OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG

FACULTY OF
COMPUTER SCIENCE

Final Presentation:

„A Dashboard for Evolving Variability in Configurable System Software“

Lukas Petermann, 13.09.2025

1 Introduction

- Software systems evolve constantly
  - Configurability is often expected or required
- Especially interesting: **System software**
  - Safety and security: System software is the connecting link between hardware and software
  - Flexibility: Virtually countless combinations of hardware and software → Variability
- System software is often developed in product lines (SPL)
  - Related products that share the same core but otherwise differ in functionality
  - Effective and systematic development, variability management

Fig. 1: #Configurations of the Linux kernel [1]

- This variability of SPLs can be modeled via **feature models**
  - Describe valid configurations of an SPL by modeling features and dependencies
- System software variability is often described in DSLs like **KConfig**
  - No direct mapping between KConfig and SPL feature model
  - But: features can be extracted and analyzed automatically

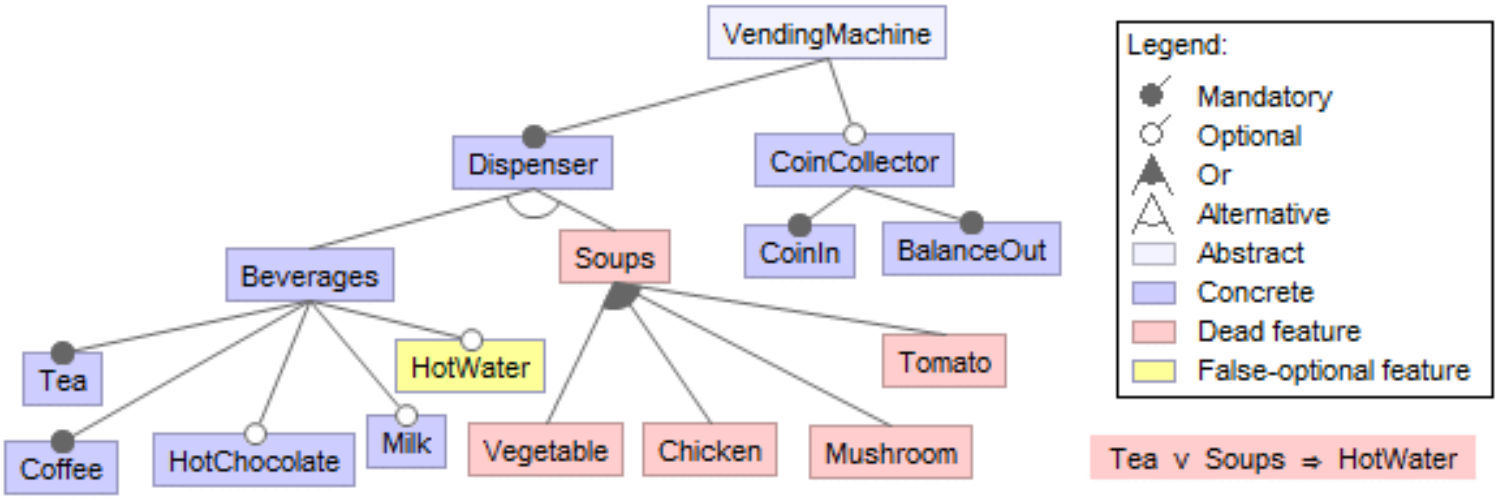


Fig. 2: Example feature model for a vending machine product line [2]

- This variability of SPLs can be modeled via **feature models**
  - Describe valid configurations of an SPL by modeling features and dependencies
- System software variability is often described in DSLs like **KConfig**
  - No direct mapping between KConfig and SPL feature model
  - But: features can be extracted and analyzed automatically

```
menu "Bluetooth device drivers"
    depends on BT
config BT_INTEL
    tristate
    select REGMAP

config BT_HCIBTUSB
    tristate "HCI USB driver"
    depends on USB
    select BT_INTEL
    help
      Bluetooth HCI USB driver.
      This driver is required if you want to use Bluetooth devices with
      USB interface.
      Say Y here to compile support for Bluetooth USB devices into the
      kernel or say M to compile it as module (btusb).
```

Fig. 3: Excerpt of the bluetooth driver KConfig

- Automated system software product line analysis
  - Feature model **analysis** → Analysis of configuration space, i.e., feature model semantics
  - Feature model **evolution** → Analysis of configuration history, i.e., feature model evolution over time
- Evolution is especially interesting:
  - Iterative development, open source → Development history is available
  - Usage in various settings → All revisions are interesting, not just the most recent
- Papers have been published with static tables and figures [1],[3],[4],[5]
- Tools like Torte[1] can automatically extract and analyze features

If only there was a way to better communicate these experiment results...

[1] https://github.com/ekuiter/torte

2 Torte Dashboard

- **Goal**: Visualization of current state & historical evolution
  - Choice of system software project and metric
  - Interactive plot illustrates growth over time and per revision
- **Vision:** Support for researchers:
  - Interactive plots add additional information (to static tables)
  - Reference dashboard from publication for more information
  - Room for more plots than in publication due to page limit
  - Other researchers can create their own dashboards
  - Easily extendable with new projects and metrics

Fig. 4: Torte dashboard concept

2.1 Projects & Metrics

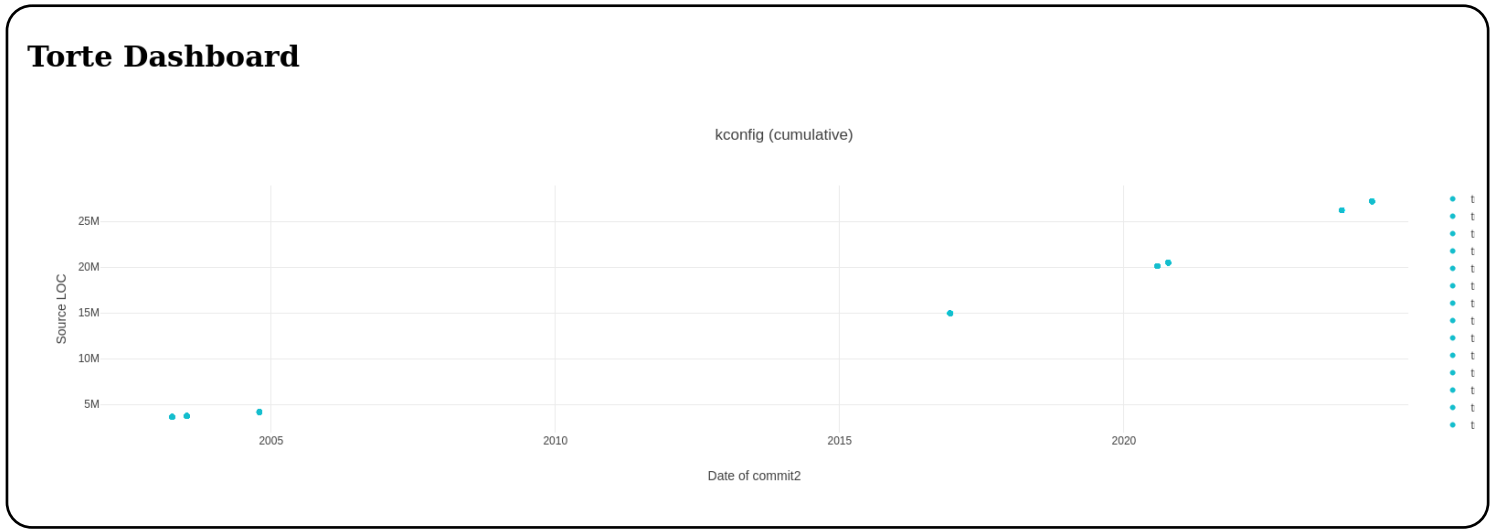
„*what you dont measure, you cannot control*“

- All metrics relate different projects in terms of size, complexity and variability
- Quantitative metrics give insight on system complexity
  - Lines of code, #Features, #Configurations
- Computation times hint at necessary effort of analysis
  - For instance, the Linux kernel has grown too complex to analyze
- Differentiate between Linux and non-Linux projects

Fig. 5: #Configurations of the Linux kernel [1]

3 Implementation

- Initial Setup
  - ExpressJS + Astro
- Second Setup
  - Flask + Svelte
- Third Setup
  - Serverless, static, and vanilla HTML

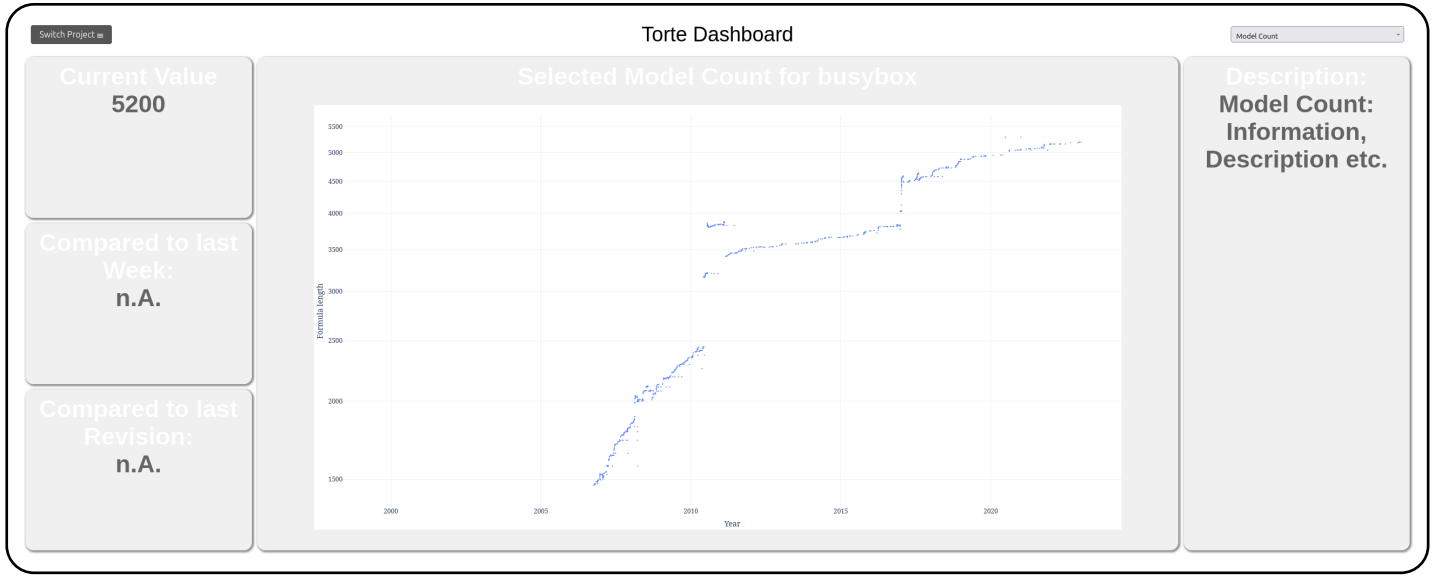


3 Implementation

- Initial Setup
  - ExpressJS + Astro
- Second Setup
  - Flask + Svelte
- Third Setup
  - Serverless, static, and vanilla HTML

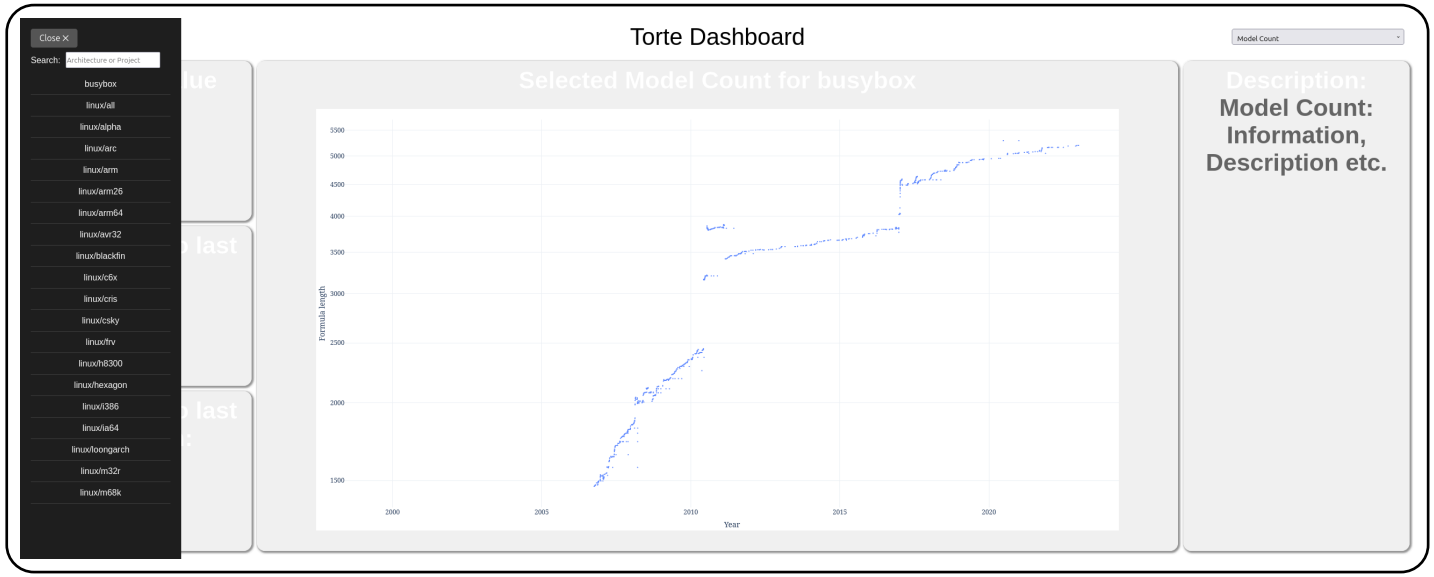
3 Implementation

- Initial Setup
  - ExpressJS + Astro
- Second Setup
  - Flask + Svelte
- Third Setup
  - Serverless, static, and vanilla HTML



3 Implementation

- Initial Setup
  - ExpressJS + Astro
- Second Setup
  - Flask + Svelte
- Third Setup
  - Serverless, static, and vanilla HTML



Final Iteration → Demo!

https://lupeterm.github.io (also kind of works on mobile!)

3.1 Workflow of Integrating New Data

1. Scientist generates new experiment results with `Torte`
2. Scientist modifies `gen_init.json`
3. The script autogenerates all figures and metrics
   - New Figures are saved directly into the frontend sources folder
   - New Metrics are merged into the pre-existing `init.json`
4. Run local development server
   1. Review the generated metrics and plots
   2. Repeat from 2., if necessary (e.g. incorrect `gen_init.json`)
5. Publish updated dashboard

   → Share results with the scientific community

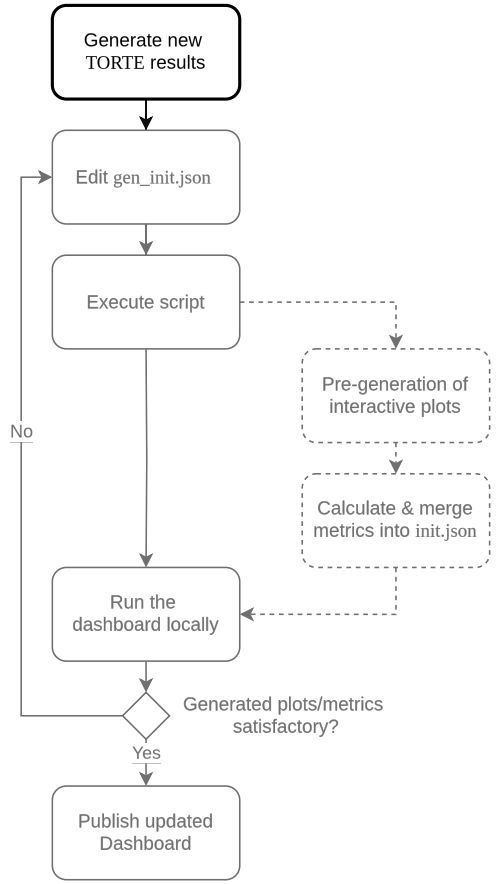


Fig. 10: Dashboard Extension Workflow

3.1 Workflow of Integrating New Data

1. Scientist generates new experiment results with `Torte`
2. Scientist modifies `gen_init.json`
3. The script autogenerates all figures and metrics
   - New Figures are saved directly into the frontend sources folder
   - New Metrics are merged into the pre-existing `init.json`
4. Run local development server
   1. Review the generated metrics and plots
   2. Repeat from 2., if necessary (e.g. incorrect `gen_init.json`)
5. Publish updated dashboard

   → Share results with the scientific community

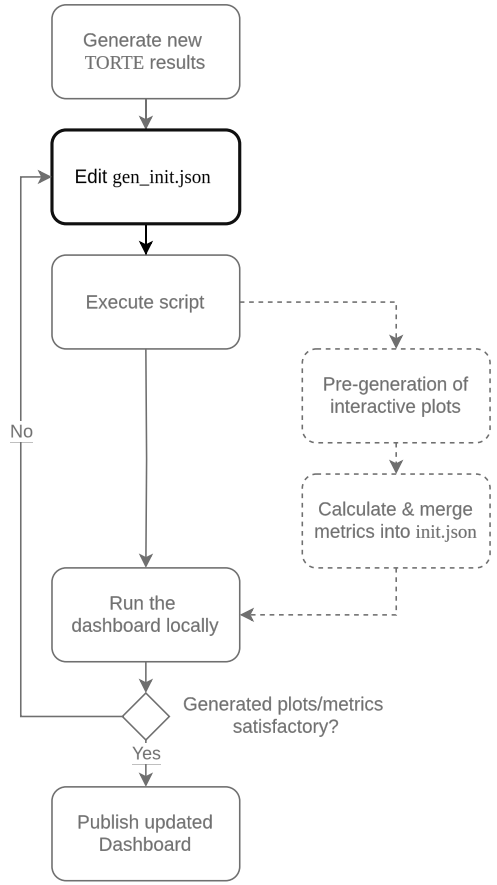


Fig. 11: Dashboard Extension Workflow

3.1 Workflow of Integrating New Data

1. Scientist generates new experiment results with `Torte`
2. Scientist modifies `gen_init.json`
3. The script autogenerates all figures and metrics
   - New Figures are saved directly into the frontend sources folder
   - New Metrics are merged into the pre-existing `init.json`
4. Run local development server
   1. Review the generated metrics and plots
   2. Repeat from 2., if necessary (e.g. incorrect `gen_init.json`)
5. Publish updated dashboard

   → Share results with the scientific community

```
{
    "nonLinux": {
        "busybox": {
            "output_directory": "./busybox",
            "ignore_systems": [
                "busybox-models"
            ],
            "figures_directory": "./figures"
        }
    },
    "linux": {
        "output_directory": "output-linux",
        "figures_directory": "./figures"
    }
}
```

Fig. 12: Dashboard Extension Workflow

3.1 Workflow of Integrating New Data

1. Scientist generates new experiment results with `Torte`
2. Scientist modifies `gen_init.json`
3. The script autogenerates all figures and metrics
   - New Figures are saved directly into the frontend sources folder
   - New Metrics are merged into the pre-existing `init.json`
4. Run local development server
   1. Review the generated metrics and plots
   2. Repeat from 2., if necessary (e.g. incorrect `gen_init.json`)
5. Publish updated dashboard

   → Share results with the scientific community

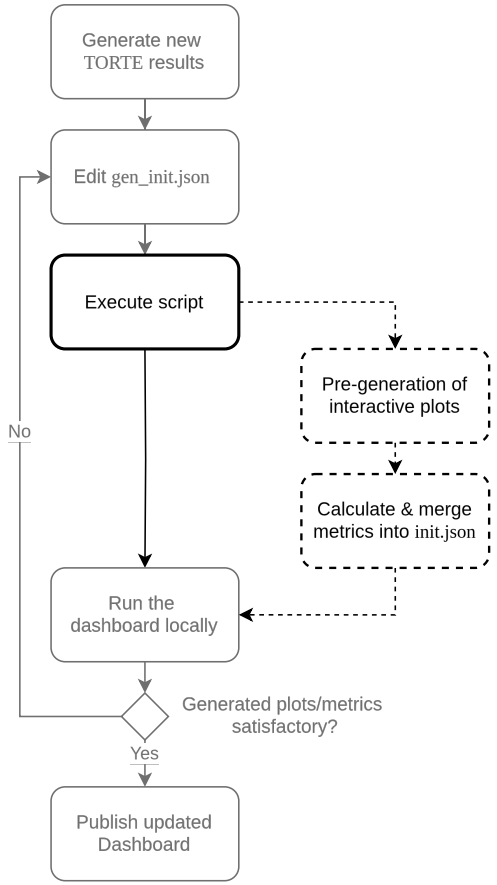


Fig. 13: Dashboard Extension Workflow

3.1.3 Metric Generation from Config

```
{
    "nonLinux": {
        "busybox": {
            "output_directory": "output-busybox",
            "ignore_systems": [
                "busybox-models"
            ],
            "figures_directory": "src/public/figures"
        }
    },
    "linux": {
        "output_directory": "output-linux",
        "figures_directory": "src/public/figures"
    }
}
```

Fig. 14: Entry in `gen_init.json`

→

```
"busybox": {
    "source_lines_of_code": {
        "currentValue": {
            "value": "209492 loc",
            "date": "From January 03, 2023"
        },
        "history": {
            "1-years-before": {
                "value": "205741 loc",
                "date": "December 04, 2021"
            },
            "2-years-before": {
                "value": "201837 loc",
                "date": "January 03, 2021"
            },
            "5-years-before": {
                "value": "189415 loc",
                "date": "January 04, 2018"
            },
            "10-years-before": {
                "value": "201076 loc",
                "date": "January 05, 2013"
            }
        }
    }
  // #configs, #features etc.
}
```

Fig. 15: Generated values in `init.json`

3.1 Workflow of Integrating New Data

1. Scientist generates new experiment results with `Torte`
2. Scientist modifies `gen_init.json`
3. The script autogenerates all figures and metrics
   - New Figures are saved directly into the frontend sources folder
   - New Metrics are merged into the pre-existing `init.json`
4. Run local development server
   1. Review the generated metrics and plots
   2. Repeat from 2., if necessary (e.g. incorrect `gen_init.json`)
5. Publish updated dashboard

   → Share results with the scientific community

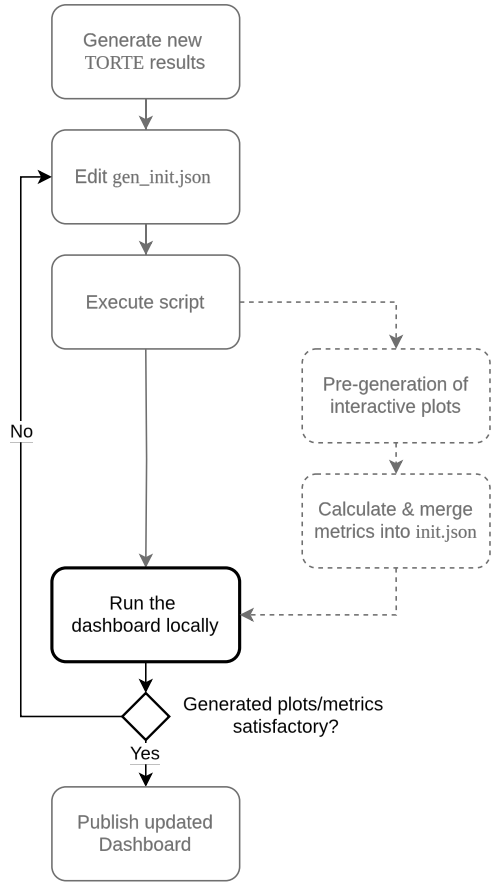


Fig. 16: Dashboard Extension Workflow

3.1 Workflow of Integrating New Data

1. Scientist generates new experiment results with `Torte`
2. Scientist modifies `gen_init.json`
3. The script autogenerates all figures and metrics
   - New Figures are saved directly into the frontend sources folder
   - New Metrics are merged into the pre-existing `init.json`
4. Run local development server
   1. Review the generated metrics and plots
   2. Repeat from 2., if necessary (e.g. incorrect `gen_init.json`)
5. Publish updated dashboard

   → Share results with the scientific community

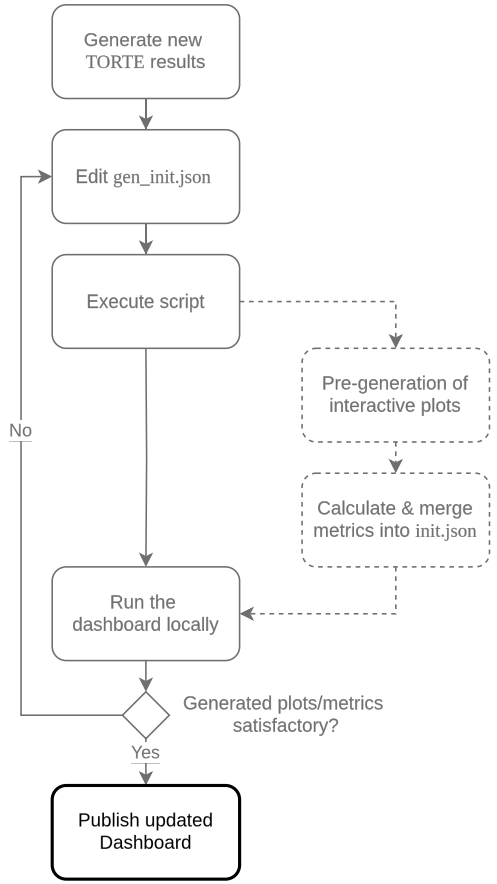


Fig. 17: Dashboard Extension Workflow

TORTE DASHBOARD

Select Project: linux/all

Select Plot: Number of Configurations

Current Value (KClause)

10^434 models
From June 30, 2013

History: KClause

1 year ago	10^430 models (-0.9%)
2 years ago	10^627 models (+30.8%)
5 years ago	10^570 models (+23.9%)
10 years ago	10^482 models (+10.0%)

Current Value (KConfigReader)

10^428 models
From May 19, 2011

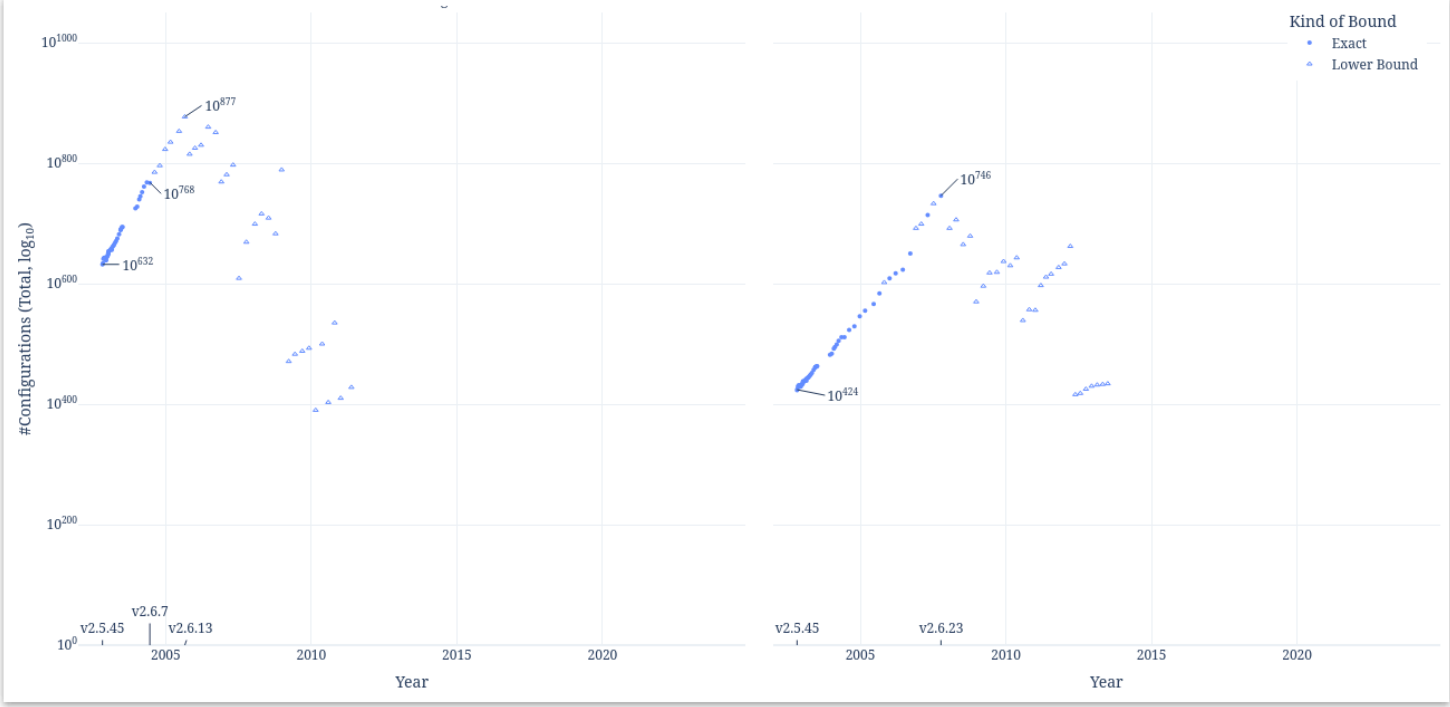
History: KConfigReader

1 year ago	10^535 models (+20.0%)
2 years ago	10^493 models (+13.2%)
5 years ago	10^769 models (+44.3%)

Number of Configurations

Number of Configurations: Description

You can edit this in the init.json under `plotData.model-count.description`



4 Conclusion

Stakeholder	**Benefit of a Scientific Dashboard**
Scientists	+ Quick insight on metric evolution and current state + Easy Comparison between projects and extractors + Supplementary to publications
Maintainer	+ Same benefits as above! + Automatic extraction of data & figure generation
Developer	+ Valuable lessons learned

Thanks for listening!

References

[1] E. Kuiter, C. Sundermann, T. Thüm, T. Hess, S. Krieter, und G. Saake, „How Configurable is the Linux Kernel? Analyzing Two Decades of Feature-Model History“, *ACM Trans. Softw. Eng. Methodol.*, Apr. 2025, doi: 10.1145/3729423.

[2] A. Sree-Kumar, E. Planas, und R. Clarisó, „Analysis of Feature Models Using Alloy: A Survey“, *Electronic Proceedings in Theoretical Computer Science*, Bd. 206, S. 46–60, 2016, doi: 10.4204/EPTCS.206.5.

[3] T. Thum, C. Kastner, S. Erdweg, und N. Siegmund, „Abstract Features in Feature Modeling“, in *2011 15th International Software Product Line Conference*, Aug. 2011, S. 191–200. doi: 10.1109/SPLC.2011.53.

[4] M. Nieke, J. Mauro, C. Seidl, T. Thüm, I. C. Yu, und F. Franzke, „Anomaly analyses for feature-model evolution“, *ACM SIGPLAN Notices*, Bd. 53, Nr. 9, S. 188–201, Apr. 2020, doi: 10.1145/3393934.3278123.

[5] „Evolution of the Linux Kernel Variability Model“, *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, S. 136–150, 2010. doi: 10.1007/978-3-642-15579-6_10.