Domain Engineering with Concepts

Magne Haveraaen

Bergen Language Design Laboratory (BLDL)
Department of Informatics, University of Bergen, Norway

Workshop on Quality Software: A Festschrift for Bjarne Stroustrup
TAMU, College Station, 2012-04-28

Software Product Line Benefits

Software product line development versus normal development

• Productivity improvement: up to a factor of 10
• Quality improvement: up to a factor of 10
• Decreased cost: by as much as 60%
• Decreased labour needs: by as much as 87%
• Decreased time to market: by as much as 98%
• Ability to move into new markets: in months, not years

Each of the above is based on a documented product line effort
Linda Northrop, 2008
Software Product Line

Also called a *product family*

- A set of software-intensive systems
- Built for a particular market segment (domain)
- Created from a common set of core assets
  - Libraries, architectures, tests, tools, project planning

Core asset development:

*Domain engineering*

Application development: *Application engineering*

---

**Defining the Core Assets of a Domain**

Must fit the language of software

**Algorithms + Data Structures = Programs**

Niklaus Wirth 1976

- **A Data Structure** *abstracts to a type*
  - Values of a type can be compared for *equality*
- **An Algorithm** *abstracts to a function*
  - Input argument list
  - Result type
- Properties of a type are defined by **predicates** on expressions
  
  ```
  T a,b,c;
  assert ( (a+b)+c == a+(b+c) );
  ```
Questions to ask of a Domain

- What are the *types*
- What are the *functions*
- What are the *axioms*

What are the (C++) *concepts*

```c++
template<typename m>
concept monoid (binary<m> bin, nullary<m> unit) {
  axiom associative (m a, m b, m c) {
    assert bin(bin(a,b),c) == bin(a,bin(b,c));
  }
  axiom neutral (m a) {
    assert bin(a,unit()) == a;
    assert bin(unit(),a) == a;
  }
}
```

Data Structure Algebra

Isomorphisms

- The same information content for different declarations

```c++
struct {
  int a[100];
  int b[100];
} d1;
struct D {
  int a;
  int b;
};
D d2[100];
```

- Alternative data structures
  - Different access patterns
  - Different *abstractions*
The Heat Problem: Norway

The Heat Problem: Texas
The Heat Equation

\[ \frac{\partial}{\partial t} u = \alpha \left( \nabla \cdot (\nabla u) \right) + f \]

Variables, in space and time
- \( u \) – temperature, scalar field
- \( \alpha \) – thermal diffusivity, scalar field
- \( f \) – heat source, scalar field

Derivatives
- \( \partial/\partial t \) – partial derivative in time
- \( \nabla \) – gradient, scalar field to vector field
- \( \nabla \cdot \) – divergence, vector field to scalar field

Operations
- \( * \) – scalar field multiplication
- \( + \) – scalar field addition

Concepts for Arithmetic Operations

```cpp
template< typename r >
concept unit_ring(binary<r> plus, unary<r> minus, binary<r> mult) {
   axiom abelian_group(r a, r b, r c) {
      assert plus(plus(a,b),c) == plus(a,plus(b,c));
      assert plus(a,b) == plus(b,a);
      assert plus(a,r(0)) == a;
      assert plus(a, minus(a)) == r(0);
   }
   axiom monoid(r a, r b, r c) {
      assert mult(mult(a,b),c) == mult(a,mult(b,c));
      assert mult(a,r(1)) == a;
      assert mult(r(1),a) == a;
   }
   axiom distributive(r a, r b, r c) {
      assert mult(a,plus(b,c)) == plus(mult(a,b),mult(a,c));
      assert mult(plus(a,b),c) == plus(mult(a,c),mult(b,c));
   }
}
```
Engineering the PDE domain

- Data field df<r>: a value of type r at every point in space-time
  - Scalar field sf<real>, ring with pointwise +,-,* and ∂/∂t, ∂/∂x, ..
- Matrix matrix<r> with +,-,mm from any ring r
- Matrix field with ∇·, ∇
  - df<matrix<real>>
  - matrix<sf<real>>

Choosing matrix field format: consider the derivation operations
- Derivatives require access to neighbouring data
- Scalar field has partial derivatives ∂/∂t, ∂/∂x, ..
  - The derivations can be defined from partial derivatives

Dot Product Problem

```cpp
template<typename r> r dot(vector<r> a, vector<r> b) {
    return ∑ a[i] * b[i];
}
template<typename r> vector<r> new_coordinate( matrix<r> m, vector<r> v) {
    return mm(m,v);
}

template<typename r> concept dot_properties () {
    axiom coordinate_system_invariance(matrix<r> m, vector<r> u, vector<r> v) {
        assert dot(u,v) == dot(new_coordinate(m,u),new_coordinate(m,v));
    }
    // ...
}
```
- Dot algorithm is wrong? Take coordinate system into account
- Typing is wrong? Vector and covector
- Change of coordinate algorithm is wrong? Covectors are different
Seismic Waves

\[ \rho \frac{\partial^2 u}{\partial t^2} = \nabla \cdot \sigma + f, \]

\[ \sigma = \Lambda \circ e, \]

\[ e = L(u, g) \]

Elastic wave equation

Variables
- \( \rho \) – density, scalar field
- \( u \) – displacement, vector field
- \( \sigma \) – stress, matrix field
- \( f \) – external force, vector field
- \( \Lambda \) – stiffness, tensor field
- \( e \) – strain, matrix field
- \( g \) – metric, matrix field

Derivatives
- \( \frac{\partial}{\partial t} \) – partial derivative in time
- \( \nabla \cdot \) – divergence, matrix field to vector field
- \( L \) – Lie derivative, matrix field to matrix field

Operations
- \( \circ \) – tensor application, returns matrix field
- \( + \) – vector field addition

Conclusions

- Domain engineering
  - Defines the core assets of a software domain
  - Essential for software product lines
  - Precedes application engineering

- C++ style concepts for core asset development
  - Libraries
    - Declares types, declares functions, defines axioms
    - Drives towards a comprehensive API
  - Architectural considerations
  - Testing
    - Axioms as test oracles
  - Tools: refactoring and optimisation
    - Equational axioms as refactoring rules