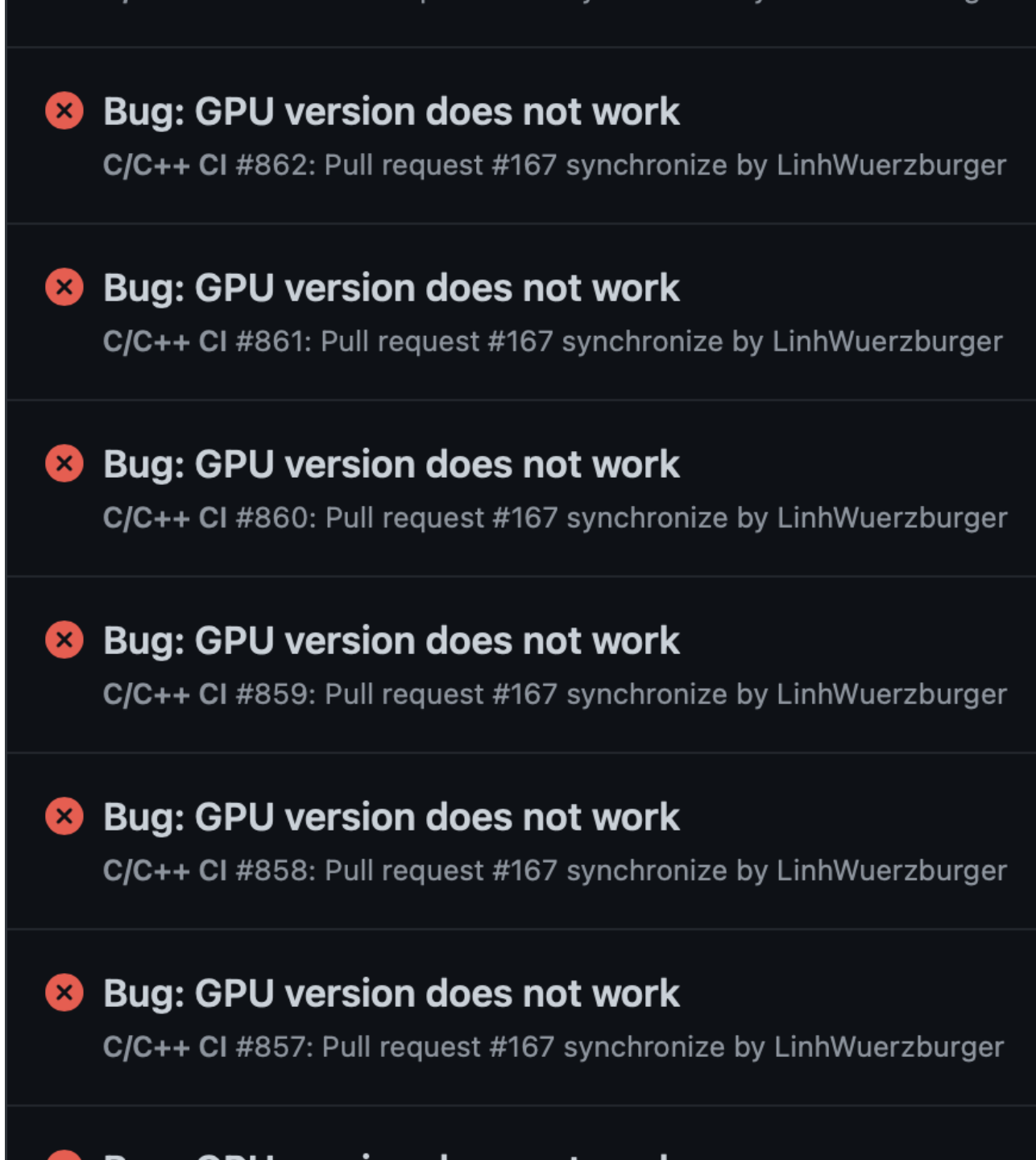
From Bug to Testing

Presenters

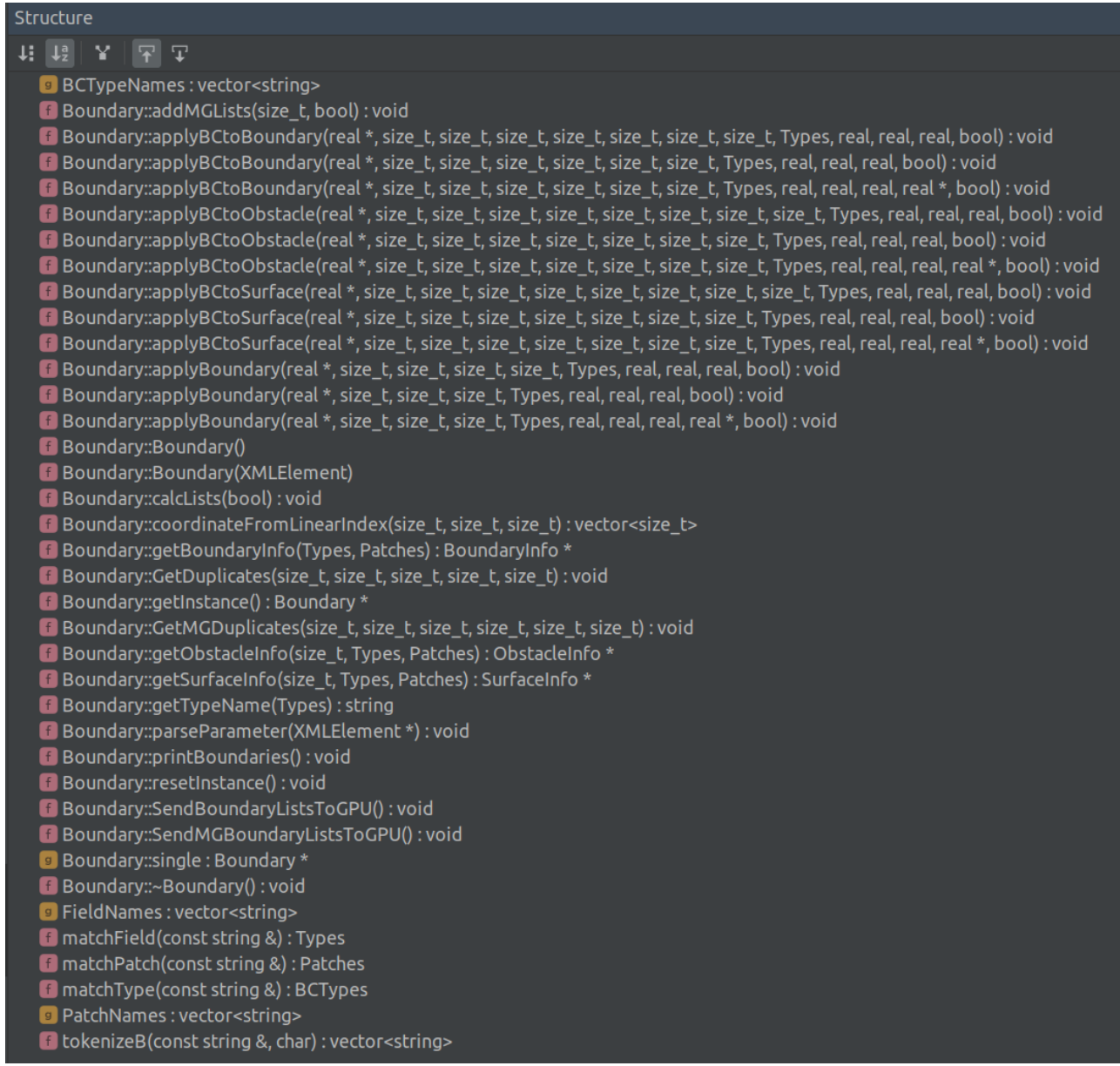
My Linh Würzburger
m.wuerzburger@fz-juelich.de

Kai Kratz
k.kratz@fz-juelich.de



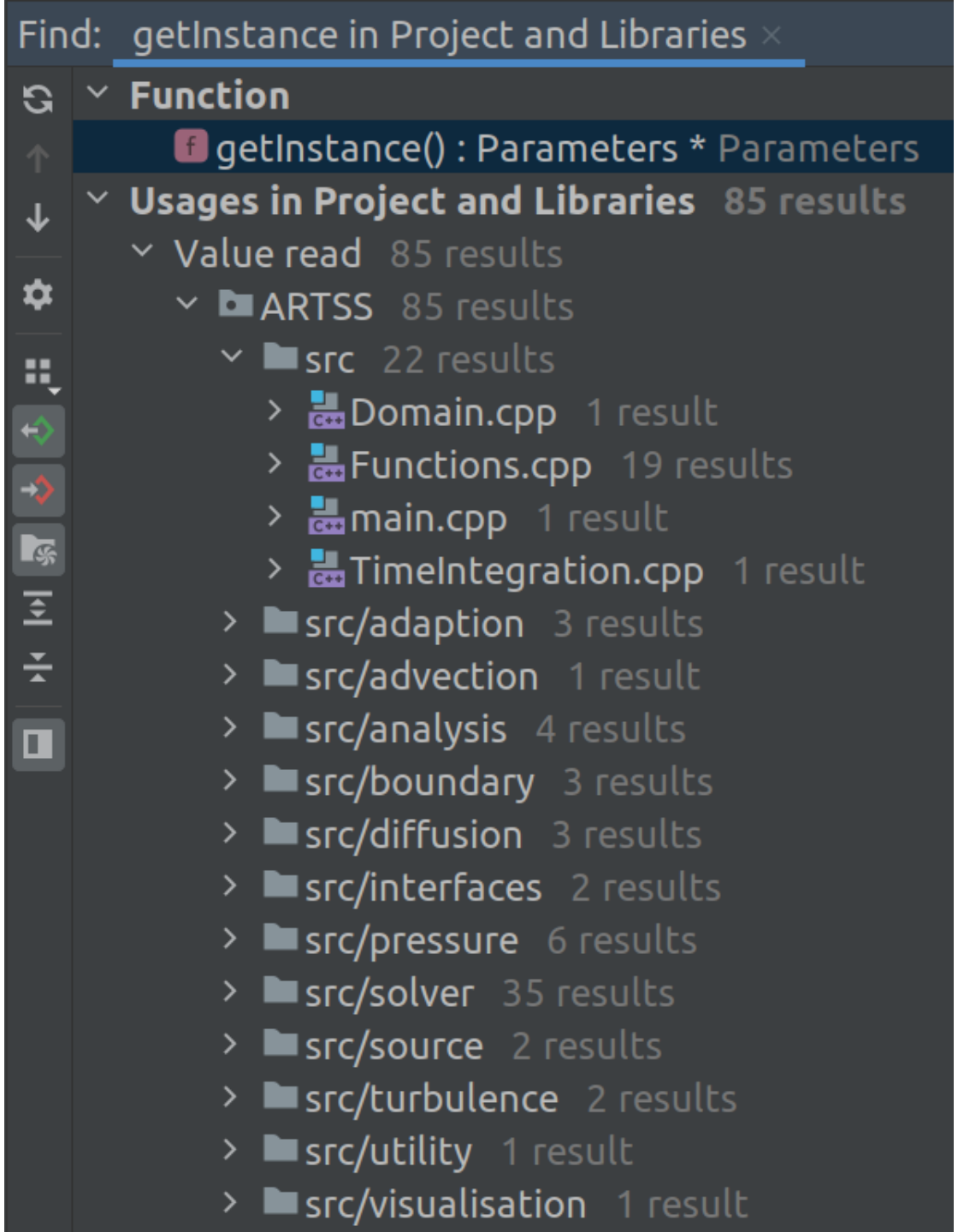
How did it start

- New code for me, new as PHD
- 8k lines of code, no comments :(
- Need to rewrite this
  -> now 7 small classes :)
- Results were bogus, no idea why
  -> add example / explanation
- Now lets write some tests to verify this is all working as expected



Ok I want to test but cannot

- Parameters `Singelton`
- Singelton constructed from `settings.xml`
- Many, many uses of this singelton



Lets have a look at a part of the Problem

```
class Parameters {
 public:
    static Parameters* getInstance();
    void parse(const std::string& filename);
    std::string get(const std::string& raw_path);
    real get_real(const std::string& raw_path);
    double get_double(const std::string& raw_path);
    int get_int(const std::string& raw_path);
    std::string get_filename() {return m_filename; }
    void printAllXMLAttributes(std::string prefix, tinyxml2::XMLElement *node);
    tinyxml2::XMLElement *get_first_child(const std::string &raw_path);
    tinyxml2::XMLElement *get_first_child(const char *raw_path);

 private:
    tinyxml2::XMLDocument* doc;
    static Parameters* single;
    Parameters() {this->doc = new tinyxml2::XMLDocument;}
    std::string m_filename;
};
```

What properties should `Parameters` have instead?

We are looking for specific properties to make this testable:

- Parameters to be "default constructible" w.o. reading a file.
- Parameters can easily be validated.
- Parameters should be explicitly given to consumers.
- Parameters should be as self documenting as possible.

How does the goal look like

```
struct VisualisationParameters {
    bool save_vtk;
    bool save_csv;
    size_t vtk_nth_plot;
    size_t csv_nth_plot;
};
struct LoggingParameters {
    std::string file;
    std::string level;
};
//... more structs ...
struct Settings {
    VisualisationParamters visualisation_parameters;
    LoggingParameters logging_parameters;
    // ...more fields ...
};
Settings parse_settings(const std::string &file_content);
Settings parse_settings_from_file(const std::filesystem::path &path);
```

How does the goal look like

```
std::string get_required_string(const Map &map, const std::string &key, const std::string &context) {
    try {
        return map.at(key);
    } catch (const std::exception &e) {
        throw config_error(fmt::format("Value {} is for {} required.", key, context));
    }
}
LoggingParameters parse_logging_parameters(tinyxml2::XMLDocument &doc) {
    std::string context = "logging";
    Map values = map_parameters(doc, context);
    LoggingParameters lp{};
    lp.file = get_required_string(values, "file", context);
    lp.level = get_required_string(values, "level", context);
    return lp;
}
```