Improving API Design Skills with the API Design Fest: Insights from a Preliminary Experiment with Students

Achim Röll
Universität Hamburg
post@achimroell.de

Leif Bonorden
Universität Hamburg
leif.bonorden@uni-hamburg.de

Abstract

APIs should be stable and demand a careful evolution, which requires a good initial design. Such API design skills usually come from experience, but the *API Design Fest* intends to compress such experience into a dense course. While the original training event was intended for practitioners, we are interested in the applicability in software engineering education. Thus, we conducted a slightly adopted *API Design Fest* with students and report on initial insights. While we find the overall event and its API design activities suitable, we recommend extended preparation on breaking changes for future *API Design Fests* with students.

1 API Evolution

As software systems evolve, APIs should remain relatively stable—serving as a layer of abstraction: Even if implementation details change, the interface ensures continued compatibility. However, if interfaces are set in stone, they may hinder evolution. Thus, interfaces may also change, and some of their modifications (*breaking changes*) may impair compatibility [2].

If APIs are well-designed, the need for breaking changes is minimized. Yet, API design skills are often learned from experience as most API designers don't have specific training [3].

2 API Design Fest

The *API Design Fest* by Jaroslav Tulach [1] is a training event that confronts API designers with typical situations of API evolution. The idea is to experience the challenges of API evolution that would typically occur over a long time in a dense course. Furthermore, as the participants attempt to break each other's solutions, a peer-learning effect is expected.

The original *API Design Fest* uses the Java programming language and comprises three phases:

1. **Initial Design:** The participants design an API and a corresponding implementation for boolean circuits supporting `AND`, `OR` and `NOT`.
2. **Evolution:** The participants extend their boolean circuits to circuits that handle all double values from `0` (equivalent to `false`) to `1` (equivalent to `true`) with adjusted definitions for `AND`, `OR` and `NOT`. Furthermore, clients should be able to define new circuit elements, e.g., `GTE` ($\geq$), by providing a suitable mathematical expression. At the same time, any client code working with the old API version must also work with the new API version.
3. **Competition:** The solutions are shared between the participants. Subsequently, the participants try to break each other's APIs by writing tests that succeed with the version from phase 1 but fail with the version from phase 2. Participants win points by breaking others' APIs and earn bonus points if their own API remains unbroken.

The event has been carried out several times, and Tulach has shared the main ideas of participants' solutions and their respective challenges [1].

3 Experiment with Students

We adopted the *API Design Fest* for an experimental setting with students: First, we extended the task descriptions to provide guidance for novel programmers. Second, we automated the provision of tasks and submission of solutions in GitLab. Finally, we added a fourth phase to compare the students' API design before and after the *API Design Fest*:

4. **Check**: The participants design an API for aggregators—independent from the circuit API. An aggregator takes an initial value and an aggregate function. The aggregate function updates the current value if a new value is added. A simple example is the sum aggregation: A new value is added to the current value. Additionally, the participants were given a hint that the API might be extended for more complex aggregate functions in the future, e.g., arithmetic mean aggregation.

We invited students from several informatics-related study programs to participate in the *API Design Fest*, leading to 12 participants from various bachelor's and master's programs with diverse programming experiences. Due to the COVID-19 pandemic,

we conducted this *API Design Fest* remotely. Furthermore, we scheduled 2 days for each phase to allow for asynchronous participation, although each phase comprises at most 2 hours of work.

In addition to unit tests and the competition results, we further analyzed the participants' solutions from all phases to identify common patterns (e.g., object factories), evaluate general code quality (e.g., names), and check API design principles (e.g., immutability of method parameters).

4 Insights

The participants chose various design approaches in the first phase, including a single static class, comprehensive class hierarchies, object factories, and extensive string parsing. All of these approaches were also observed by Tulach in his *API Design Fests*. On the other hand, some solutions described by Tulach were not given in our experiment. In the second phase, participants who applied information hiding in the first phase could easily extend their solutions. However, others encountered problems as their solutions from the first phase were too rigid or too specific.

In the third phase, the participants in our *API Design Fest* deviated more from Tulach's observations: Attempts to break each other's APIs were relatively simple and did not use more comprehensive techniques, e.g., Java reflection. Moreover, some participants missed even simple breaking changes, e.g., method renaming.

Analyzing the solutions further, we found general high quality regarding basic quality criteria, e.g., consistently and well-named methods and acceptable method length. The solutions were more diverse for other criteria: API documentation and code readability stretched from 'very well' to 'unacceptable', only some solutions used appropriate types or handled exceptions and corner cases.

In the fourth phase, we observed more comprehensive solutions, particularly solutions with more complex structures using appropriate (generic) types and factories. However, many solutions still did not handle corner cases or hide internal information.

Overall, we clustered the solutions into three categories: (1) collection of methods, (2) basic structure, (3) advanced design. The number of solutions per category is given in Table 1—indicating a general increase in quality from phase 1 to phase 4.

Category	(1)	(2)	(3)	N/A
# of solutions, phase 1	4	4	2	2
# of solutions, phase 4	1	4	6	1

Table 1: Observed solutions for each category from first and fourth phase (higher is better)

Finally, we asked the participants to reflect on the experiment: All students rate their solution in phase 4 equal or better than their respective solution in phase 1. However, asked about their learning effect, their statements range from 'learned almost nothing' to 'learned a lot'.

The first author's master's thesis [4] presents the experiment's results in more detail and discusses them further.

5 Conclusion

We conducted a preliminary experiment with students on the learning effects of the *API Design Fest* and found the training event generally suitable for educational settings. While we saw similar designs to Tulach's original training event, we observed a lack of comprehensive breaking attempts. We will use our observations to improve future *API Design Fests*:

Experiments with students: For future experiments with students, we will use on-site settings and provide extended preparation about breaking changes. Furthermore, we will investigate the respective influence of the second and third phases on the overall learning effect.

Experiments with practitioners: We intend to conduct experiments with experienced API designers to compare solutions and learning opportunities.

Transfer to other APIs: While Java APIs—and programming language APIs in general—remain an important topic, remote APIs—in particular, web APIs via REST—gained importance. Future work on the *API Design Fest* may include an adoption for such APIs.

References

[1] J. Tulach. *Practical API Design – Confessions of a Java Framework Architect*. Apress, 2012.

[2] L. Xavier et al. "Historical and impact analysis of API breaking changes: A large-scale study". In: *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2017, pp. 138–147.

[3] L. Murphy et al. "API Designers in the Field: Design Practices and Challenges for Creating Usable APIs". In: *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2018, pp. 249–258.

[4] A. Röll. "Wirksamkeit eines Programmierwettbewerbs zur Verbesserung von API-Design-Fähigkeiten". Master's Thesis. Hamburg, Germany: Universität Hamburg, 2022.