VDM Animation for a Wider Range of Stakeholders

Tomohiro Oda	**Software Research Associates, Inc.**
Yasuhiro Yamamoto	**University of Tokyo**
Kumiyo Nakakoji	**Kyoto University**
Keijiro Araki	**Kyushu University**
Peter Gorm Larsen	**Aarhus University**

This work is supported by Grant-in-Aid for Scientific Research (S) 24220001

expressiveness

Formal specification

A formal specification of a system explains the system's functionality:

- what concepts are involved
- what should be achieved

- It's **hard to express things like "user's feelings"** in formal specs, which may change value of the system, i.e.
  - crispy transition of modes
  - instant and responsive update of info

Expressiveness of Formal Specs

- Animation can let people to experience the specified system
  - implications to UI
  - implications to client modules
  - implications to programs that is out of the scope of the formal spec.
  - implications to the value of the system

Animation can be understood by people who have no formal methods background.

Topics in this talk

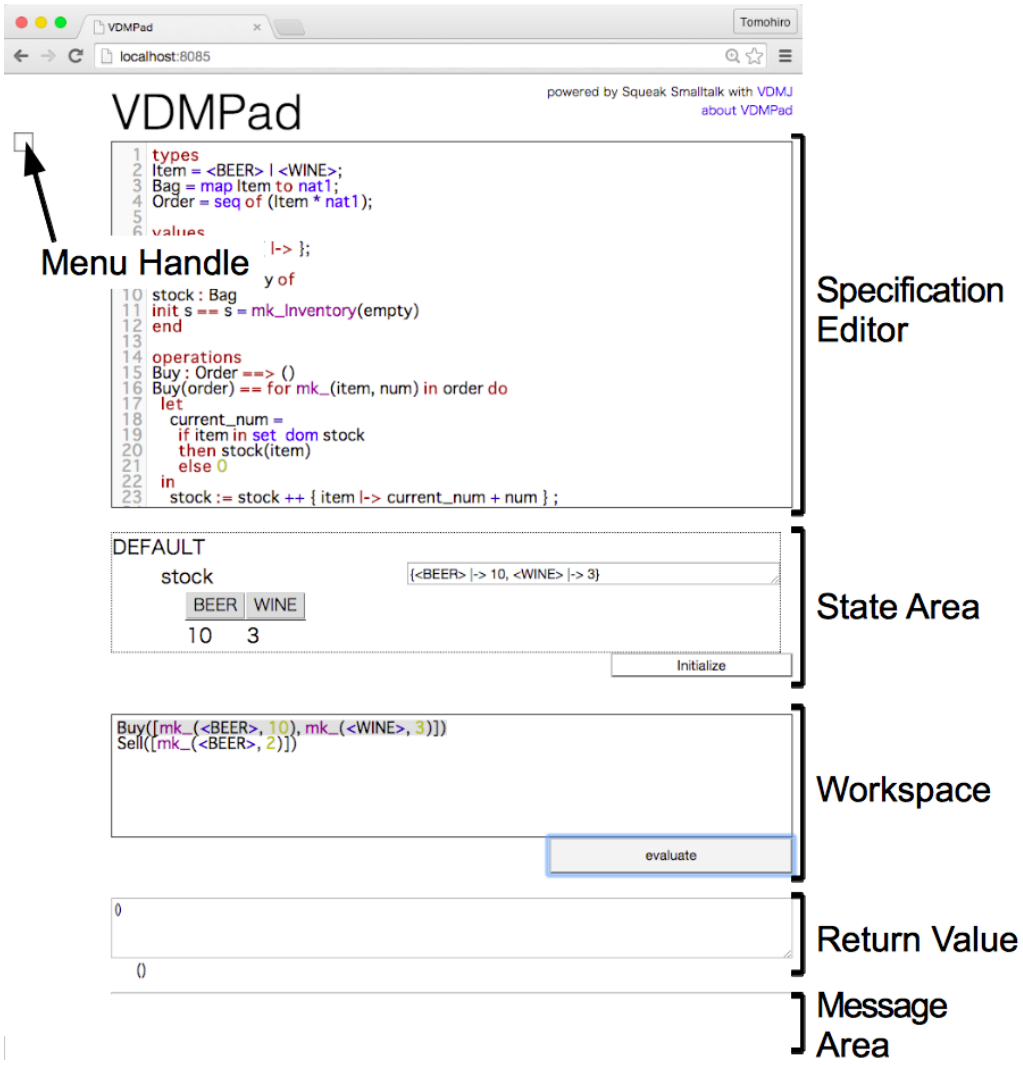
to extend users of formal specs

from readers to experiental group

- **formal engineers**
  - overture tool, VDMTools, **VDMPad**
- **programmers and testers**
  - **Webly Walk-Through, pyVDMC**
- **UI designers**
  - **Lively Walk-Through**
- **non-engineering stakeholders**
  - **Cloudly Walk-Through**

basis

VDMPad
Animation to explore spec space



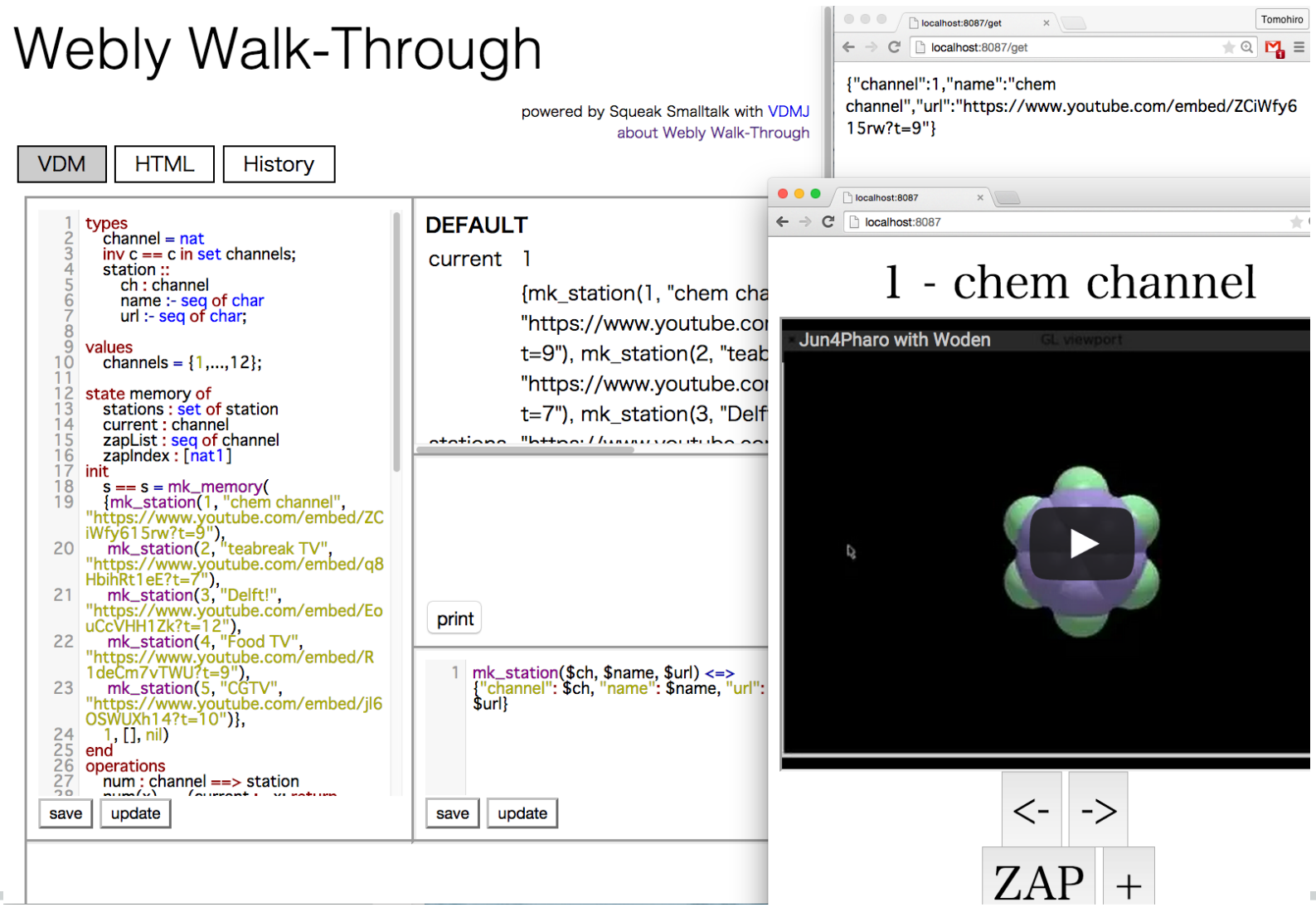
outreach

Webly Walk-Through

Web API prototype for web programmers

- specify Web API in VDM-SL
- serve the Web API
  - http://localhost:8087/<module>/<operation>?arg...
  - VDM-SL ⇔ JSON conversion
- build a prototype of web client
  - html/css/javascript
- evaluate and discuss the Web API
  - history of Web API calls

Webly Walk-Through
demo



pyVDMC
example fibonacci numbers

```
from pyVDMC import *
@vdm_module('n1', 'n2')
class fibonacci:
    """
    state State of
        n1 : nat
        n2 : nat
        init s == s = mk_State(0, 1)
    end
    operations
        next : () ==> nat
        next() == (dcl n : nat := n1 + n2; n1 := n2;
n2 := n; return n)
        post RESULT = n1~ + n2~ and n1 = n2~ and n2 =
RESULT;
        prev : () ==> nat
        prev() == (dcl n : nat := n2 - n1; n2 := n1;
n1 := n; return n2)
        post n1 + n2 = n2~ and n2 = n1~ and n2 =
RESULT;
    """
```

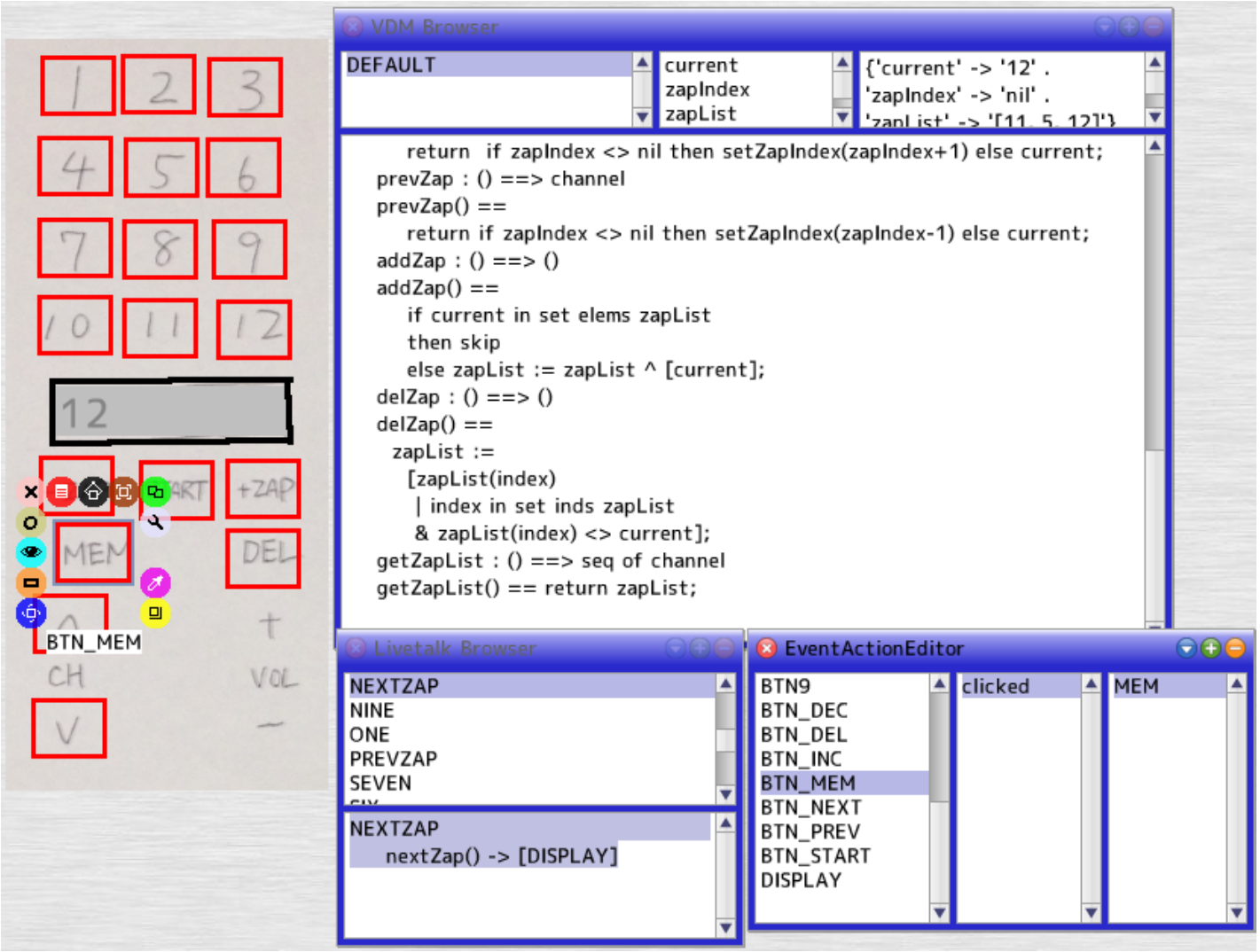
the class comment as a VDM-SL spec

```
    def __init__(self):
        self.n1 = 0
        self.n2 = 1
    @vdm_method
    def next(self):
        pass
    @vdm_test
    def prev(self):
        n = self.n2 - self.n1
        self.n2 = self.n1
        self.n1 = n
        return self.n2
```

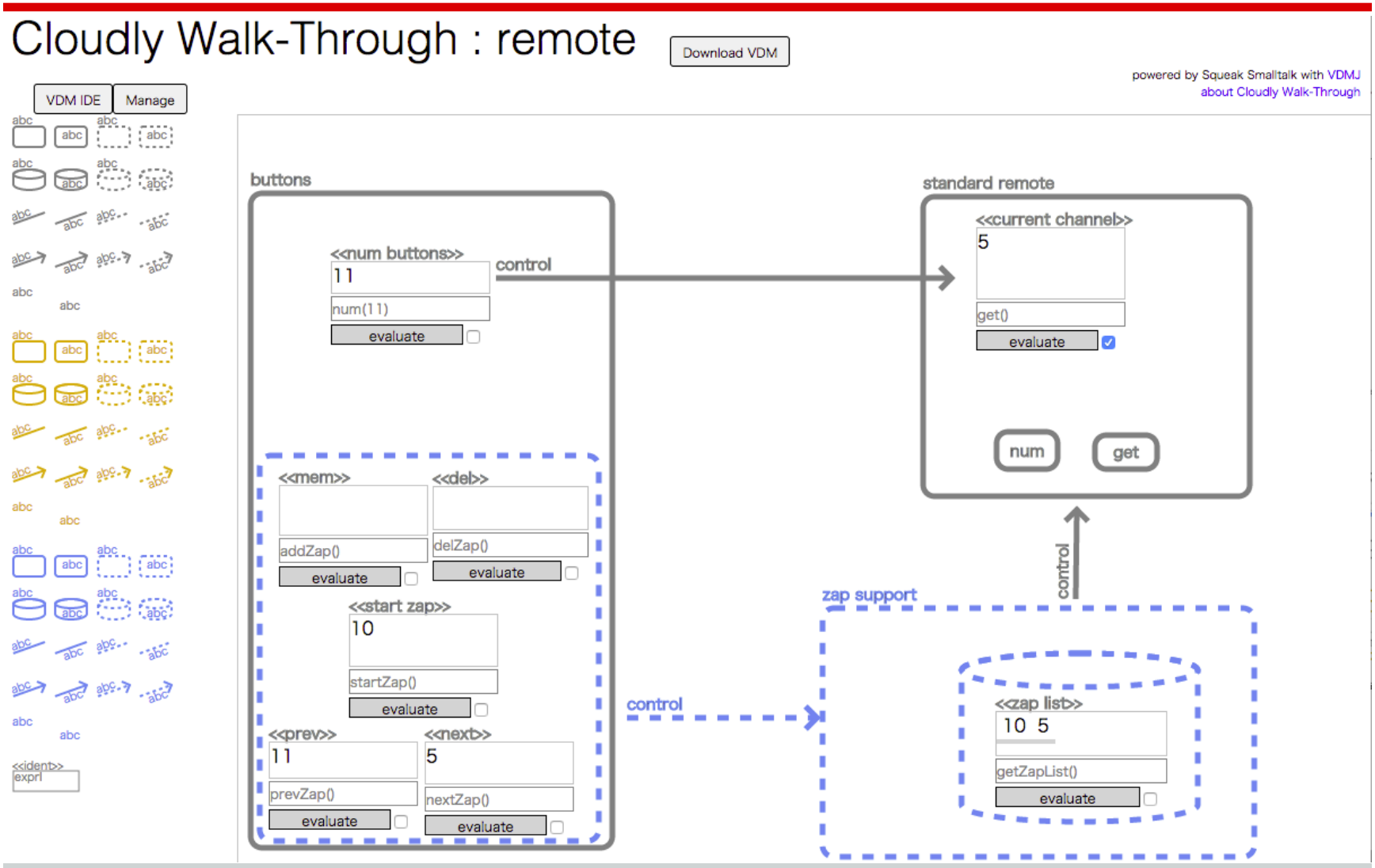
the spec is used as an implementation.

the spec is used as a test oracle.

Lively Walk-Through
prototyping with UI designers

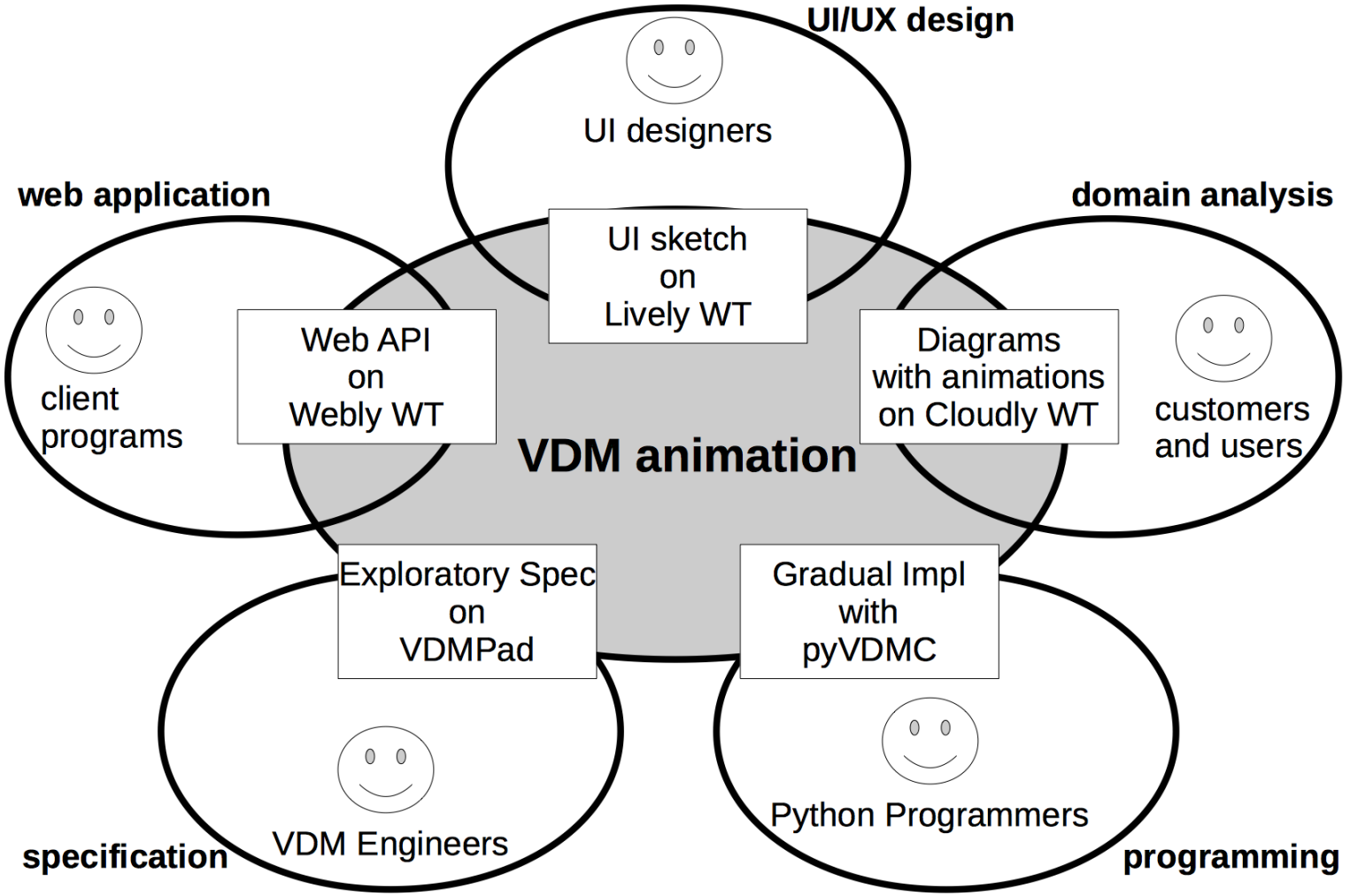


Cloudly Walk-Through
general diagrams with mini-VDMPads



conclusion

Summary


UI/UX design
UI designers
web application
domain analysis
UI sketch
on
Lively WT
Web API
on
Webly WT
Diagrams
with animations
on Cloudly WT
client
programs
customers
and users
VDM animation
Exploratory Spec
on
VDMPad
Gradual Impl
with
pyVDMC
VDM Engineers
Python Programmers
specification
programming

Overture in 1, 5, 10 years

in 1 year

- supports gradual implementation
  - all ops in VDM -> some ops in PL -> ...
    -> most ops in PL -> all ops in PL

in 5 years

- supports more impl languages

in 10 years

- Dynabook of VDM
  - live spec
  - VDM spec and code on one dynamic media
  - VDM spec for everyone including end users