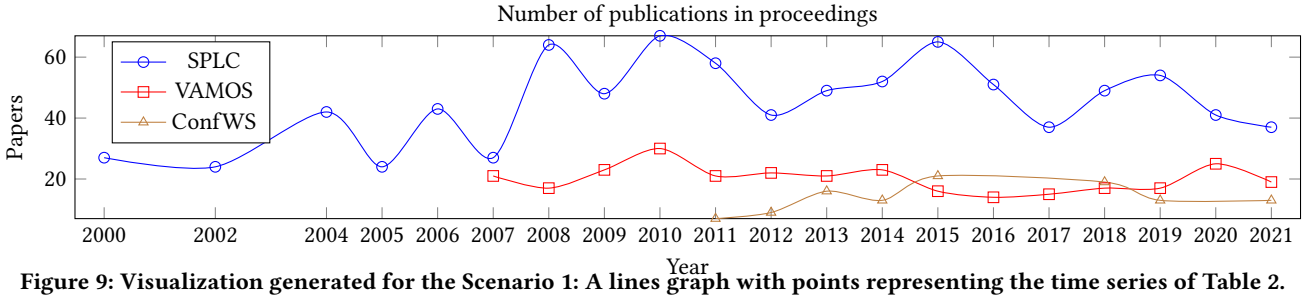
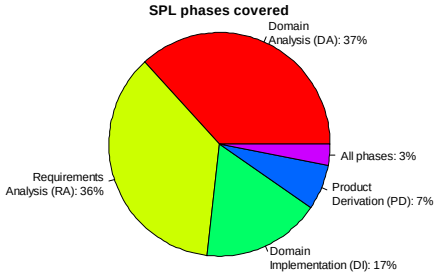
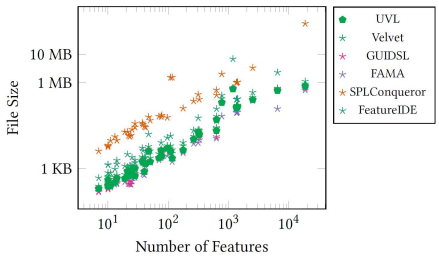


Figure 9: Visualization generated for the Scenario 1: A lines graph with points representing the time series of Table 2.





(a) Scenario 2: A pie chart (taken from [25]) is not recommended to represent a relationship between a part and a whole (the tool support for the SPL processes in this case) because differences in the size of the angles are difficult to be perceived, requiring labels to display the values.

(b) Scenario 3: A scatter plot (taken from [55]) comparing the file size of different formats for feature models. It suffers from a *over-lapping* problem [19] and in addition it uses shapes with the same form that are difficult to distinguish from each other.

(c) Scenario 4: A map (taken from *JHU CSSE COVID-19 Data* [16]) showing the total incidence of Covid-19 cases in Spain. It uses the size of the points as visual attribute to encode the quantitative data which is not appropriate because the values cannot be easily compared.

Figure 10: Visualizations scenarios, extracted from the literature, that suffer from design problems.

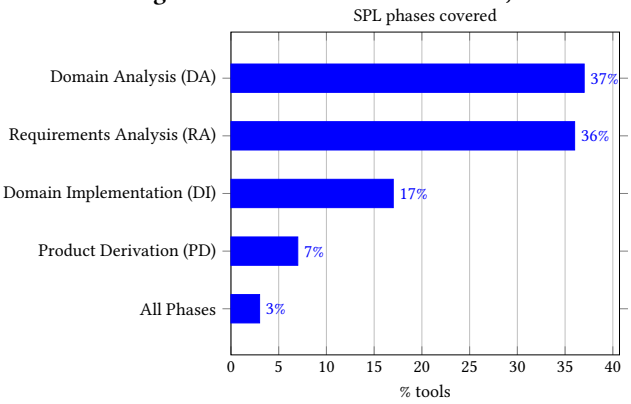


Figure 11: Visualization for Scenario 2: a bar chart for the same part-to-whole information as the pie chart of Fig. 10a.

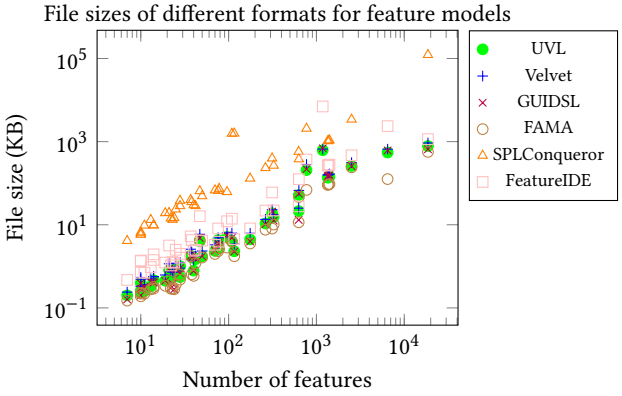


Figure 12: Visualization for Scenario 3: scatter plot of Fig. 10b with a different design improving its visual attributes.

In the following three scenarios, we use our approach to improve three visualizations described in Figure 10 by fixing their flaws and applying the best design practices encoded in our SPL.

Scenario 2. The pie chart in Figure 10a represents a part-to-whole relationship about the tool support for the different SPL processes [25]. We use the quantitative data available in [25], and create the configurations that result in the visualization of Figure 11. The recommended display for a part-to-whole relationship is a bar graph; or a line graph, in case of showing how parts of a whole

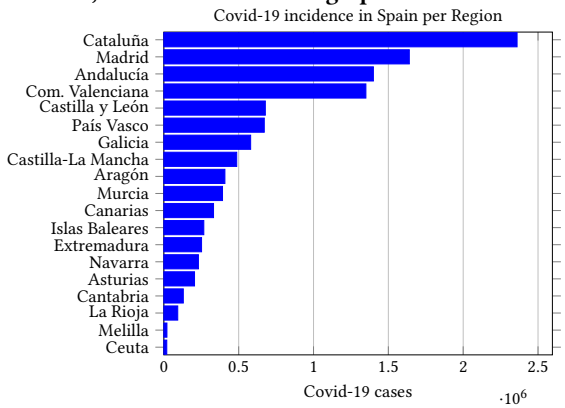


Figure 13: Visualization for Scenario 4: a bar chart to compare the Covid-19 incidence in Spain from map of Figure 10c.

have changed over time, which is not the case. The reason to prefer encoding a part-to-whole relationship as bars in contrast to a pie chart is that people can see the differences easily in the bar graph but with difficulty in the pie chart because the visual perception is highly tuned for seeing differences among the lengths of objects that share a common baseline but not well tuned for discerning difference among 2D areas [19]. For example, we cannot easily compare how much bigger is the "Domain Analysis (DA)" part from the "Requirements Analysis (RA)" part in the pie chart (Figure 10a) without looking at the numbers on the textual labels. In the bar graph (Figure 11), we have manually chosen horizontal bars in contrast to vertical bars because the categorical labels are too long to fit side by side in vertical bars. Vertical grid lines, as secondary components in the background, also facilitate quantitative comparison between parts. Finally, colors do not contribute to this visualization and may distract (the original pie chart even uses the red and green colors together); thus, we opt to use the same color for all datasets.

Scenario 3. The visualization in Figure 10b illustrates a correlation relationship between the file size and different formats of feature models [55]. In fact, a scatter plot is the preferred display to show a correlation relationship; however, the visual attributes of the original scatter plot present some flaws such as an *over-plotting* problem [19] (i.e., points are hidden or overlapped by other points to the degree that they cannot be distinguished), and the use of different scale units on the Y axis. We use the quantitative data

available in [55] to configure and recreate the display, which results in the visualization of Figure 12. First, we configure different shapes that are easiest to distinguish from each other, in contrast to using the same mark for five of six datasets as in the original display. Second, we enlarge the graph, decrease the size of the points, and keep the interior of the points empty of fill color to remedy the overplotting problem. Finally, we use a logarithmic scale to maintain the same unit on the vertical axis.

Scenario 4. In this final scenario, we apply our approach outside the SPL domain. Consider the visualization of Figure 10c showing the total incidence of Covid-19 cases per region in Spain [16]. The map encodes the quantitative values as points of different sizes, which, despite being perceived as fancy and pretty, is not effective in transmitting the message about which region has more incidence than others or in making comparisons between regions [12]. A nominal comparison like this is better represented with a bar graph. The configurations of our feature models generate the visualization of Figure 13. Since we have many regions (19), horizontal bars are more appropriate to fit the regions' labels than vertical bars. In addition, we maintain the same color for all bars as appears on the map and rank the regions by quantities to facilitate comparisons. We can observe on the bar graph (Figure 13) that the incidence in Cataluña (2,36 million) is much higher than in Madrid (1,64 million), but this difference is not appreciated on the map of Figure 10c.

7 RELATED WORK

The visualization and SPL research have crossed paths on several occasions, but from a different perspective than the one presented in this article. Studies focus on applying visualization techniques in the context of SPLs [49] mainly for the visualization of feature models [48], but also to visualize variability and configurations [4, 50], feature interactions [27], and constraints [39]. Techniques include the use of *colors* to visualize variabilities in source code [18, 30], polymetric views [36] in the form of a *variability blueprint* for the decomposition of complex feature models [62], *feature relations graphs* (Frogs) for feature constraints [39], *cone trees* to draw feature models in 3D [60], or the use of statistics to visualize large-scale feature models [24], among others. From this perspective, Lopez-Herrejon et al. [38] present a systematic mapping study of visualization techniques that have been used for different SPL activites. In contrast, to the best of our knowledge, the variability of the visualization domain has not been studied yet [22].

Several formal processes for visualization design have been proposed [8, 11, 46, 63], such as the one presented in Section 2 by Walny et al. [63]. Chi [11] also describes a framework to make information visualization systems easier to develop through the creation of a reference *Data State Model*. Munzner [46] provided a four-level nested model for visualization design and validation that was further extended in [40] and [42]. Brehmer et al. [8] discussed pre-design empirical methods for information visualization using four illustrative scenarios, highlighting the methods and challenges unique to the visualization domain. All of these works expose the need to design the visualization at different levels as we have done in our approach in Figure 4 (Section 4), but none of them takes into account the high variability that exists in the visualization design process. To synthesize variability and best design practices, numerous theoretical books have been published [9, 15, 17, 19, 23, 32, 47, 58, 64], of which we have mainly relied on Few [19] because it is the most practical, presenting the variations, principles and best practices of visual perception, as well as its application to graphical communication.

Finally, regarding the evaluation of a visualization, there is an ongoing discussion in the visualization community [34] about the relevant factors that make a visualization effective, expressive, memorable, aesthetically pleasing, etc. These factors are typically led by guidelines and model theories [37] of how different data representations and interaction concepts are perceived and processed by human users, requiring qualitative evaluation methods based on observation and interviews. For instance, in [34], Kurzhals and Weiskopf adopt the *repertory grid* methodology in the context of visualization. The repertory grid is a technique based on the personal construct theory for identifying the ways that a person interprets or gives meaning to, in this case, a visualization. The evaluation of design criteria for visualization can be performed using automatic algorithms (e.g., computation of *visual clutter* [54] and *scagnostics* [66]), or with structured approaches, such as *questionnaires* [20], that involve the human user for a quantitative and qualitative assessment of the visualization. Isenberg et al. [28] reviewed evaluation methods applied in visualization research, based on a coding scheme and seven scenarios by Lam et al. [35]. Our SPL can help with the quantitative assessment of the visualization by identifying those features that should be present in a configuration to fullfill the quality factors of a visualization (e.g., to be effective).

8 CONCLUSIONS AND FUTURE WORK

We have presented an SPL approach to deal with the high variability that exists in the visualization design process. Our solution helps practitioners communicate their quantitative data effectively by assisting them in the selection and generation of visualizations that best represent the desired information. The proposed SPL allows for the configuration of different displays, visual components, and visual attributes taking into account the principles and best design practices of visualization experts.

With this contribution, we open a new window of research where data visualization and variability meet each other, and where several challenges remain open. It is easy to imagine that more variability can be found in other parts of the data visualization process, and therefore finding different options to handle this variability is challenging. Data visualization is an area of research that is increasingly active with the current *data era* we are facing. Data that are related to computers but also data that are all over different areas ranging from biology to archaeology. We envision that variability management knowledge can be used in this trending area to facilitate the data visualization process, as we have partially shown in this contribution. In this regard, as part of our ongoing work, we plan to extend the modeling of other principles and best practices of the visualization knowledge that cannot be directly encoded in feature models, such as, for example, the fact that horizontal bars work better than vertical bars when the categorical labels are too long, as shown in Section 6, or concepts such as the data-ink ratio presented in Section 2. We also plan to include an explicit process for the quantitative assessment of visualizations in our SPL, considering the factors (e.g., effectiveness) that are typically led by guidelines.

OPEN SCIENCE

All the source code and data can be downloaded and executed from the following repository: https://doi.org/10.5281/zenodo.6474762

ACKNOWLEDGMENTS

This work was supported by the project *OPHELIA* (RTI2018-101204-B-C22) funded by FEDER/Ministry of Science and Innovation — State Research Agency; the *COPERNICA* (P20_01224) and *METAMORFOSIS* (FEDER_US-1381375) projects funded by Junta de Andalucía; and Juan de la Cierva—Formación 2019 grant.

REFERENCES

[1] Mauricio Alférez, Mathieu Acher, José Angel Galindo, Benoit Baudry, and David Benavides. 2019. Modeling variability in the video domain: language and experience report. *Softw. Qual. J.* 27, 1 (2019), 307–347. https://doi.org/10.1007/s11219-017-9400-8

[2] Georg Rock Anna Schmitt, Christian Bettinger. 2018. Glencoe – A Tool for Specification, Visualization and Formal Analysis of Product Lines. In *25th International Conference on Transdisciplinary Engineering (Advances in Transdisciplinary Engineering)*. 665–673. https://doi.org/10.3233/978-1-61499-898-3-665

[3] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer. https://doi.org/10.1007/978-3-642-37521-7

[4] Mohsen Asadi, Samaneh Soltani, Dragan Gasevic, and Marek Hatala. 2016. The effects of visualization and interaction techniques on feature model configuration. *Emp. Soft. Eng.* 21, 4 (2016), 1706–1743. https://doi.org/10.1007/s10664-014-9353-5

[5] David Benavides, Pablo Trinidad Martín-Arroyo, and Antonio Ruiz Cortés. 2005. Automated Reasoning on Feature Models. In *17th International Conference on Advanced Information Systems Engineering (CAiSE) (LNCS, Vol. 3520)*. Springer, 491–503. https://doi.org/10.1007/11431855_34

[6] Jacques Bertin. 1983. *Semiology of graphics*. University of Wisconsin press.

[7] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 Data-Driven Documents. *IEEE Trans. Vis. Comput. Graph.* 17, 12 (2011), 2301–2309. https://doi.org/10.1109/TVCG.2011.185

[8] Matthew Brehmer, Sheelagh Carpendale, Bongshin Lee, and Melanie Tory. 2014. Pre-Design Empiricism for Information Visualization: Scenarios, Methods, and Challenges. In *5th BELIV*. 147–151. https://doi.org/10.1145/2669557.2669564

[9] Alberto Cairo. 2012. *The Functional Art: An introduction to information graphics and visualization*. New Riders.

[10] Mainak Chakraborty and Ajit Pratap Kundan. 2021. *Grafana*. Apress, 187–240. https://doi.org/10.1007/978-1-4842-6888-9_6

[11] Ed Huai-hsin Chi. 2002. *A framework for visualizing information*. Vol. 1. Springer Science & Business Media.

[12] William S Cleveland and Robert McGill. 1985. Graphical perception and graphical methods for analyzing scientific data. *Science* 229, 4716 (1985), 828–833.

[13] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. 2005. Formalizing cardinality-based feature models and their specialization. *Softw. Process. Improv. Pract.* 10, 1 (2005), 7–29. https://doi.org/10.1002/spip.213

[14] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. 2005. Staged configuration through specialization and multilevel configuration of feature models. *Sw. Proc. Imp. Prac.* 10, 2 (2005), 143–169. https://doi.org/10.1002/spip.225

[15] Stephan Diehl. 2007. *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media.

[16] Ensheng Dong, Hongru Du, and Lauren Gardner. 2020. An interactive web-based dashboard to track COVID-19 in real time. *The Lancet Infectious Diseases* 20, 5 (2020), 533–534. https://doi.org/10.1016/S1473-3099(20)30120-1

[17] Stephanie DH Evergreen. 2019. *Effective data visualization: The right chart for the right data*. SAGE publications.

[18] Janet Feigenspan, Christian Kästner, Sven Apel, Jörg Liebig, Michael Schulze, Raimund Dachselt, Maria Papendieck, Thomas Leich, and Gunter Saake. 2013. Do background colors improve program comprehension in the #ifdef hell? *Empir. Softw. Eng.* 18, 4 (2013), 699–745. https://doi.org/10.1007/s10664-012-9208-x

[19] Stephen Few. 2012. *Show Me the Numbers: Designing Tables and Graphs to Enlighten* (2nd ed.). Analytics Press, Oakland, CA, USA.

[20] Camilla Forsell and Matthew Cooper. 2012. Questionnaires for Evaluation in Information Visualization. In *4th Workshop: Beyond Time and Errors: Novel Evaluation Methods for Visualization (BELIV)*. https://doi.org/10.1145/2442576.2442592

[21] José A. Galindo and David Benavides. 2020. A Python framework for the automated analysis of feature models: A first step to integrate community efforts. In *24th ACM International Systems and Software Product Line Conference (SPLC)*, Vol. B. ACM, Montreal, Canada, 52–55. https://doi.org/10.1145/3382026.3425773

[22] Matthias Galster, Danny Weyns, Dan Tofan, Bartosz Michalik, and Paris Avgeriou. 2014. Variability in Software Systems—A Systematic Literature Review. *IEEE Trans. on Softw. Eng.* 40, 3 (2014), 282–306. https://doi.org/10.1109/TSE.2013.56

[23] Ian E Gordon. 2004. *Theories of visual perception*. Psychology press.

[24] Ruben Heradio, David Fernández-Amorós, Christoph Mayr-Dorn, and Alexander Egyed. 2019. Supporting the statistical analysis of variability models. In *41st Int. Conf. on Soft. Eng. (ICSE)*. 843–853. https://doi.org/10.1109/ICSE.2019.00091

[25] José Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2022. Empirical analysis of the tool support for software product lines. *Software and Systems Modeling* (2022). https://doi.org/10.1007/s10270-022-01011-2

[26] J. D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* 9, 03 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[27] Sheny Illescas, Roberto E. Lopez-Herrejon, and Alexander Egyed. 2016. Towards Visualization of Feature Interactions in Software Product Lines. In *4th IEEE Work. Conf. on Soft. Vis. (VISSOFT)*. 46–50. https://doi.org/10.1109/VISSOFT.2016.16

[28] Tobias Isenberg, Petra Isenberg, Jian Chen, Michael Sedlmair, and Torsten Möller. 2013. A Systematic Review on the Practice of Evaluating Visualization. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2818–2827. https://doi.org/10.1109/TVCG.2013.126

[29] Ivar Jacobson, Martin L. Griss, and Patrik Jonsson. 1997. *Software reuse - architecture, process and organization for business*. Addison-Wesley-Longman.

[30] Christian Kästner, Salvador Trujillo, and Sven Apel. 2008. Visualizing Software Product Line Variabilities in Source Code. In *12th International Conference on Software Product Lines (SPLC)*, Vol. 2 (Workshops). 303–312.

[31] Andy Kirk. 2012. *Data Visualization: a successful design process*. Packt publishing.

[32] Cole Nussbaumer Knaflic. 2015. *Storytelling with data: A data visualization guide for business professionals*. John Wiley & Sons.

[33] Jacob Krüger, Marcus Pinnecke, Andy Kenner, Christopher Kruczek, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2018. Composing annotations without regret? Practical experiences using FeatureC. *Softw. Pract. Exp.* 48, 3 (2018), 402–427. https://doi.org/10.1002/spe.2525

[34] Kuno Kurzhals and Daniel Weiskopf. 2018. Exploring the Visualization Design Space with Repertory Grids. *Computer Graphics Forum* 37, 3 (2018), 133–144. https://doi.org/10.1111/cgf.13407

[35] Heidi Lam, Enrico Bertini, Petra Isenberg, Catherine Plaisant, and Sheelagh Carpendale. 2012. Empirical Studies in Information Visualization: Seven Scenarios. *IEEE Transactions on Visualization and Computer Graphics* 18, 9 (2012), 1520–1536. https://doi.org/10.1109/TVCG.2011.279

[36] M. Lanza and S. Ducasse. 2003. Polymetric views - a lightweight visual approach to reverse engineering. *IEEE Transactions on Software Engineering* 29, 9 (2003), 782–795. https://doi.org/10.1109/TSE.2003.1232284

[37] Andrea Lau and Andrew Vande Moere. 2007. Towards a Model of Information Aesthetics in Information Visualization. In *11th International Conference Information Visualization (IV)*. 87–92. https://doi.org/10.1109/IV.2007.114

[38] Roberto Erick Lopez-Herrejon, Sheny Illescas, and Alexander Egyed. 2018. A systematic mapping study of information visualization for software product line engineering. *J. Softw. Evol. Process.* 30, 2 (2018). https://doi.org/10.1002/smr.1912

[39] Jabier Martinez, Tewfik Ziadi, Raul Mazo, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2014. Feature Relations Graphs: A Visualisation Paradigm for Feature Constraints in Software Product Lines. In *2nd IEEE Working Conference on Software Visualization (VISSOFT)*. 50–59. https://doi.org/10.1109/VISSOFT.2014.18

[40] Sean McKenna, Dominika Mazur, James Agutter, and Miriah Meyer. 2014. Design Activity Framework for Visualization Design. *IEEE Trans. on Vis. and Comp. Graph.* 20, 12 (2014), 2191–2200. https://doi.org/10.1109/TVCG.2014.2346331

[41] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer. https://doi.org/10.1007/978-3-319-61443-4

[42] Miriah Meyer, Michael Sedlmair, P Samuel Quinan, and Tamara Munzner. 2015. The nested blocks and guidelines model. *Information Visualization* 14, 3 (2015), 234–249. https://doi.org/10.1177/1473871613510429

[43] Stephen R. Midway. 2020. Principles of Effective Data Visualization. *Patterns* 1, 9 (2020), 100141. https://doi.org/10.1016/j.patter.2020.100141

[44] Daniel Moody. 2009. The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering* 35, 6 (2009), 756–779. https://doi.org/10.1109/TSE.2009.67

[45] Daniel-Jesus Munoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don S. Batory. 2019. Uniform random sampling product configurations of feature models that have numerical features. In *23rd International Systems and Software Product Line Conference (SPLC)*. ACM, 39:1–39:13. https://doi.org/10.1145/3336294.3336297

[46] Tamara Munzner. 2009. A Nested Model for Visualization Design and Validation. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (2009), 921–928. https://doi.org/10.1109/TVCG.2009.111

[47] Scott Murray. 2017. *Interactive data visualization for the web: an introduction to designing with D3*. " O'Reilly Media, Inc.".

[48] Daren Nestor, Luke O'Malley, Aaron J. Quigley, Ernst Sikora, and Steffen Thiel. 2007. Visualisation of Variability in Software Product Line Engineering. In *1st Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*. 71–78. http://www.vamos-workshop.net/proceedings/VaMoS_2007_Proceedings.pdf

[49] Daren Nestor, Steffen Thiel, Goetz Botterweck, Ciarán Cawley, and Patrick Healy. 2008. Applying visualisation techniques in software product lines. In *ACM Symp. on Software Visualization*. 175–184. https://doi.org/10.1145/1409720.1409748

[50] Andreas Pleuss and Goetz Botterweck. 2012. Visualization of variability and configuration options. *Int. J. Softw. Tools Technol. Transf.* 14, 5 (2012), 497–510. https://doi.org/10.1007/s10009-012-0252-z

[51] Klaus Pohl, Günter Böckle, and Frank van der Linden. 2005. *Software Product Line Engineering - Foundations, Principles, and Techniques.* Springer. https://doi.org/10.1007/3-540-28901-1

[52] Daniela Rabiser, Herbert Prähofer, Paul Grünbacher, Michael Petruzelka, Klaus Eder, Florian Angerer, Mario Kromoser, and Andreas Grimmer. 2018. Multi-purpose, multi-level feature modeling of large-scale industrial software systems. *Soft. Sys. Model.* 17, 3 (2018), 913–938. https://doi.org/10.1007/s10270-016-0564-7

[53] Mark-Oliver Reiser and Matthias Weber. 2007. Multi-level feature trees. *Requir. Eng.* 12, 2 (2007), 57–75. https://doi.org/10.1007/s00766-007-0046-0

[54] Ruth Rosenholtz, Yuanzhen Li, and Lisa Nakano. 2007. Measuring visual clutter. *Journal of Vision* 7, 2 (2007), 17–17. https://doi.org/10.1167/7.2.17

[55] Chico Sundermann, Kevin Feichtinger, Dominik Engelhardt, Rick Rabiser, and Thomas Thüm. 2021. Yet another textual variability language?: a community effort towards a unified language. In *25th International Systems and Software Product Line Conference (SPLC).* 136–147. https://doi.org/10.1145/3461001.3471145

[56] Eugene Syriani, Lechanceux Luhunu, and Houari A. Sahraoui. 2018. Systematic mapping study of template-based code generation. *Comput. Lang. Syst. Struct.* 52 (2018), 43–62. https://doi.org/10.1016/j.cl.2017.11.003

[57] Till Tantau. 2013. Graph Drawing in TikZ. *J. Graph Algorithms Appl.* 17, 4 (2013), 495–513. https://doi.org/10.7155/jgaa.00301

[58] Alexandru C Telea. 2014. *Data visualization: principles and practice.* CRC Press.

[59] The dblp team. 2022. dblp computer science bibliography. Monthly snapshot release of March 2022. https://dblp.org/xml/release/dblp-2022-03-01.xml.gz

[60] Pablo Trinidad, Antonio Ruiz Cortés, David Benavides, and Sergio Segura. 2008. Three-Dimensional Feature Diagrams Visualization. In *12th International Conference on Software Product Lines (SPLC)*, Vol. 2 (Workshops). 295–302.

[61] Edward R Tufte. 1985. The visual display of quantitative information. *The Journal for Healthcare Quality (JHQ)* 7, 3 (1985), 15.

[62] Simon Urli, Alexandre Bergel, Mireille Blay-Fornarino, Philippe Collet, and Sébastien Mosser. 2015. A visual support for decomposing complex feature models. In *3rd IEEE VISSOFT.* 76–85. https://doi.org/10.1109/VISSOFT.2015.7332417

[63] Jagoda Walny, Christian Frisson, Mieka West, Doris Kosminsky, Søren Knudsen, Sheelagh Carpendale, and Wesley Willett. 2020. Data Changes Everything: Challenges and Opportunities in Data Visualization Design Handoff. *IEEE Trans. Vis. Comp. Graph.* 26, 1 (2020), 12–22. https://doi.org/10.1109/TVCG.2019.2934538

[64] Matthew O Ward, Georges Grinstein, and Daniel Keim. 2010. *Interactive data visualization: foundations, techniques, and applications.* AK Peters/CRC Press.

[65] Hadley Wickham. 2009. *ggplot2 - Elegant Graphics for Data Analysis.* Springer. https://doi.org/10.1007/978-0-387-98141-3

[66] Leland Wilkinson, Anushka Anand, and Robert L. Grossman. 2005. Graph-Theoretic Scagnostics. In *IEEE Symposium on Information Visualization (InfoVis).* 157–164. https://doi.org/10.1109/INFVIS.2005.1532142

A PRINCIPLES AND BEST DESIGN PRACTICES FOR DATA VISUALIZATION

Tables 3 and 4 summarize the recommendations in visualization design to select the best display for graphs and tables according to the data relationships, extracted from Few [19].

Table 3: Principles and best practices for tables design.

Data relationships in tables	Unidirectional	Bidirectional
Quantitative to Categorical (Look-up)		
Single set of quantitative values and a single set of categorical items. • *BP:* Use unidirectional tables; bidirectional tables are not applicable because there is only one set of categorical items.	■	□
Single set of quantitative values and the intersection of multiple categories. • *BP:* Unidirectional tables is preferable because of convention; bidirectional tables save space.	■	■
Single set of quantitative values and the intersection of multiple hierarchical categories. • *BP:* Use unidirectional tables to clearly display hierarchy by placing the separate levels side by side in adjacent columns; use bidirectional tables if the separate levels of the hierarchy are not split between the columns and rows.	■	⊠
Quantitative to Quantitative (Comparison)		
A single set of quantitative values associated with multiple categorical items. • *BP:* Bidirectional works especially well because the quantitative values are arranged closely together for easy comparison.	■	■
Distinct sets of quantitative values associated with a single categorical item. • *BP:* Use unidirectional tables; bidirectional tables tend to get messy as multiple sets of quantitative values are added.	■	⊠

■: Most appropriate; ⊠: Appropriate under certain conditions; □: Avoid.

Table 4: Principles and best practices for graphs design.

Data relationships in graphs	Points	Lines	Bars	Boxes
Time series: Values display how something changed through time. • *Keywords:* change, rise, increase, fluctuate, grow, decline, decrease, trend. • *BP:* Use preferably lines and vertical bars to emphasize on overall patterns and individual values, respectively; use points as *dot plots* only when missing values at time intervals; use boxes only for distributions changing through time.	⊠	■	■	⊠
Ranking: Values are ordered by size (ascending or descending). • *Keywords:* larger than, smaller than, equal to, greater than, less than. • *BP:* Use preferably bars; use *dot plots* when bars cannot be used because quantitative scale does not begin at zero; use boxes only when ranking multiple distributions.	⊠	□	■	⊠
Part-to-whole: Values represent parts (proportions) of a whole. • *Keywords:* rate or rate of total, percent or percentage of total, share, accounts for X percent. • *BP:* Use preferably bars; use lines to display how parts of a whole have changed through time.	□	■	■	□
Deviation: The difference between two sets of values. • *Keywords:* plus or minus, variance, difference, relative to. • *BP:* Use bars, but always vertical when combined with time series; *dot plots* when quantitative scale does not begin at zero; lines when combined with time series.	⊠	⊠	■	□
Distribution: Counts of values per interval from lowest to highest. • *Keywords:* normal curve, concentration, frequency, distribution, range, normal distribution, bell curve. • *BP:* Use *strip plots* to emphasis individual features; use a *frequency polygon* to emphasis the overall shape; use bars as *histograms* to emphasis on individual intervals in single distributions; avoid bars for multiple distributions; use *box plots* to compare multiple distributions.	■	■	⊠	⊠
Correlation: Comparison of two paired sets of values to determine if there is a relationship between them. • *Keywords:* increases with, decreases with, changes with, varies with, caused by, affected by, follows. • *BP:* Use a *scatter plot*; use a *table lens* when the audience is not familiar with *scatter plots.*	■	□	⊠	□
Geospatial: Values are displayed on a map to show their location. • *Keywords:* geography, location, where, region, territory, country, state, city. • *BP:* Use bubbles of various sizes on a map; use lines to mark routes on a map.	■	■	□	□
Nominal comparison: A simple comparison of values for a set of unordered items. • *Keywords:* this is bigger than that, this is the biggest of all, this is almost twice as big as that, these two are far bigger than all the others. • *BP:* Use preferably bars; use *dot plots* when bars cannot be used because quantitative scale does not begin at zero.	⊠	□	■	□

■: Most appropriate; ⊠: Appropriate under certain conditions; □: Avoid.