Now we are porting

(a short time ago in a company not so far away)

Mihails Strasuns
me@dicebot.lv

Some basics:

- Sociomantic Labs : Real-Time Bidding startup founded in 2009
- http://en.wikipedia.org/wiki/Real-time_bidding
- One of biggest D users (and growing, aiming for ~50 full time D developers in nearby future)
- More info at http://dconf.org/2014/talks/clugston.html
- Have been stuck with D1 for all that time
- Until recently

Mihails Strasuns
me@dicebot.lv

Incoming:

- Porting process explanation
- Progress report
- Evaluating impact of various D1 → D2 changes
- Why breaking things is important
- Why deprecations matter

Mihails Strasuns
me@dicebot.lv

Migration process

Challenges

- can't compromise daily development
- lots of code
- lots of code in shared dependencies
- minimize communication overhead
- real-time services are inherently more fragile

Requirements

- must happen in parallel with feature development
- avoid maintenance of multiple versions
- must be able to rollback to D1 compiler/runtime at any moment

Mihails Strasuns
me@dicebot.lv

Observations:

- Some of the changes are simply due to D2 being more strict
- Can hide semantic differences behind wrapper functions / templates
- A few more changes remain but can be automated

Need better diagnostics:

```
const int* oops;
void main()
{
    auto x = 042;
    do { } while (true)
}

$ dmd1 -v2 -w -o- sample.d
sample.d(6): Warning: octal literals 042 are not in D2, use std.conv.octal!42
instead or hex 0x22 [-v2=octal]
sample.d(7): Warning: D2 requires that 'do { ... } while(...)' end with a ';' [-
v2=syntax]
sample.d(2): Warning: There is no const storage class in D2, make variable 'oops'
non-const [-v2=const]
```

Mihails Strasuns
me@dicebot.lv

transition.d

```
template Typedef(T, istring name, T initval)
{
    static assert (name.length, "Can't create Typedef with an empty identifier");
    version(D_Version2)
    {
        mixin(`
            enum Typedef =
                ("struct " ~ name ~
                "{ " ~
                T.stringof ~ " value = " ~ initval.stringof ~ ";" ~
                "alias value this;" ~
                "this(" ~ T.stringof ~ " rhs) { this.value = rhs; }" ~
                " }");
        `);
    }
    else
    {
        const Typedef = ("typedef " ~ T.stringof ~ " " ~ name ~
            " = " ~ initval.stringof ~ ";");
    }
}
```

- Hosts all migration utilities and wrappers
- Encapsulates version blocks

Mihails Strasuns
me@dicebot.lv

Dealing with const

```
template Const(T)
{
    version(D_Version2)
    {
        mixin("alias const(T) Const;");
    }
    else
    {
        alias T Const;
    }
}
version(D_Version2)
{
    mixin("
        alias immutable(char)[] istring;
        alias const(char)[]     cstring;
        alias char[]            mstring;
    ");
}
else
{
    alias char[] istring;
    alias char[] cstring;
    alias char[] mstring;
}
```

- Makes it possible to define const correctness for D1 functions
- By far most effort consuming part of the basic porting
- Plain `const` keyword used only for manifest constants
- Works, but can be very challenging with templates (see next slide)

Mihails Strasuns
me@dicebot.lv

Const!(T) + templates

```
T[] escape(T) (T[] src, T[] dst = null);
```

Original D1 template function

```
Const!(T)[] escape(T) (Const!(T)[] src, T[] dst = null);
```

Nope : can't infer `Const!(T)`

```
TC[] escape(T, TC) (T[] src, TC[] dst = null);
// static assert (is(Unqual!(T) == Unqual!(TC)));
```

Nope : wrongly inferred `null` type

```
TC[] escape(T, TC = Unqual!(T)) (T[] src, TC[] dst = null);
// static assert (is(Unqual!(T) == Unqual!(TC)));
```

Actual ported code

Mihails Strasuns
me@dicebot.lv

d1to2fix

- Based on https://github.com/Hackerpilot/libdparse
- Takes care of changes trivial to automate and annoying to do manually
- Last step that turns D1 source code into working D2 source code
- Imperfect but good enough for our needs

```
const something = init;
// ->
enum something = init;
```

```
struct S {
    S* foo() {
        return this;
    }
}
// ->
struct S {
    S* foo() {
        return (&this);
    }
}
```

Mihails Strasuns
me@dicebot.lv

Runtime

```
void appendTo(ref int[] dst)
{
    dst ~= 42;
}

void main()
{
    int[] buffer = [ 1, 2, 3, 4 ];
    auto slice = buffer; slice.length = 0;
    appendTo(slice); appendTo(slice);
    // ok in D1, fails in D2
    assert (buffer == [ 42, 42, 3, 4 ]);
}
```

```
// transition.d
void enableStomping(T)(ref T array)
{
    version(D_Version2)
    {
        assumeSafeAppend(array);
    }
    else
    {
        // no-op
    }
}
```

Mihails Strasuns
me@dicebot.lv

GC

- Latency requirements more important than throughput – special coding style that rarely triggers GC
- Use custom CDGC : http://dconf.org/2013/talks/lucarella.html
- Proof of concept port to D2 https://github.com/D-Programming-Language/druntime/pull/985
- Will likely to be redone completely on top of existing druntime GC
- Remains speculative topic until at least one real-time application is fully ported and can be benchmarked

Mihails Strasuns
me@dicebot.lv

Porting process summary

Stage 1

Ensure the code compiles with `dmd1 -v2 -w` using helpers from `transition.d` – fixes as many issues as possible while staying within D1 toolchain.

Stage 2

Try running `d1to2fix` and compiling the code with `dmd2`, fixing any remaining issues (mostly const correctness). Revert `d1to2fix` changeset to ensure that it still compiles with `dmd1`

Stage 3

Regression control and maturity. Add Jenkins job that ensures application master stays compatible with D2 and automatically pushes output of `d1to2fix` to dedicated branch.

Do any runtime profiling as necessary.

Mihails Strasuns
me@dicebot.lv

https://www.sociomantic.com/search/tag/dlang

Mihails Strasuns
me@dicebot.lv

sociomantic
from dunnhumby

Tiers of language changes

“What can possibly go wrong?”

Mihails Strasuns
me@dicebot.lv

Good

```
// Warning: D2 requires that 'do { ... } while(...)' end with a ';'
// Deprecation: implicitly overriding base class method A.foo with B.foo deprecated
// Deprecation: function mymod.foo is deprecated - use mymod.bar instead
```

- Fix is straightforward and suggested by the error message
- Gives time to adjust for a change
- No fundamental change in semantics

Mihails Strasuns
me@dicebot.lv

Bad

```
const MyClass obj;
obj.foo();
// Error: mutable method mod.MyClass.foo is not callable using a const object
```

- Total change in semantics – hard to even track the point of failure without dedicated diagnostics
- No 1-to-1 replacement for old semantics, impossible to automate
- No intermediate adjustment step
- Can't be done in small chunks (transitivity)

Mihails Strasuns
me@dicebot.lv

Ugly

```
int[] arr1 = [ 1, 2, 3 ];
int[] arr2 = arr1[0 .. $];
arr2.length = 0; arr2 ~= 42;
assert (arr1[0] == 42);
```

- Manifests only as runtime change
- May result in silent performance degradation with no error
- Impossible to track down without custom runtime build and/or performance profiling
- Primary suspect effect : can never be sure it is all fixed

Mihails Strasuns
me@dicebot.lv

ROI

- Suggested by Don Clugston as a way to measure how justified the change is
- “Investment” from the language developer PoV – how hard it is to implement and maintain, how much does it complicate the language.
- “Investment” from the language user PoV – how much effort it takes to upgrade existing code, how much disruption in daily development it causes

Mihails Strasuns
me@dicebot.lv

#pleasebreakmycode

Getting the Devil from the Details

Mihails Strasuns
me@dicebot.lv

sociomantic
from dunnhumby

Stance on breaking changes

- Necessary : technical debt becomes more costly with increased team size
- Even nitpick changes are justified if they result in an improved learning curve and communication
- Can't afford to be stable in moving industry
- Lack of hope makes developers unhappy

- Process matters : same breakage can be both welcome and hated depending on how well it was presented and managed
- Deprecations matter : “normal” development takes priority

Mihails Strasuns
me@dicebot.lv

Bugfixes

User is reading the changelog. His reaction is likely to be:

“Oh. We'd better not have any code that uses it in production. Sound the alarm!”

Just break it

2.067 example: invariants inside unions

“Yeah, I probably should clean that after implementing those two next features”

Deprecation process is still desired

2.067 example: redundant postfix qualifiers

Mihails Strasuns
me@dicebot.lv

Better versioning?

SemVer

- Version pattern MAJOR.MINOR.PATCH
- PATCH : when you make backwards-compatible bug fixes
- MINOR : when you add functionality in a backwards-compatible manner
- MAJOR : when you make incompatible API changes

DMD

- Version pattern 2.XXX.Y
- Language has changed a lot since 2.000
- Each DMD release pretends to be minor but is in fact major
- No clear timelines for deprecations

Mihails Strasuns
me@dicebot.lv

Migration Instructions?

- Currently not present in changelog at all
- Most important thing to check upon release : helps to plan upgrade, reduces research investment
- Clearly differentiates intended changes from regressions
- Comes as the very first block in Sociomantic internal changelogs

Mihails Strasuns
me@dicebot.lv

dfix?

- https://github.com/Hackerpilot/dfix is sweet
- More promises of https://github.com/Hackerpilot/libdparse
- Hard to do reliable refactorings without full semantics analysis
- Example: symbol renaming. Requires to implement imports, scopes, fully qualified names, templates, mixins...
- “compiler as library” seems necessary. SDC?

Mihails Strasuns
me@dicebot.lv

Just one more slide...

Mihails Strasuns
me@dicebot.lv

We are hiring!

https://www.sociomantic.com/careers

- Berlin, Germany
- Both backend (D) and frontend (PHP/JavaScript) developers
- Ask us anything : careers@sociomantic.com

Mihails Strasuns
me@dicebot.lv

sociomantic
from dunnhumby