Software Maintenance Competences

Jurgen J. Vinju

NWO-I Centrum Wiskunde & Informatica
TU Eindhoven
Swat.engineering

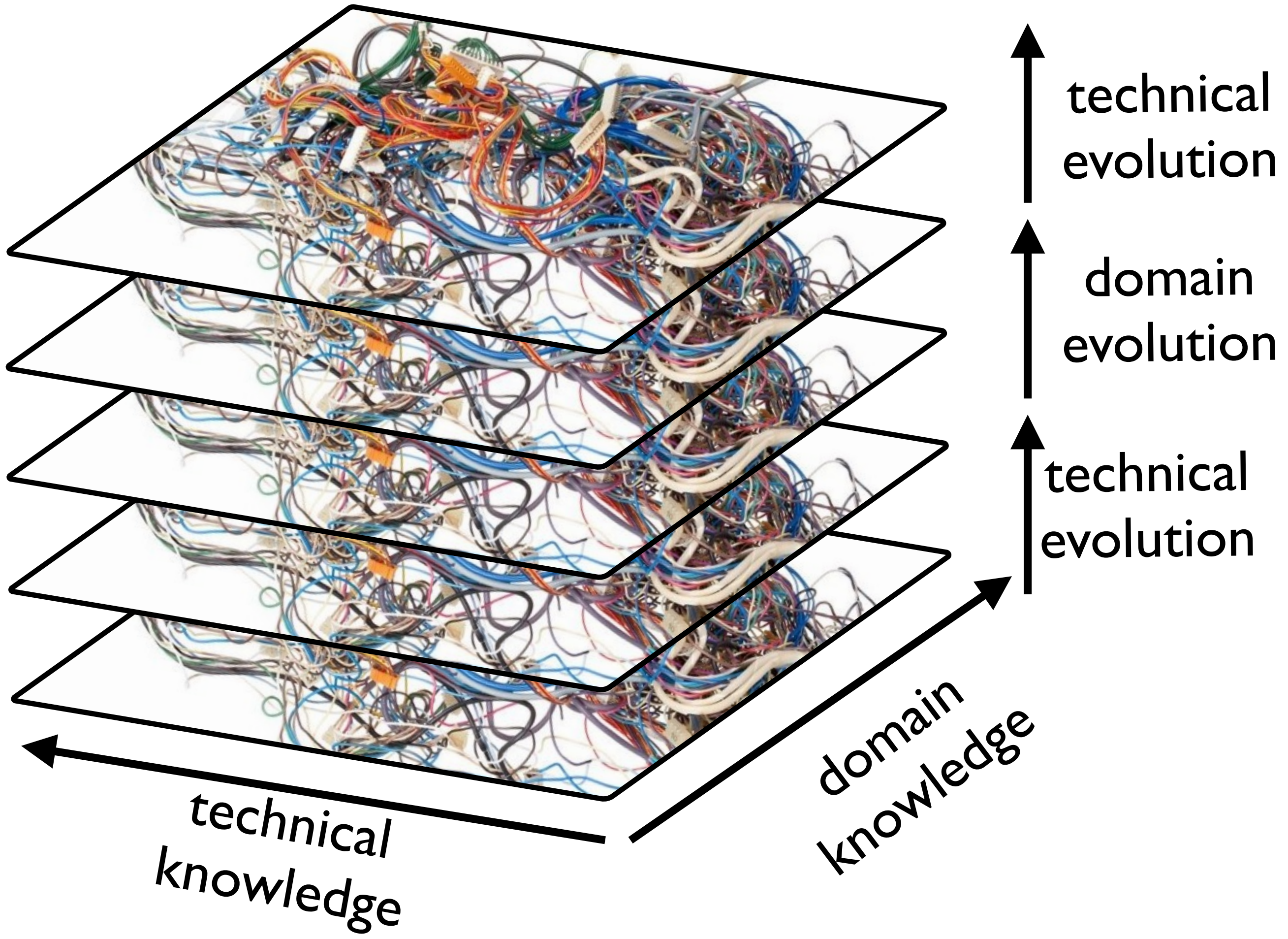
swat.engineering
control your software

Source Code Complexity

- Source Complexity results from **tangling** *four code dimensions*

  Domain Complexity **x** Domain Evolution **x**

  Technical Complexity **x** Technical Evolution



software starts <u>simple</u> <u>and</u> <u>flexible</u> and then gradually grows into <u>a big knot</u> which is hard to maintain

Software Maintenance

“changing source code after the initial release”

Engineering

- **Design** stage
  - System architecture is *designed*
  - Reversible design decisions
  - Short term successes
  - Testing is easy
- **Growth** stage
  - Incremental additions and corrections *grow*
  - Misconceptions and haste lead to design erosion
  - Co-evolution: changes become scattered
  - “Accidental” code, when it works -> commit
  - Testing becomes cumbersome
- **Stagnation** stage
  - Changes break the system; increasing focus on analysis
  - Working on bugs rather than features
  - “How did this ever work”?
  - Critical reading pushes out (creative) writing

Business

- Early **Benefits** of software ownership
  - Tactical advantage: fast time-to-market
  - Short horizons
  - Incremental costs
- Growing **Cost** of software ownership
  - Lower margins over time
  - Increasing maintenance costs
  - Cost of replacement out-weighs the ROI
- Inevitable **Risks** of software ownership
  - Software becomes cause of stagnation
  - Employee turn-over rate too high
  - Cost of maintenance outweighs total value

Source Code Maintenance: **Necessary** but Challenging

certify GDPR compliance

add WWW interface

add live user feedback

fix performance bottleneck for peek user loads

upgrade to Windows 10 from Windows 95

scale to {giga,tera,peta}byte/s throughput

add live user feedback

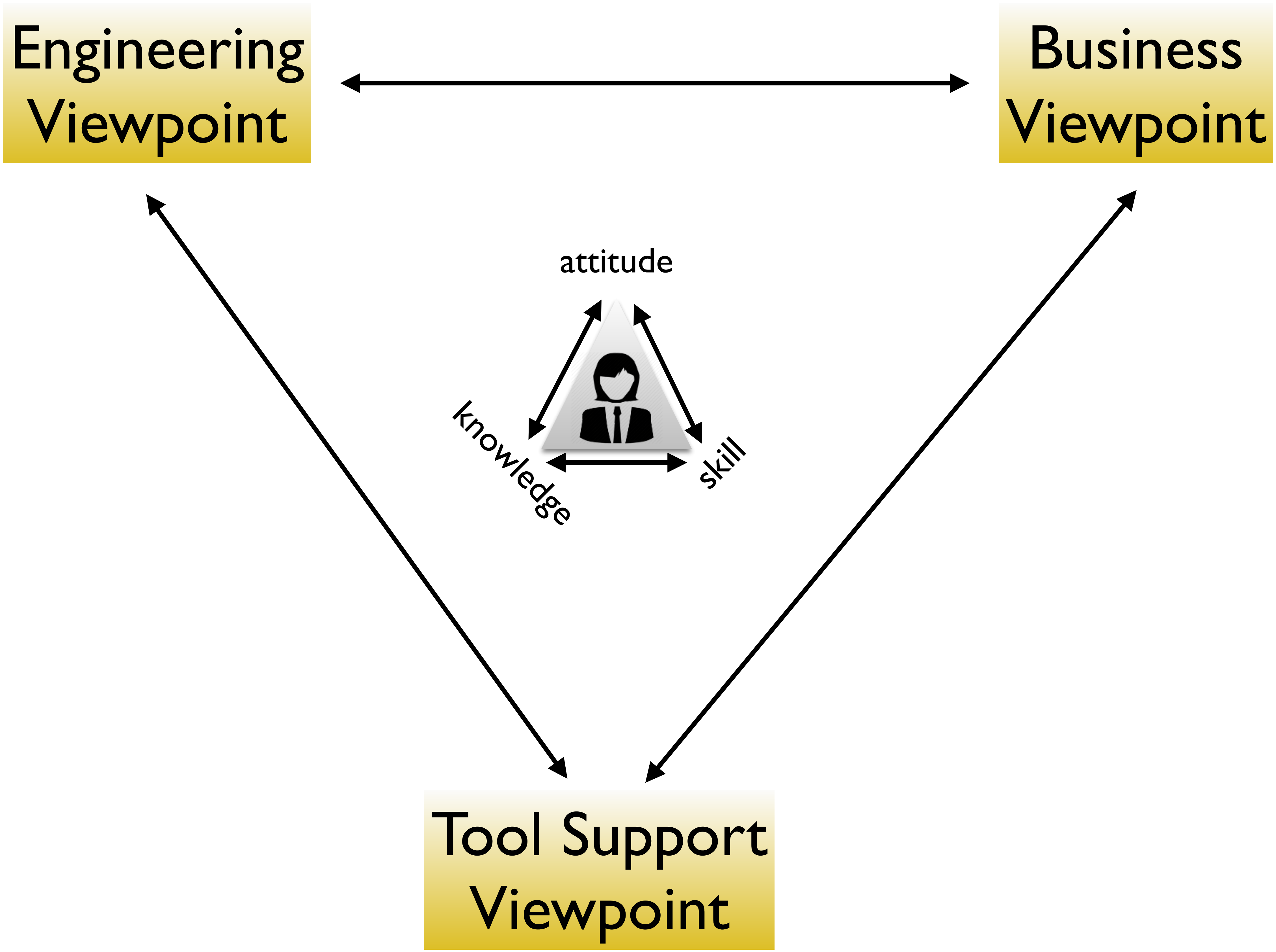
integrate 3D simulation

merge this acquired software "stack" into our own, with backward compatibility

port to Linux

switch to ARM

Software Maintenance Competences



Engineering Viewpoint

Business Viewpoint

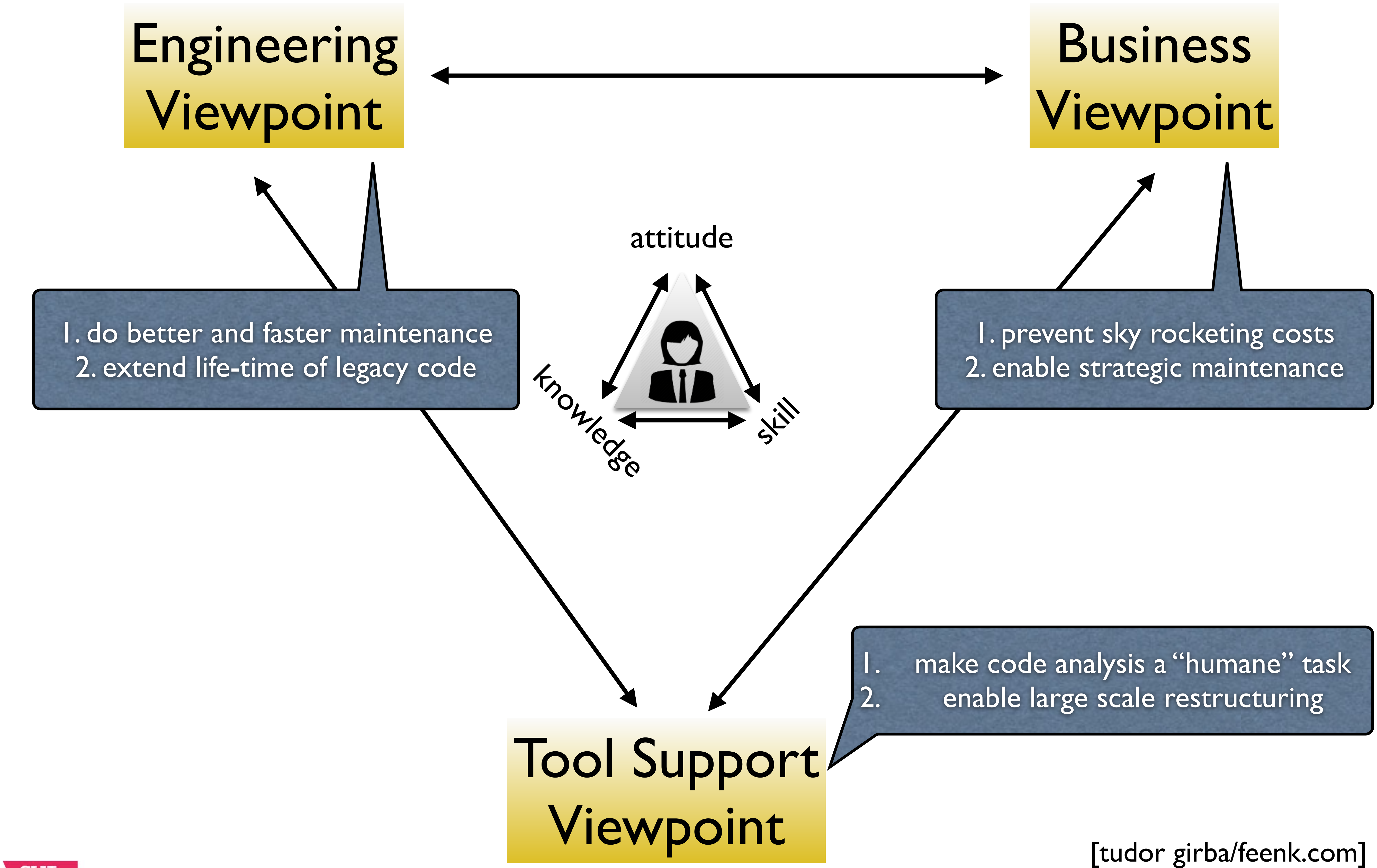
attitude

knowledge

skill

Tool Support Viewpoint

Goals of Software Maintenance



Engineering Viewpoint

Business Viewpoint

1. do better and faster maintenance
2. extend life-time of legacy code

1. prevent sky rocketing costs
2. enable strategic maintenance

1. make code analysis a “humane” task
2. enable large scale restructuring

Tool Support Viewpoint

[tudor girba/feenk.com]

Business Attitudes

- I want to *motivate* good software maintenance by *evaluating* maintenance:
  - New KPI’s measure costs and benefits of maintenance
- I want to invest in *Preventive Maintenance*
  - Because maintainable software is flexible software
- I know that high-quality (=maintainable) software does not come for free
  - Engineers have to invest in software quality
  - Before they can offer agility
- I want to explain my requirements precisely

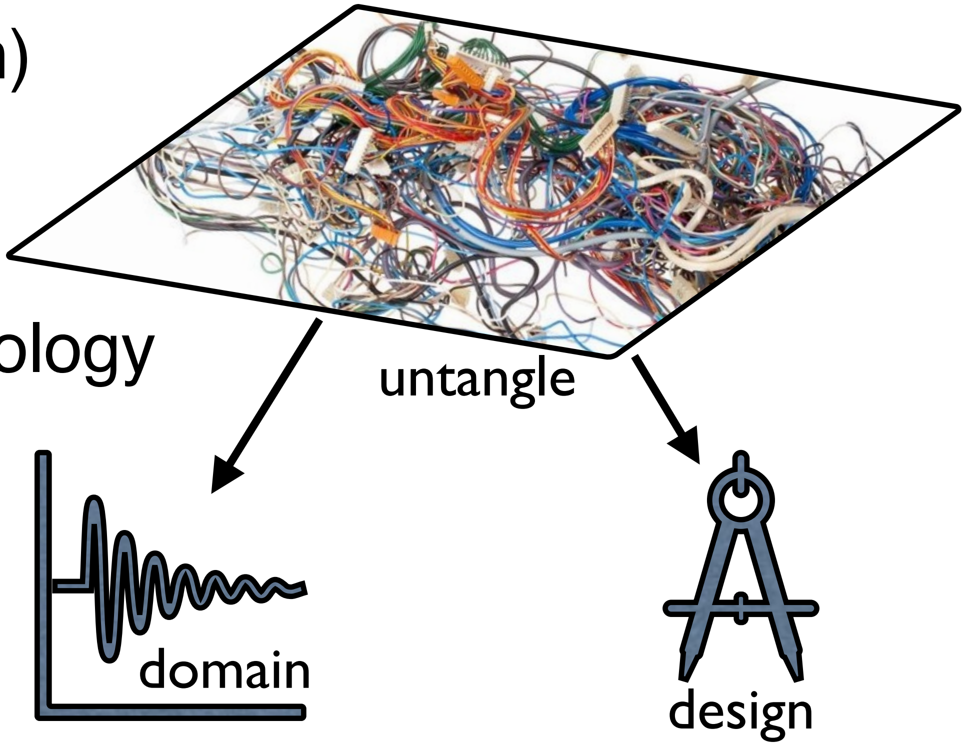
Use Better Key Performance Indicators

Plan for maintenance: maintenance now = future ROI

Appreciate Software Quality

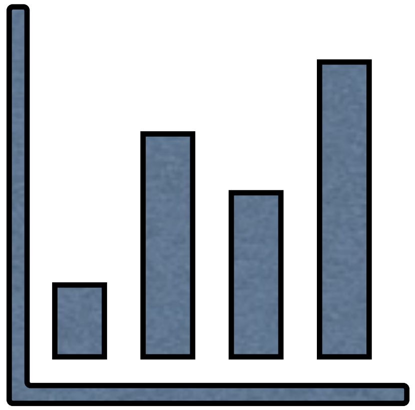
Grow Business Knowledge

- “I know my business”: model business complexity independently, e.g:
  - *image algorithms* **vs** {SIMD,GPU,FPGA} instructions
  - *documented protocols* **vs** inter-process communication library calls
  - *telemetry and control state machines* **vs** code design patterns
- Model Driven Engineering benefits start with **domain knowledge**
  - **benefits:** early feedback (verification, simulation)
  - **benefit**: code generation
  - **benefit**: *independent evolution:* domain & technology



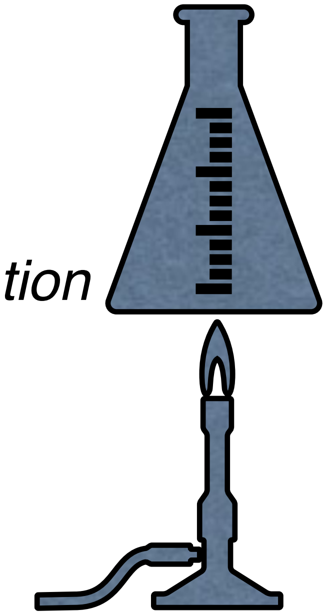
Learn New Business Skills

- I can “measure maintenance quality (contra)-indicators”
  - **growth** of volume & complexity
  - **issues**: registration and resolution of maintenance tasks, …
  - **versions**: locality of change, commit coherence, …
  - **tests**: coverage (branches) and quality (mutant score)



metrics that make sense

- I can “author executable domain models”
  - lightweight formalization of **requirements**
  - (interactively) simulate, explore, test, verify software products *before implementation*
  - *evolve* domain models to address new business opportunities
  - *predict* impact of business changes on technology stack



MDE experimentation
= understanding

Need business tools

- I need tailor-made modeling, simulation, validation tools
- I need tools to measuring maintenance quality and productivity

- Off-the-shelf is _not_ the answer
  - domain knowledge is contextual
  - maintenance quality is contextual

- Rascal is a metaprogramming language for tailor-made software analysis and manipulation tools.

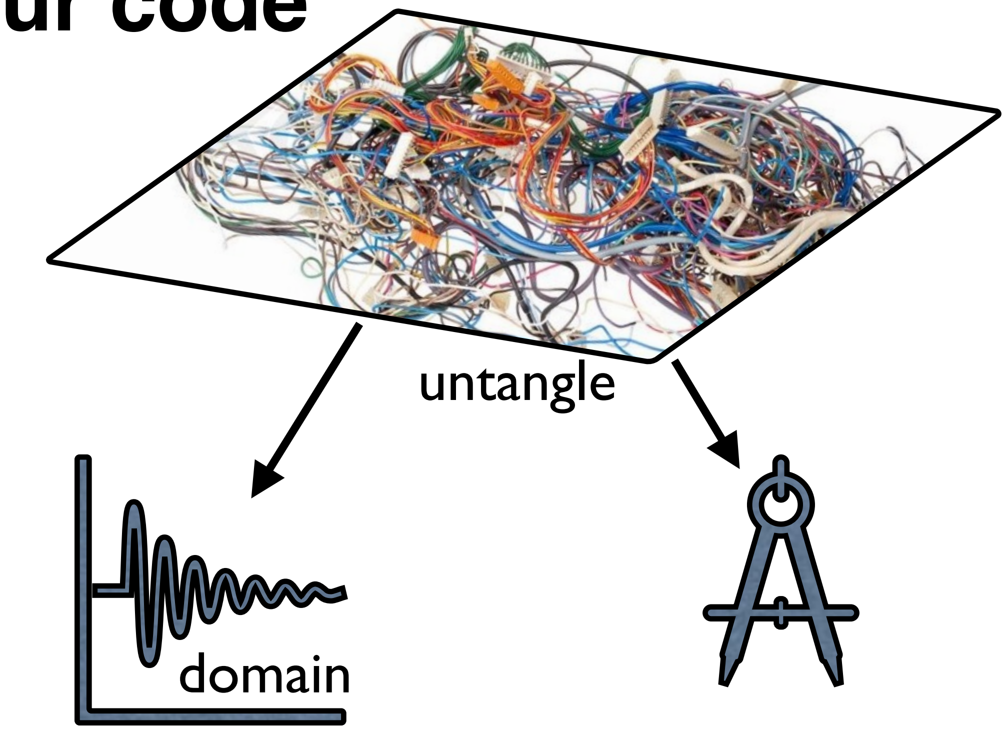
Engineering attitudes

Maintenance
is my job
and I like it

- I think analyzing software puzzles is *interesting*
- I *own* the maintenance of this legacy code
- I want to *automate* analyses and transformations

Increase engineering knowledge

- I understand the programming language and the OS
- I understand **code-as-data:** AST, CG, PDG, SDG
- I know **our domain** independent from **our code**



Learn new engineering skills

going meta

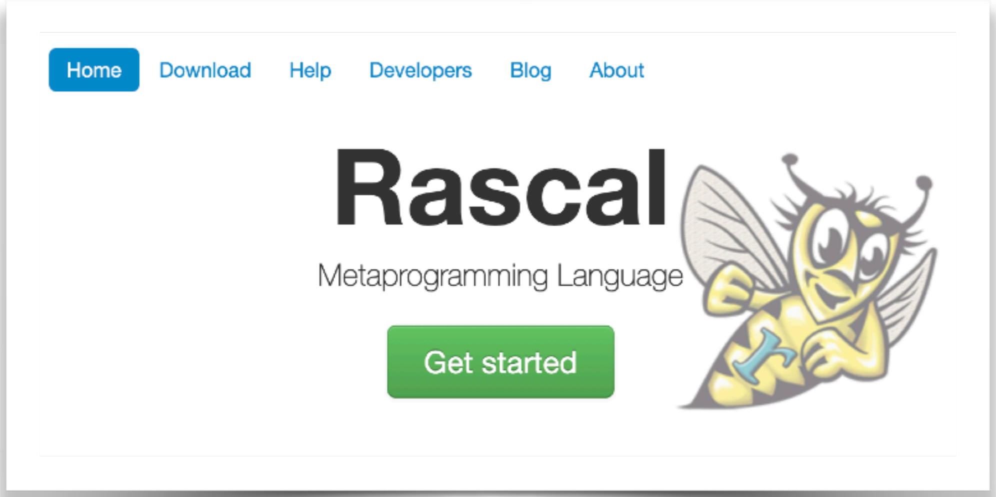
- “I can automate analysis tasks”:
    - Write tailor-made source code analysis queries (code-to-model)
    - Map models to existing analysis platforms (SMT, PDE,
- “I can automate refactoring tasks”
    - Writing source-to-source transformations (code-to-code)
- “I can automated software construction tasks”
    - I can separate domain knowledge from code design knowledge
    - I can write source code generators (model-to-code)

Rascal is for easily writing meta programs

Need new engineering tools

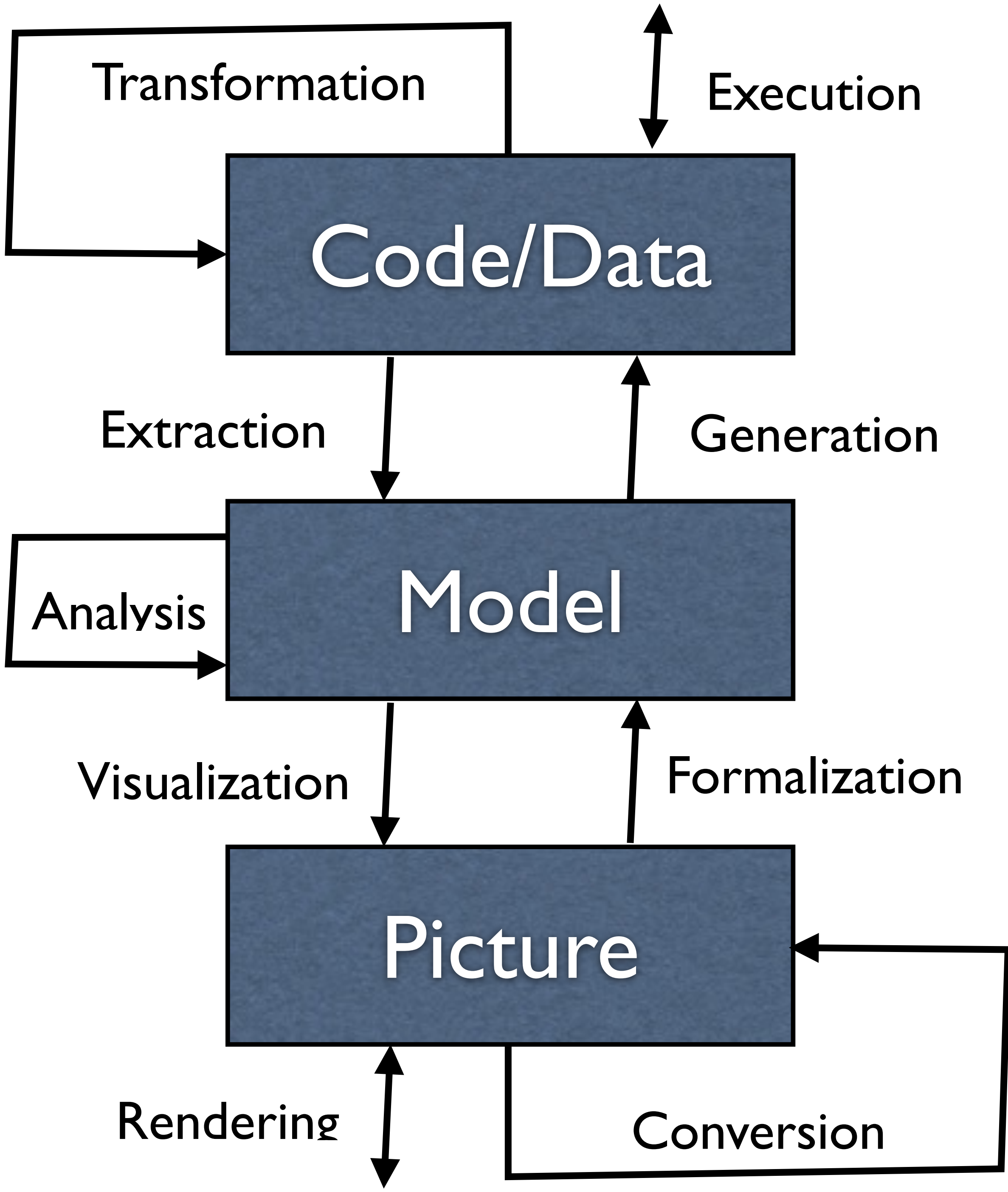
- Need API for handling *programming language complexity*
  - code-as-data: syntactic and semantic intermediate models
  - query/expression: pattern matching, relational queries, templates
- Need ability to encode domain knowledge and design knowledge

- A “one-stop-shop” meta-programming language: Rascal



Rascal MPL

- **Comprehensive** metaprogramming language
    - For creating tailor-made modeling and analysis tools
    - For creating analysis, transformation and generation tools
    - Same functionality in ±10% of lines of code
- (Inter)National **Community**:
    - **Research**: incorporates results from 1982 to 2021
    - **Education**: UvA, TUE, RUG, OU, ECU, Bergen
    - **Business**: swat. engineering control your software
- **Support**
    - *Languages:* Java, C++, C#, PHP, JS, JVM bytecode, …
    - *Analysis:* SMT, Relational Algebra, State Machines
    - *UI:* Eclipse, **VScode** (LSP), Commandline Interface
- **Track record:** Philips Healthcare, ING, OCÉ, NFI, SIDN, Stokhos, EU Typhon, EU CROSSMINER, EU OSSMETER, …



“E-A-SY” Rascal Example

Extract, Analyse, and SYnthesize

1. parse input code

grammars

```
module Syntax

extend lang::std::Layout;
extend lang::std::Id;

start syntax Machine = machine: State+ states;
syntax State = state: "state" Id name Trans* out;
syntax Trans = trans: Id event ":" Id to;
```

2. create “model”, transitive closure, and query

matching & query

```
module Analyze

import Syntax;

set[Id] unreachable(Machine m) {
  rel[Id,Id] r = { <q1,q2> | (State) `state <Id q1> <Trans* ts>` <- m.states,
                             (Trans) `<Id _>: <Id q2>` <- ts
                 }+;

  qs = [ q.name | /State q := m ];
  return { q | q <- qs, q notin r[qs[0]] };
}
```

3. generate a visual representation

overviews

```
module Visualize

import Syntax;
import DagreD3;

void visualize(Machine m) {
  edges = { edge("<q1>", "<q2>") | (State)`state <Id q1> <Trans* ts>` <- m.states,
            (Trans)`<Id _>: <Id q2>` <- ts };

  nodes = { node(“<q.name>") | /State q := m };

  showGraph(nodes, edges);
}
```

4. generate implementation

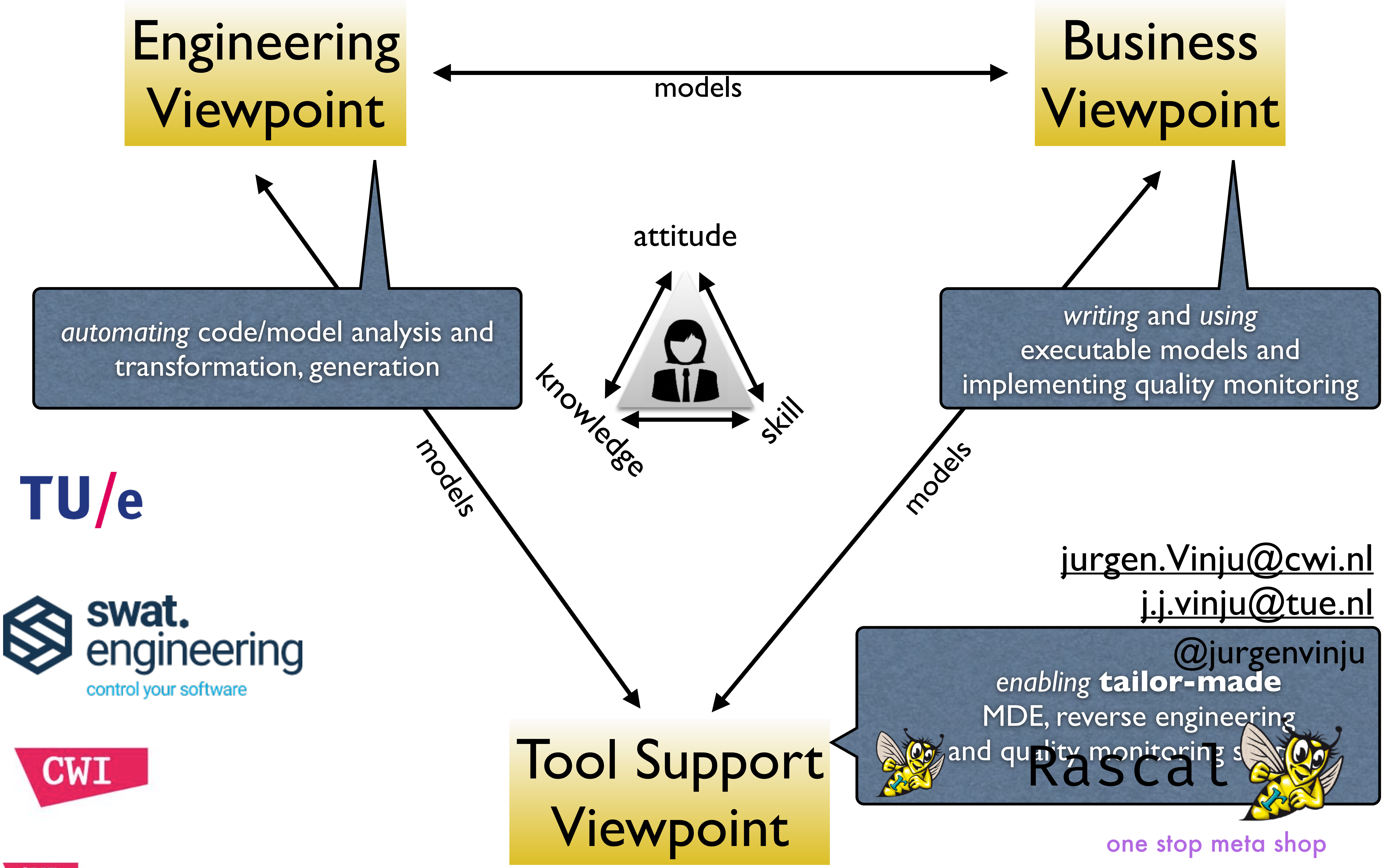
templates

```
module Compile

import Syntax;

str compile(Machine m) =
  "while (true) {
  '  event = input.next();
  '  switch (current) {
  '    <for (q <- m.states) {>
  '    case \"<q.name>\":
  '      <for (t <- q.out) {>
  '      if (event.equals(\"<t.event>\"))
  '        current = \"<t.to>\";
  '      <}>
  '      break;
  '    <}>
  '  }
  '}";
```

Conclusion



Engineering Viewpoint

models

Business Viewpoint

automating code/model analysis and transformation, generation

attitude

knowledge

skill

writing and *using* executable models and implementing quality monitoring

models

models

Tool Support Viewpoint

enabling **tailor-made** MDE, reverse engineering and quality monitoring support

Rascal

one stop meta shop

jurgen.Vinju@cwi.nl
j.j.vinju@tue.nl
@jurgenvinju

TU/e

swat. engineering
control your software

CWI