

From Bug to Testing

Presenters

My Linh Würzburger

m.wuerzburger@fz-juelich.de

Kai Kratz

k.kratz@fz-juelich.de

✘ **Bug: GPU version does not work**

C/C++ CI #862: Pull request #167 synchronize by LinhWuerzburger

✘ **Bug: GPU version does not work**

C/C++ CI #861: Pull request #167 synchronize by LinhWuerzburger

✘ **Bug: GPU version does not work**

C/C++ CI #860: Pull request #167 synchronize by LinhWuerzburger

✘ **Bug: GPU version does not work**

C/C++ CI #859: Pull request #167 synchronize by LinhWuerzburger

✘ **Bug: GPU version does not work**

C/C++ CI #858: Pull request #167 synchronize by LinhWuerzburger

✘ **Bug: GPU version does not work**

C/C++ CI #857: Pull request #167 synchronize by LinhWuerzburger

✘ **Bug: GPU version does not work**

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

```
Structure
Structure
BCTypeNames : vector<string>
Boundary::addMGLists(size_t, bool) : void
Boundary::applyBCtoBoundary(real *, size_t, size_t, size_t, size_t, size_t, size_t, size_t, Types, real, real, real, bool) : void
Boundary::applyBCtoBoundary(real *, size_t, size_t, size_t, size_t, size_t, size_t, Types, real, real, real, bool) : void
Boundary::applyBCtoBoundary(real *, size_t, size_t, size_t, size_t, size_t, size_t, Types, real, real, real, real *, bool) : void
Boundary::applyBCtoObstacle(real *, size_t, size_t, size_t, size_t, size_t, size_t, size_t, Types, real, real, real, bool) : void
Boundary::applyBCtoObstacle(real *, size_t, size_t, size_t, size_t, size_t, size_t, Types, real, real, real, bool) : void
Boundary::applyBCtoObstacle(real *, size_t, size_t, size_t, size_t, size_t, size_t, Types, real, real, real, real *, bool) : void
Boundary::applyBCtoSurface(real *, size_t, size_t, size_t, size_t, size_t, size_t, size_t, Types, real, real, real, bool) : void
Boundary::applyBCtoSurface(real *, size_t, size_t, size_t, size_t, size_t, size_t, size_t, Types, real, real, real, real *, bool) : void
Boundary::applyBoundary(real *, size_t, size_t, size_t, size_t, Types, real, real, real, bool) : void
Boundary::applyBoundary(real *, size_t, size_t, size_t, Types, real, real, real, bool) : void
Boundary::applyBoundary(real *, size_t, size_t, size_t, Types, real, real, real, real *, bool) : void
Boundary::Boundary()
Boundary::Boundary(XMLMLElement)
Boundary::calcLists(bool) : void
Boundary::coordinateFromLinearIndex(size_t, size_t, size_t) : vector<size_t>
Boundary::getBoundaryInfo(Types, Patches) : BoundaryInfo *
Boundary::GetDuplicates(size_t, size_t, size_t, size_t) : void
Boundary::getInstance() : Boundary *
Boundary::GetMGDuplicates(size_t, size_t, size_t, size_t, size_t) : void
Boundary::getObstacleInfo(size_t, Types, Patches) : ObstacleInfo *
Boundary::getSurfaceInfo(size_t, Types, Patches) : SurfaceInfo *
Boundary::getTypeName(Types) : string
Boundary::parseParameter(XMLMLElement *) : void
Boundary::printBoundaries() : void
Boundary::resetInstance() : void
Boundary::SendBoundaryListsToGPU() : void
Boundary::SendMGBoundaryListsToGPU() : void
Boundary::single : Boundary *
Boundary::~Boundary() : void
FieldNames : vector<string>
matchField(const string &) : Types
matchPatch(const string &) : Patches
matchType(const string &) : BCTypes
PatchNames : vector<string>
tokenizeB(const string &, char) : vector<string>
```

Ok I want to test but cannot

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

The screenshot shows an IDE search results window. At the top, the search bar contains the text "Find: getInstance in Project and Libraries" with a close button. Below the search bar, the results are organized into a tree view. The first level is "Function", which contains a single entry: "getInstance() : Parameters * Parameters". The second level is "Usages in Project and Libraries" with a total of 85 results. This is further broken down into "Value read" (85 results), which is then categorized by folder. The "ARTSS" folder contains 85 results, and its "src" subfolder contains 22 results. The "src" folder is further divided into several subfolders, each with a specific number of results: Domain.cpp (1), Functions.cpp (19), main.cpp (1), TimeIntegration.cpp (1), src/adaption (3), src/advection (1), src/analysis (4), src/boundary (3), src/diffusion (3), src/interfaces (2), src/pressure (6), src/solver (35), src/source (2), src/turbulence (2), src/utility (1), and src/visualisation (1).

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 {
    VisualisationParameters 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;
}
```