D Practical Experience Report

Stefan Rohe, Funkwerk
Eugen Wissner, Caraus

Content

- Motivation
- History
- Fails
- Wins
- Others
  - Caraus
  - sociomantic
  - remedygames

About Funkwerk

Passenger Information Systems - Mobile and Stationary Systems

Munich, Berlin, FraPort, Finland, Norway, Austria, Luxembourg, Switzerland, Romania, …

About Funkwerk (2008)

Passenger Information Systems since 198x

- Fortran / C / C++ Core
- Java Swing UI

New Features were implemented in the language of choice not in the service where it’s domain is belonging to

20 years support for HW/SW

Dan North

Teams work best with a common set of tools

Revolutionary Approach

Starting with a new language - No statements like “we do it like this, cause we did it since ever like this”

Value clean code more than performant code

History

2008 - New Dev Lead - after long time of no dev leadership

2008 - Started first D1-Tango Project with Java Swing UI

2010 - umbrello to D1 stub generator

2011 - Extracted first D1 Idioms

2011 - ddepend for checking intended and actual package relationships

2012 - Converted nearly everything already to D2-Phobos

2014 - D Backend with HTML5 Frontend

2015 - D to UML

Fails

Multiple causes

- first D project
- D1 Tango has been stopped evolving and is now dead / halfdead
- replaced one monolith against another one
- TLS confusion
- ...

Fail: Unittest blocks for testing side effects (I)

Unittests blocks before used for IO / Network / Database and many other

Better:

- distinguish between whitebox and blackbox tests
- unittest blocks should not test side effects
- BDD black box test around the component

```
module fib;

uint fib(uint n) {
    return (n < 2) ? n : fib(n - 1) + fib(n - 2);
}

unittest {
    assert(fib(0) == 1);
    assert(fib(1) == 1);
    assert(fib(2) == 2);
    assert(fib(3) == 3);
    assert(fib(4) == 8);
}
```

Fail: Unittest blocks for testing side effects (II)

- unittest blocks do not allow you to name your tests
- no assert Helpers (assertEquals, assertIn) for better error messages
- no distinction between Error and Failure

Better:

- usage of a xUnit Framework for Tests requireing multiple lines

```
module json;

string encode(JSON value) {
    return ...;
}
-----------------------------------------------------
module jsonTest;

import dunit;
import json;

@Test
void encodesEmptyObject() {
    auto emptyJsonObject = JSON();
    assertEquals("{}", emptyJsonObject)
}
```

https://github.com/linkrope/dunit (JUnit4 Fork?)

Fail: D1 GC

- relied too much on Garbage Collector because copied the Java approach; everything is a Class
- Mark/Sweep GC; is a pointer?
- endless loops, seg faults after~½GB
- stop the world

Better:

- 64 Bit instead of 32 Bit
- D2, instead of D1
- don't use the GC for everything
- Option for Precise GC

```
module A;

// lives on the heap; GC cares about object lifetime
class Journey {
    string number;
    ...
}

// lives on stack; scope cares about object lifetime
struct Journey {
    string number;
    ...
}
```

Fail: Logging

- long time no logging module
- usage of D1-Tango Logging
- nowadays experimental.logger

Better:

- simple logging with simple API
- define what each logging level stands for
  - trace, info, warn, error, fatal
- Open: Json-Logging

https://github.com/linkrope/log

```
module A;

import util.log;

void main() {
    log = Log(stderrLogger(LogLevel.warn));
    log.warn("mostly harmless: %s, %s", 23, 42);
    // argument to warn will be evaluated lazy
    log.error("don't panic");
    try {
        throw new Exception("something broken");
    } catch (Exception exception) {
        log.fatal(exception);
    }
}
```

Fail: Handcrafted Module Dependency

- compiled module for module
- needed a dependency graph for this
- grep import || dmd -deps

Better:

- dmd src/**/*.d -ofbuild/binary
- Inlining across modules
- faster, not so error prone
- Templates need to be compiled anyway

```
module A;

uint a(uint value) {
    return ...;
}
-----------------------------------------------------
module B;

import A;

uint b(uint value) {
    return a(value);  // a will be inlined if appropriate
}
```

Fail: Singletons

- Singletons could be done by just a mixin. Nearly no run time overhead
- not testable

Better:

- (Manual) Dependency Injection
- Wiring at Compile time (module; static this { ... })

```
module A;

class JourneyRepository {
    mixin(Singleton);
}
-------------------------------------------------------
module B;

class JourneyRepository {
}

class JourneyRepoUser {
    this(JourneyRepository repository) {
        this.repository = repository;
    }
}
```

Wins

- awesome TDD Cycle Times
  - Whitebox Tests.duration <= 10.seconds
  - Blackbox Tests.duration <= 5.minutes
- No C++ Compiler Guru Knowledge needed anymore
- No Debugging cause of high coverage

Win: UML (I)

UML is unbeatable for a rough overview of the whole component

Ubiquitous language for programmers; no value if not in sync with code

First Approach: Generate Boilerplate Code

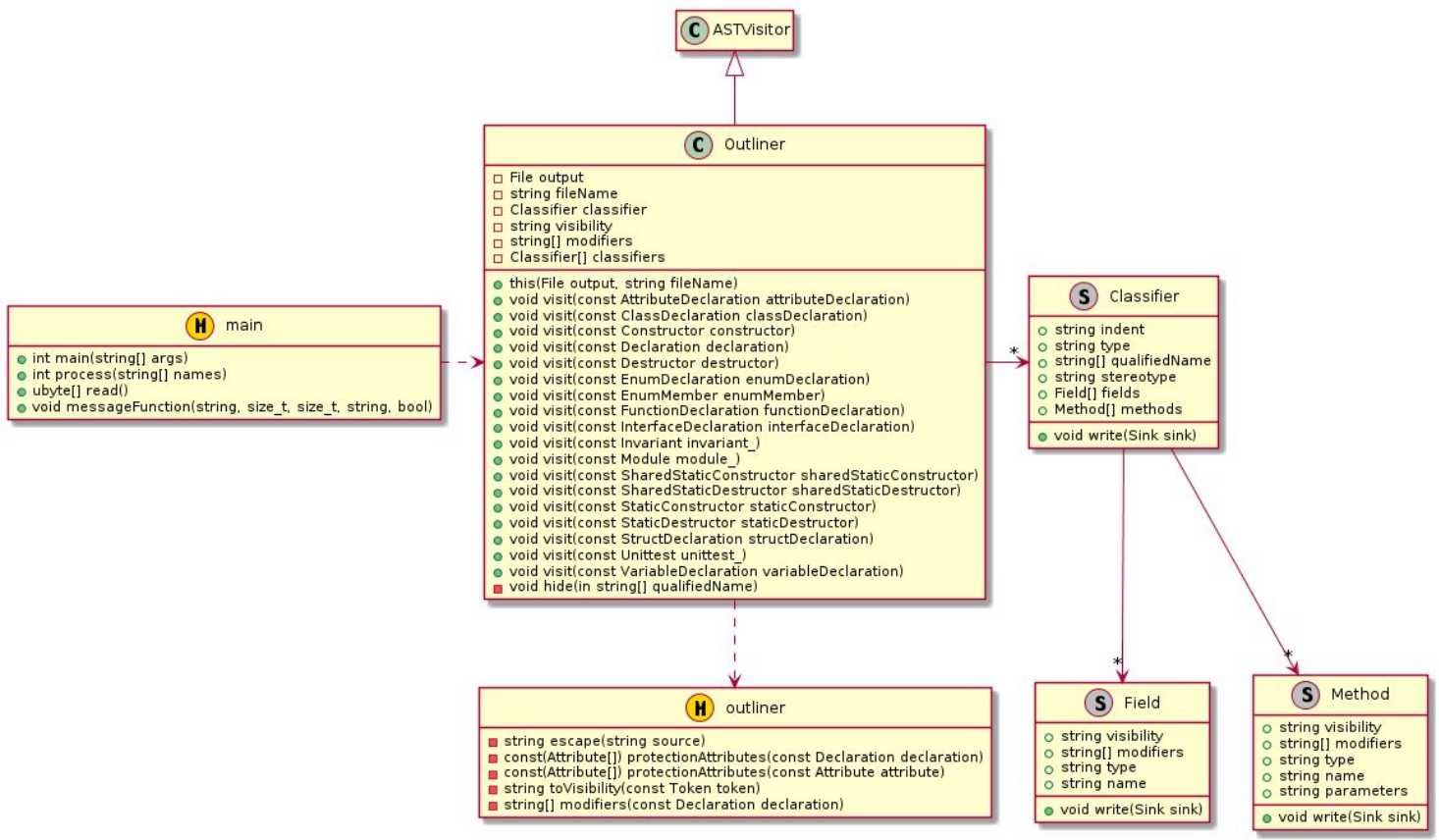
Generated Boilerplate Code from a Umbrello Model into D Stubs. (later ArgoUml)

Basically Getters, Setters, (Invariants, Pre-, Postconditions)

Current Approach: Avoid Boilerplate Code

Generate UML out of D Code

Win: UML (II)


ASTVisitor
Outliner
File output
string fileName
Classifier classifier
string visibility
string[] modifiers
Classifier[] classifiers
this(File output, string fileName)
void visit(const AttributeDeclaration attributeDeclaration)
void visit(const ClassDeclaration classDeclaration)
void visit(const Constructor constructor)
void visit(const Declaration declaration)
void visit(const Destructor destructor)
void visit(const EnumDeclaration enumDeclaration)
void visit(const EnumMember enumMember)
void visit(const FunctionDeclaration functionDeclaration)
void visit(const InterfaceDeclaration interfaceDeclaration)
void visit(const Invariant invariant_)
void visit(const Module module_)
void visit(const SharedStaticConstructor sharedStaticConstructor)
void visit(const SharedStaticDestructor sharedStaticDestructor)
void visit(const StaticConstructor staticConstructor)
void visit(const StaticDestructor staticDestructor)
void visit(const StructDeclaration structDeclaration)
void visit(const Unittest unittest_)
void visit(const VariableDeclaration variableDeclaration)
void hide(in string[] qualifiedName)
main
int main(string[] args)
int process(string[] names)
ubyte[] read()
void messageFunction(string, size_t, size_t, string, bool)
Classifier
string indent
string type
string[] qualifiedName
string stereotype
Field[] fields
Method[] methods
void write(Sink sink)
outliner
string escape(string source)
const(Attribute[]) protectionAttributes(const Declaration declaration)
const(Attribute[]) protectionAttributes(const Attribute attribute)
string toVisibility(const Token token)
string[] modifiers(const Declaration declaration)
Field
string visibility
string[] modifiers
string type
string name
void write(Sink sink)
Method
string visibility
string[] modifiers
string type
string name
string parameters
void write(Sink sink)
*
*
*

Win: UML (III)

rake generate for generate the Model

Generate the classes, do not generate the relationships between the classes

umbrello / argouml not readable and difficult for changing

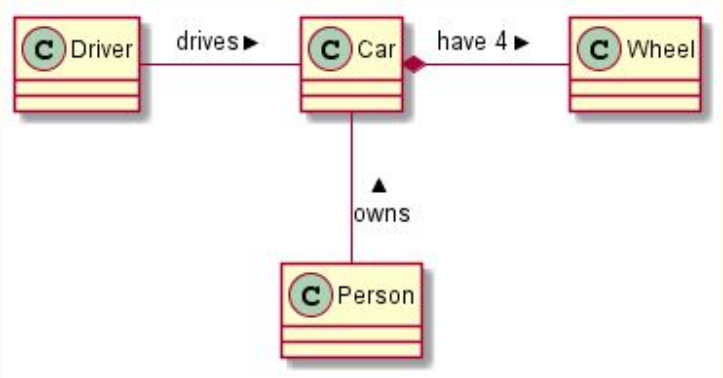
→ opted for PlantUML (http://plantuml.com/)

https://github.com/funkwerk/d2uml

```
@startuml
class Car

Driver - Car : drives >
Car *- Wheel : have 4 >
Car -- Person : < owns

@enduml
```



Win: Package Dependency

UML can model high level structure

depend to check package relationships

comparison of intended and actual dependencies

```
error: unintended dependency model -> controller
```

Probably belonging to D Standard Tools after Winter

https://github.com/funkwerk/depend

Intended Dependencies within a plantuml file

```
package model {}
package view {}
package controller {}

controller ..> view
controller ..> model // redundant
view .> model
```

Win: Properties / Getter / Setter (I)

C#:

```
using System;

class Message {
    public string text { get; set; }
}

class Hello {
    static void Main() {
        Message msg = new
Message();

        msg.text = "Hello, World!";
        Console.WriteLine(msg.text);
  }
}
```

D:

```
import std.stdio;

class Message {
    private string text_;

    @property inout(string) text() inout {
        return text_;
    }
    @property void text(in string value) {
        text_ = value;
    }
}
void main() {
    Message msg = new Message();

    msg.text = "Hello, World!";
    writeln(msg.text);
}
```

Win: Properties / Getter / Setter (II)

```
import accessors; // avail at http://github.com/funkwerk/accessors by tomorrow
import std.stdio;

class Message {
    @Read @Write
    private string text_;

    mixin(GenerateFieldAccessors);
}

void main() {
    Message msg = new Message();

    msg.text = "Hello, World!";
    writeln(msg.text);
  }
```

Win: Properties / Getter / Setter (III)

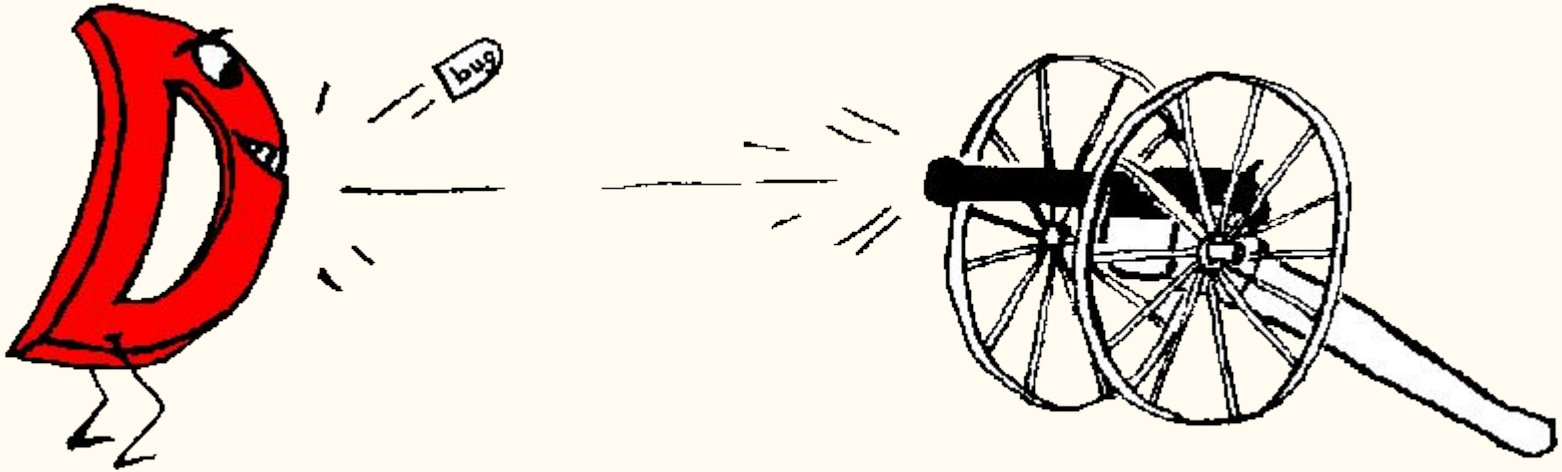
- @Read
- @Write
- @ConstRead
- @RefRead

```
class Klass {
  @Read("public") @Write("protected")
   private string text_;

   mixin(GenerateFieldAccessors);
}
```

Win: Contract Based Programming (I)

- https://dlang.org/spec/contracts.html



Win: Contract Based Programming (II)

- pre-/postconditions assert before/after every function call
- against programming errors, not against IO errors
- issues found fast, cause just the stacktrace is most of the times enough to understand / repair
- asserts are enabled within production code!
- Compiler uses Contracts for CodeGen

```
module A;

int square_root(int n)
in {
    assert(n >= 0); // usually uint should be preferred
}
out(result) {
    assert(result >= 0);
    assert(result * result == n);
}
body {
    return n.sqrt;
}
```

Win: Contract Based Programming (III)

- Invariants are checks at every public function call
- no invalid objects anymore; don't need to think about strange corner cases when debugging
- Contract Inheritance

```
struct JourneyRepository {
  private Journey[JourneyId] journeys;
  invariant {
    this.journeys.keys.all!"a !is null";
    this.journeys.values.all!"a !is null";
  }
  Journey get(JourneyId id)
  in {
    assert(id !is null);
  }
  out (result) {
    assert(result !is null);
  }
  body {
    if (id in this.journeys) { return this.journeys[id]; }
    throw new Exception(
        format("journey %s not found", id));
} }
```

Win: Uniform Function Call Syntax (UFCS)

- xml, json … for implicit schema
- for readable constants
- for DSLs
  - select(foo).where().and().but()
- for Pipeline Programming
  - list.do_something.do_another_thing

```
module A;

void main() {
    // implicit json/xml schema
    import util.text.json
    auto json = readJsonSomehow(...);
    auto timeout = json.require!Duration(”timeout”);
    uint port = json.require!uint(“port”)

    // pretty readable constants
    socket.defaultTimeout = 30.seconds;

    // awesome for DSLs / Pipeline Programming
    trains.filter((train) -> train.number.isOdd).
      map((train) -> format(“#%s”, train.number)).
    sort;
}
```

Win: Module Initializers for link time features

- Linkage decides if a feature is enabled or not
- unittest whole codebase
- tailor down the binary to what's really needed.
- nearly no additional test effort, because the black box tests exists anyway, just need to be disabled/enabled based on the features

```
module A;

import B;

shared static this {
    B.register(A);
}
```

Win: Linkage with C

Simple Interface to existing Libraries

Usage for Kafka, MQ, various TextToSpeech Vendors, …

Simple Interface preferred for Testing!

Also existing Library Headers for many popular C++ Libraries

Experimenting with gtkd; used before qtd

Caraus

Native D web framework

Aim: Developing of web-applications (first of all eCommerce, Websites, then servers)

Caraus

Why started with D?
(PHP/NodeJS/C -> Hack -> Haskell -> Rust -> D)

- Familiar C-like syntax
- No attitude to compatibility with C
- Systems programming language with the syntax of a modern high-level language (C#, Java)
- Good OOP and Generics
- Static typing
- Big enough community; good documentation for the language core

Caraus Fails

- Relative buggy official compiler; not best generated code
- Missing native libraries (e.g. DB, cryptography/TLS, Parser (HTTP/FastCGI))
- Not extendable structs (for Windows C-API)
- Missing tools (build systems (to integrate with other projects), mature IDE/editor support)
- struct: no default constructor; class: no opAssign.
- keyword jungle: inout, scope, ref (& and *), const, immutable, pure, nothrow, @safe, @nogc, shared, __gshared, @property
- High memory usage sometimes (dub fails on system, where gcc can build huge projects)

VibeD

- Not native (very slow adoption of libasync and botan)
- No usable interface to implement database support other than MongoDB, Redis (new)
- The same for templates other than Diet
- Compile time templates
- Not extendable (structs, final classes/methods, overuse of UFCS; composition over inheritance)
- Missing DI, no support for MVC-similar patterns and plugins
- No HTTP/2 support (etcimon's fork with HTTP/2 support)
- Not reusable components

Caraus. Current work

tanya: general purpose library with alternative memory management; base for the framework.

Features:

- Cross-platform event loop (inspired by asyncio and libev API)
- Containers (Queue, Vector, List)
- Advanced URL-Parsing (mostly rewritten after PHP's parse_url)
- Tools for manual memory management (native D basic malloc/free, reference counting)
- Secure random number generation (currently linux only)

Work in progress:

- Math and cryptography (TLS 1.2, AES, RSA, RNG)
- HTTP parser (similar to NodeJS' one)
- Documentation

Sociomantic (https://www.sociomantic.com/)

- real time ad bitting
- biggest D shop world wide; organizer of the DConf
- Berlin based; hired the International D Who-Is-Who
- D1-Tango
- now on the merge to a D2-Tango code base
- https://github.com/sociomantic-tsunami/ocean
- Low Level General Purpose Library with very few allocations
- Merge 100% automatic
- Developed own GC; Forking, no stop the world

Remedy Games (http://www.remedygames.com/)

- Company behind Max Paine
- Quantum Break is the first AAA-Game done with D
- http://schedule.gdceurope.com/session/d-using-an-emerging-language-in-quantum-break
- http://dconf.org/2016/talks/watson.html
- XBox One, Windows 10
- Rapid Iteration Framework for binding C/C++ (https://github.com/Remedy-Entertainment/binderoo)
- “Plugins” done in D

Further Information

Idioms: https://p0nce.github.io/d-idioms/

Areas of D Usage: https://dlang.org/areas-of-d-usage.html

Organisations using D: https://dlang.org/orgs-using-d.html