reqT

- an open, scalable systems modelling laboratory for research & teaching (& practice)

Download latest reqT.jar at: http://reqT.org/

Run in terminal: java -jar reqT.jar

Preview of discussion items after/during talk and demo

- What are your requirements on a requirements engineering open source software (OSS) tool?
  - For research?
  - For teaching ?
- How to best involve the academic RE community in taking a collective ownership of and contribution to a wide-spread OSS RE tool?
  - How should the reqT OSS project be governed to make it easy for you to contribute?

RE on planet Earth in 5-10 years ... ?

Some hypotheses

More <u>continuous</u> build, integration & deployment

Faster release cycles & Faster innovation

More SW eco systems, distributed developer communities, open source

=>

More **decentralization**

and fewer centrally controlled 'Master Plans'

More `coders`

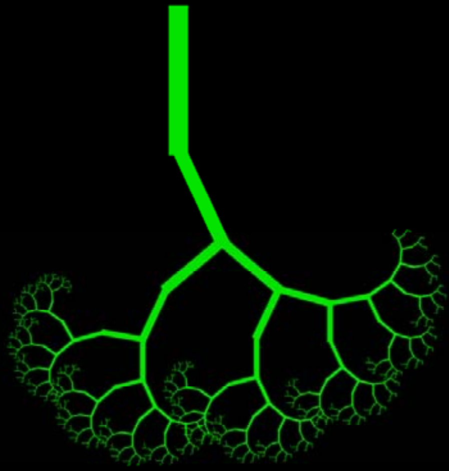
will do the bulk of requirements engineering

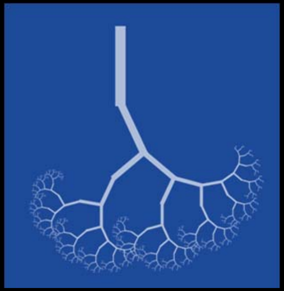
reqT inception and high-level reqts

We want...

- a free, open source tool for RE **teaching**
- a laboratory to try out, integrate and demonstrate new **research results**
- a **scalable** tool that supports models ranging from small "shopping lists" to huge reqts collections
- a **flexible** tool:
  - mix natural language with computable structures
  - "methodology-agnostic" meta-model
- a **coder-friendly** tool that integrate with coder's tools (my favorite editor, version control, scripting, api etc.)

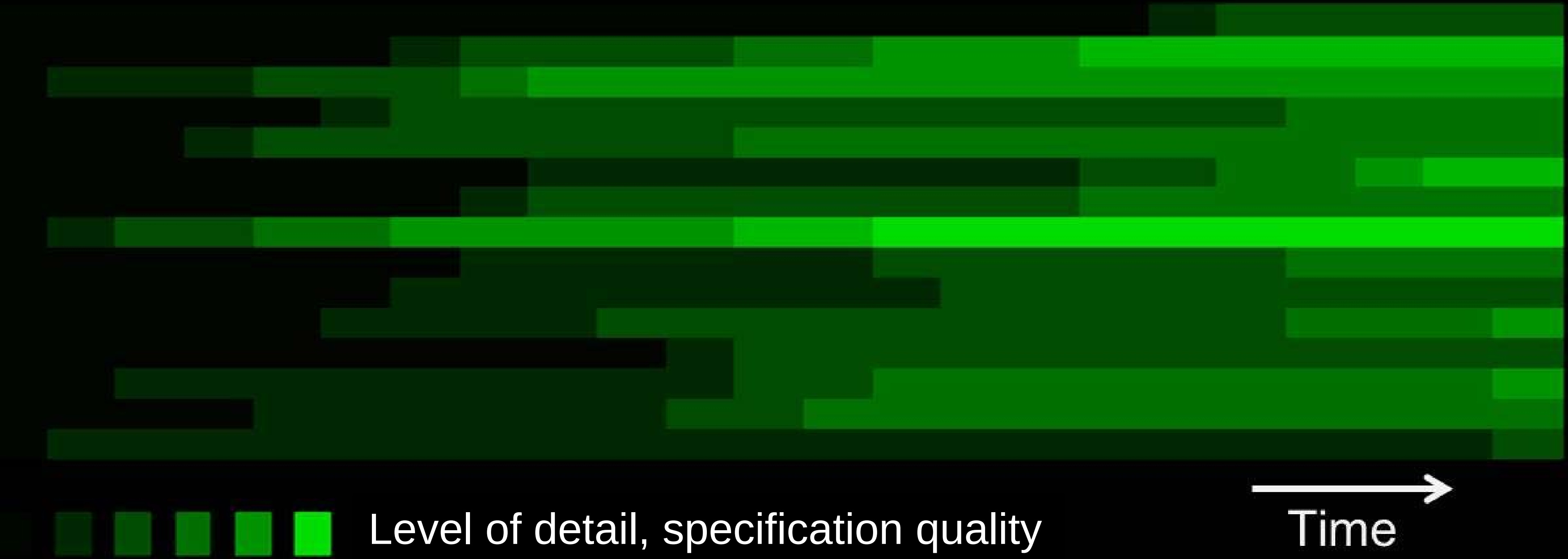
Scenario("Coders work in ecosystems with **req+code+test** in distributed git repos. Each stakeholder has its own, local understanding of ideas, roadmaps and acceptance criteria. Code is forked, pushed, pulled and merged **continuously** in the ecosystem. The implicit 'ice berg' of **mixed, semi-formal models** is neither complete nor fully consistent. We manage local trees of req+code+test and mine sets of mixed, semi-formal models with big data technology on both dev repos and UX data. The **community culture** and repo governance determine success rather than process control.")



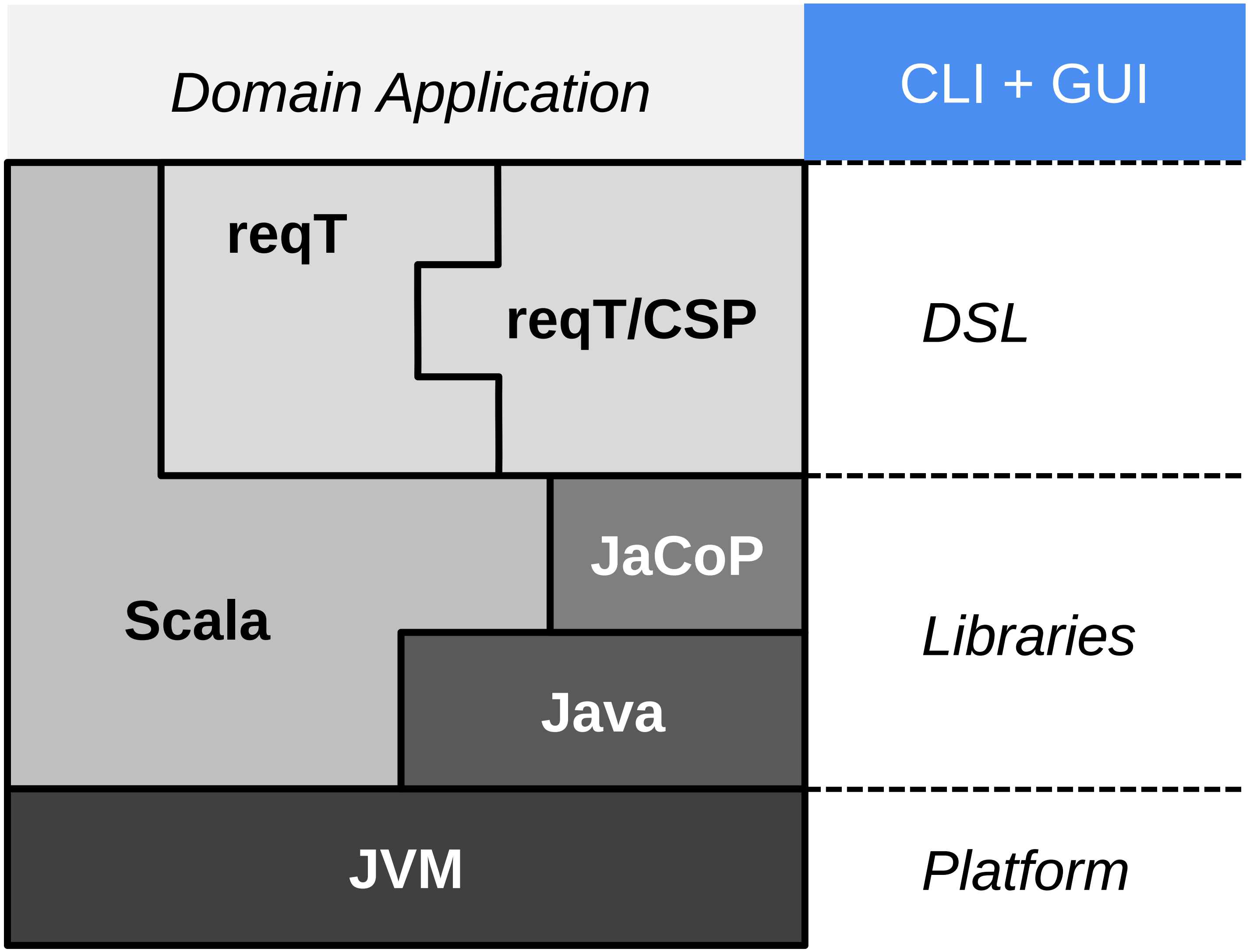


myModel ++ yourModel

Evolving heterogeneous mix of levels of detail & quality in continuous requirements engineering



reqT architecture



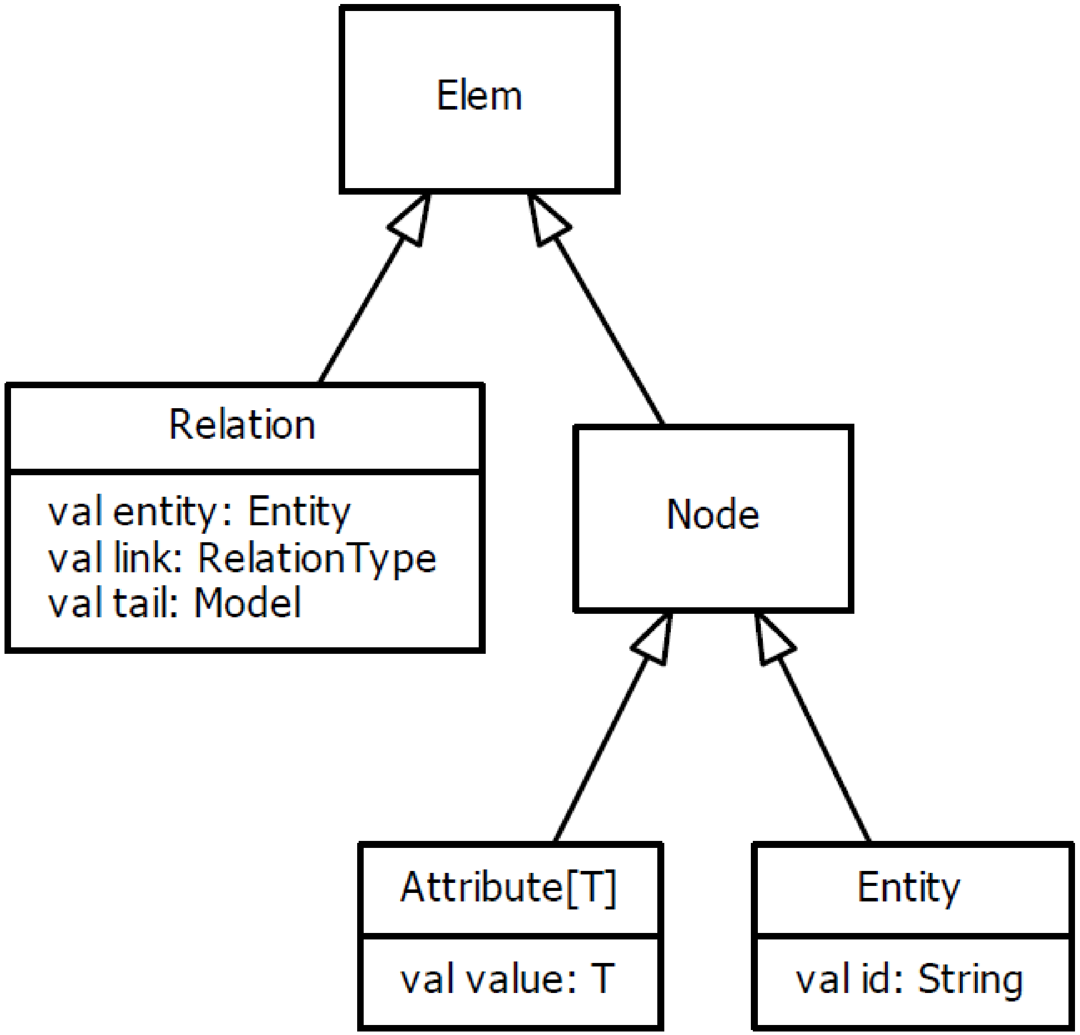
Open Source Software (OSS) in **reqT**

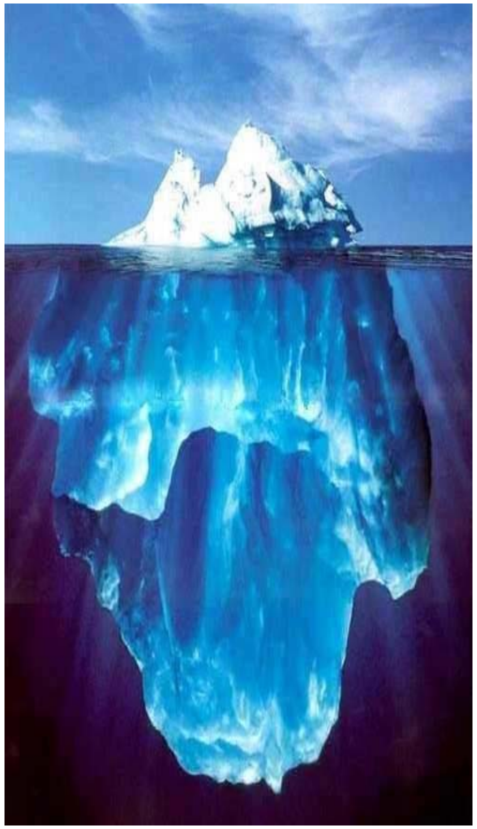
OSS	**Licence**
• reqT	• BSD-2-caluse
• Scala libs & compiler	• similar to BSD-2-caluse
• JaCoP	• GNU GPL v2 & v3
• jLine	• similar to BSD-2-caluse
• RSyntaxTextArea	• similar to BSD-2-caluse
• jFlex	• BSD-2-caluse
• GraphViz	• Eclipse Public License

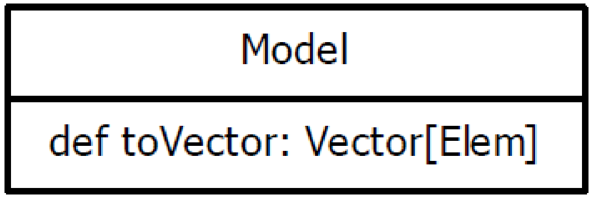
reqT is metaprogrammed

- The metamodel of reqT is expressed in reqT
- reqT has a Scala module that takes the reqT metamodel as input and generates a Scala module that implements the reqT DSL
- The bootstrap reqT DSL includes only a few elements such as Meta and superOf

The embedded DSL provides a recursive, tree-like data structure







Some essential requirements entitites and attributes

Req generic, abstract, undecided	**Gist** short one-liner
Feature decision item with status	**Spec** txt descr
Stakeholder	**Why**
Goal	**Example**
UserStory, TestCase, Issue	**Prio**
Quality	**Cost**
Function	**Benefit**
Data	**Status**
...	...

Some essential requirements **relations**

- Requirements entities have relations that turn the reqts into a **graph**

```
Model(
  Req("a") requires Req("b")
)
```

- `has`
- `requires`
- `excludes`
- `helps`
- `hurts`
- …

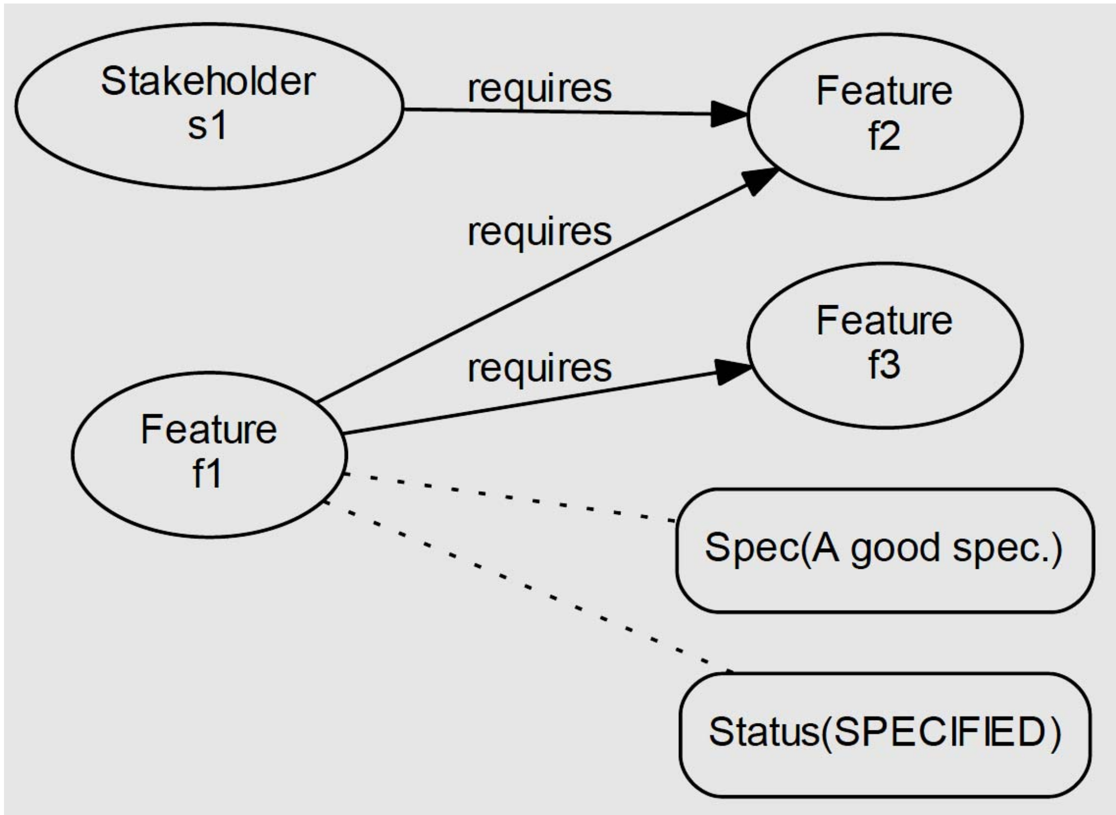
Requirements as graph structures

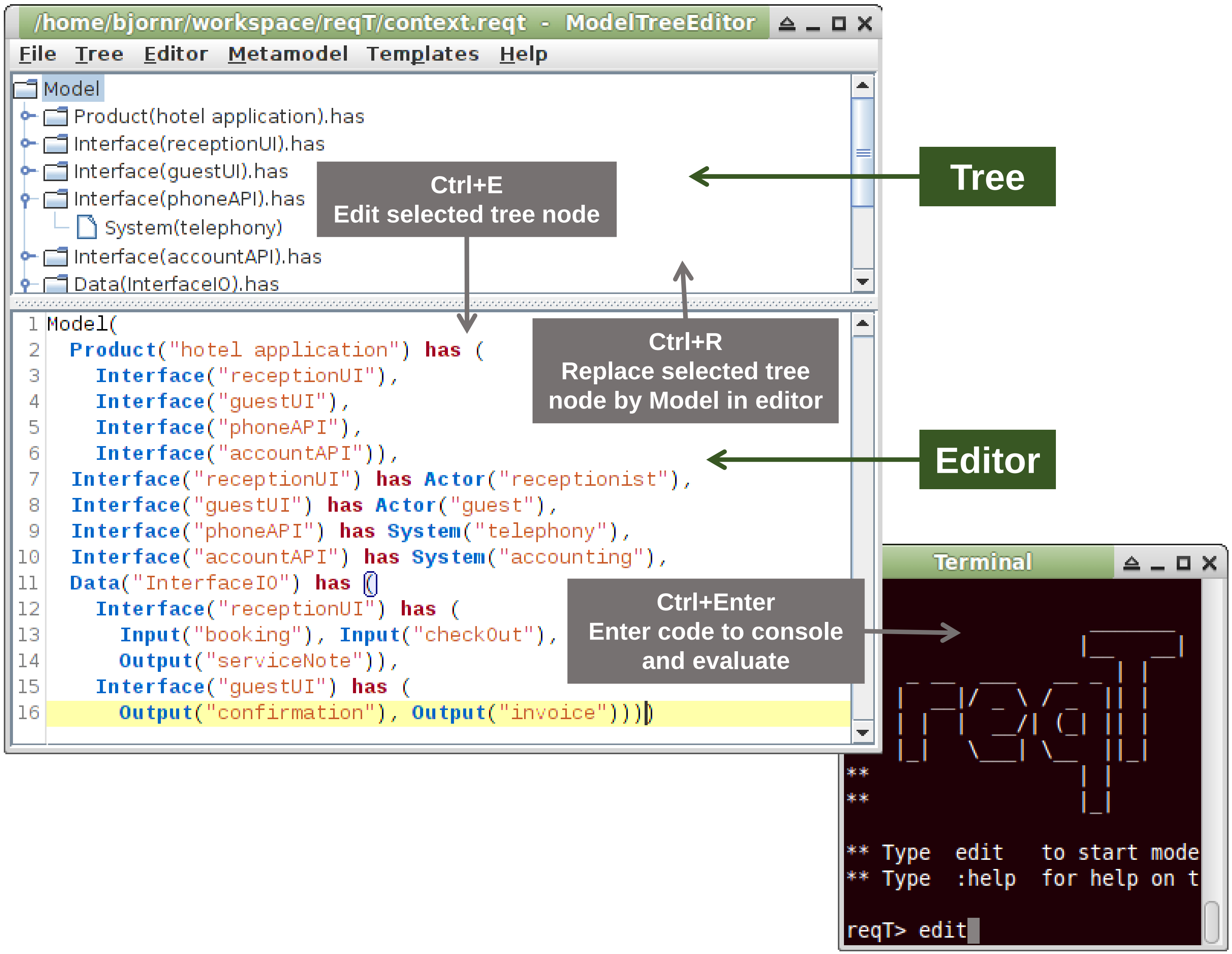
```
val m = Model(
  Feature("f1") has (Spec("A good spec."), Status(SPECIFIED)),
  Feature("f1") requires (Feature("f2"), Feature("f3")),
  Stakeholder("s1") requires Feature("f2")
)
```

```
m.toGraph.save("graph.dot")
```

```
$ dot -Tpdf graph.dot -o graph.pdf
```

http://graphviz.org




/home/bjornr/workspace/reqT/context.reqt - ModelTreeEditor
File
Tree
Editor
Metamodel
Templates
Help
Tree
Editor
Terminal
Ctrl+E Edit selected tree node
Ctrl+R Replace selected tree node by Model in editor
Ctrl+Enter Enter code to console and evaluate

```
Model
  Product(hotel application).has
  Interface(receptionUI).has
  Interface(guestUI).has
  Interface(phoneAPI).has
    System(telephony)
  Interface(accountAPI).has
  Data(InterfaceIO).has
```

```
Model(
  Product("hotel application") has (
    Interface("receptionUI"),
    Interface("guestUI"),
    Interface("phoneAPI"),
    Interface("accountAPI")),
  Interface("receptionUI") has Actor("receptionist"),
  Interface("guestUI") has Actor("guest"),
  Interface("phoneAPI") has System("telephony"),
  Interface("accountAPI") has System("accounting"),
  Data("InterfaceIO") has (
    Interface("receptionUI") has (
      Input("booking"), Input("checkOut"),
      Output("serviceNote")),
    Interface("guestUI") has (
      Output("confirmation"), Output("invoice")))
```

```
**
**

** Type  edit    to start mode
** Type  :help   for help on t

reqT> edit
```

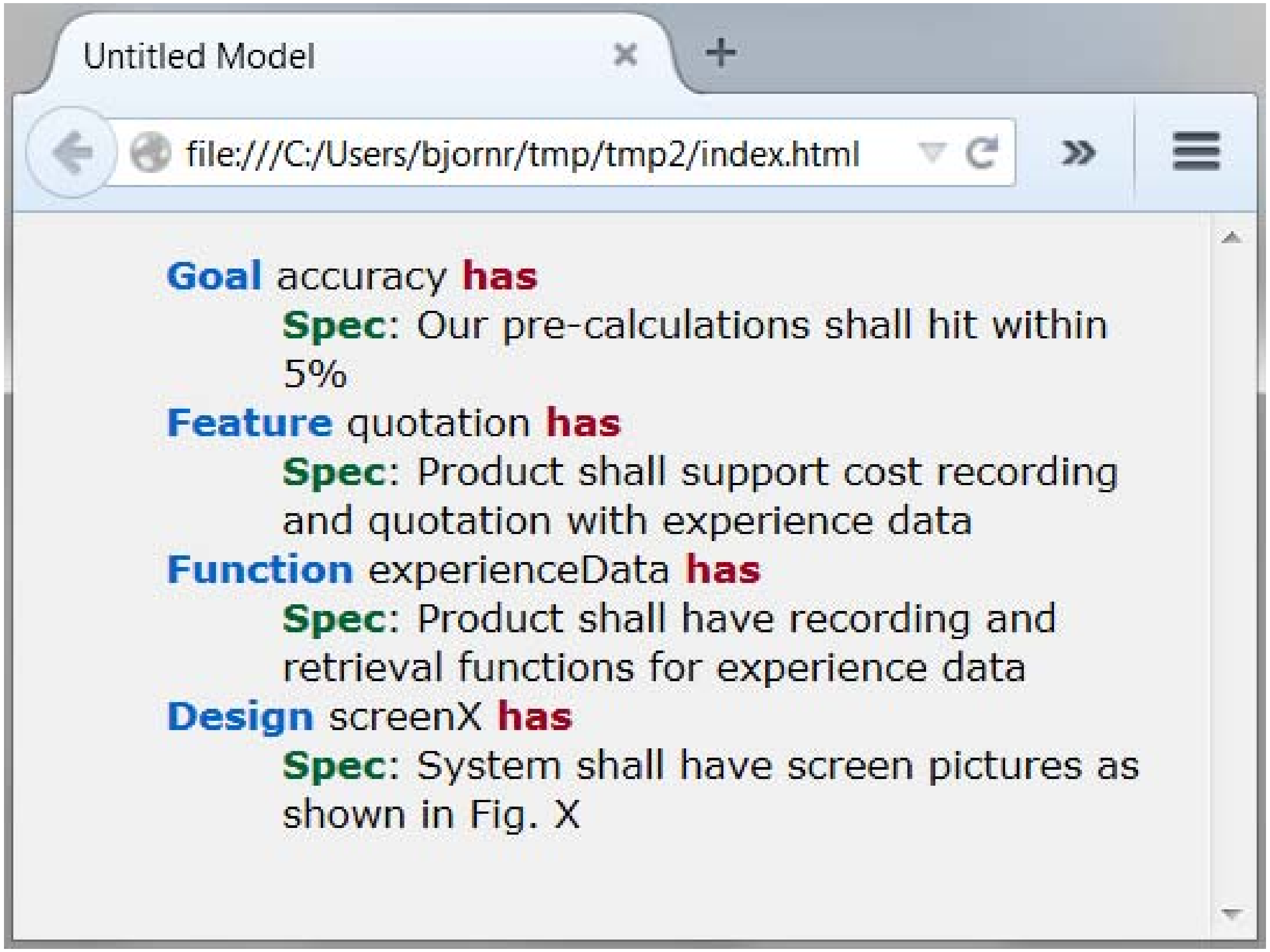
Split and merge

```
val myModel = Model(Req("x") has Spec("a"))

val yourModel = Model(Req("y") has Spec("b"))

val merged = myModel ++ yourModel

merged.toScala.save("newModel.scala")

Model(
  Req("x") has Spec("a"),
  Req("y") has Spec("b")
)
```

The goal-design scale in reqT

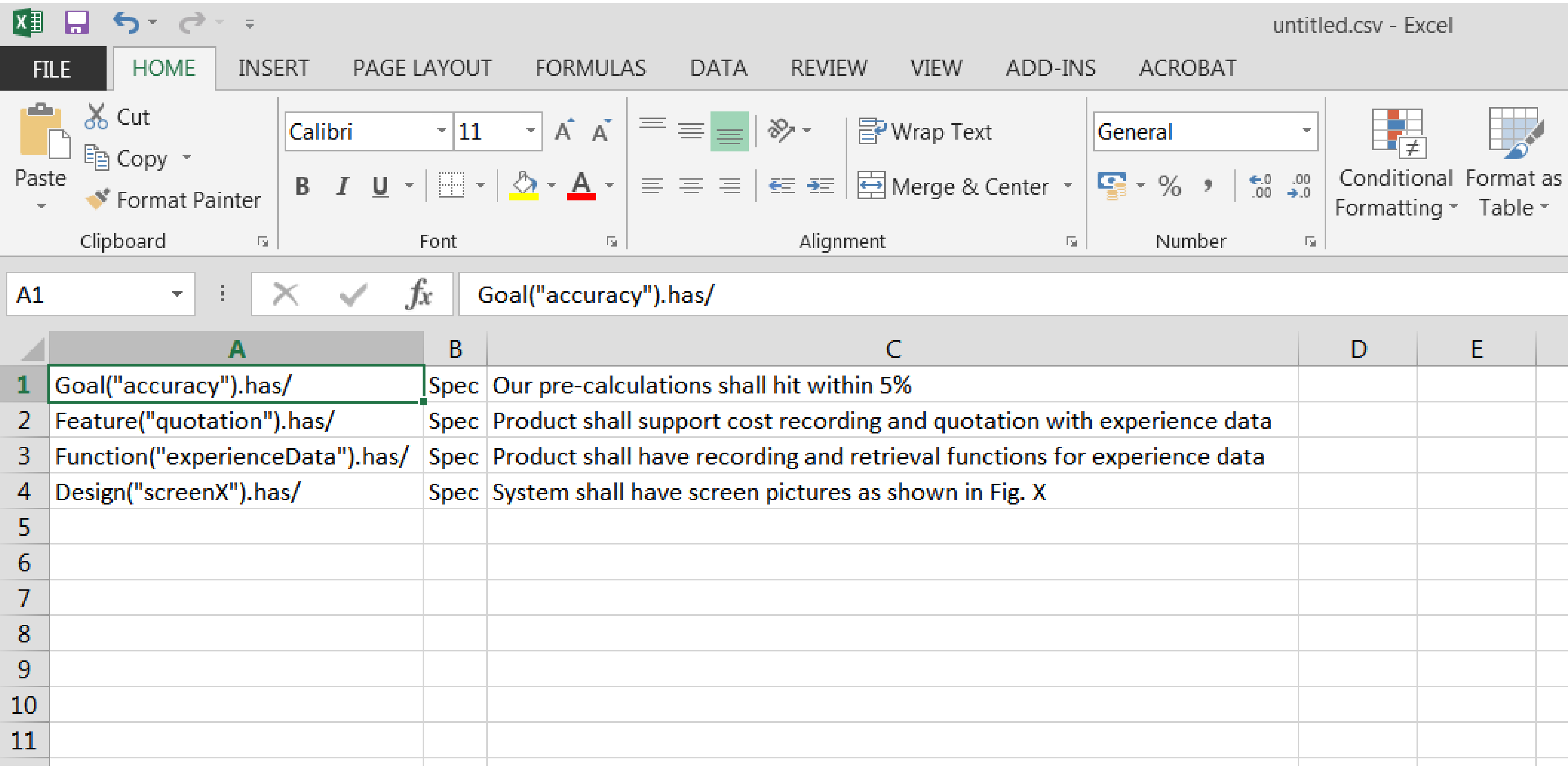
```
Model(
  Goal("accuracy") has
    Spec("Our pre-calculations shall hit within 5%"),
  Feature("quotation") has
    Spec("Product shall support cost recording and
           quotation with experience data"),
  Function("experienceData") has
    Spec("Product shall have recording and retrieval
           functions for experience data"),
  Design("screenX") has
    Spec("System shall have screen pictures as shown
           in Fig. X"))


// Adapted from RE text book by [Lauesen]
```

Product("reqT") has Feature("toHtml")



Goal accuracy **has**
    Spec: Our pre-calculations shall hit within 5%
Feature quotation **has**
    Spec: Product shall support cost recording and quotation with experience data
Function experienceData **has**
    Spec: Product shall have recording and retrieval functions for experience data
Design screenX **has**
    Spec: System shall have screen pictures as shown in Fig. X

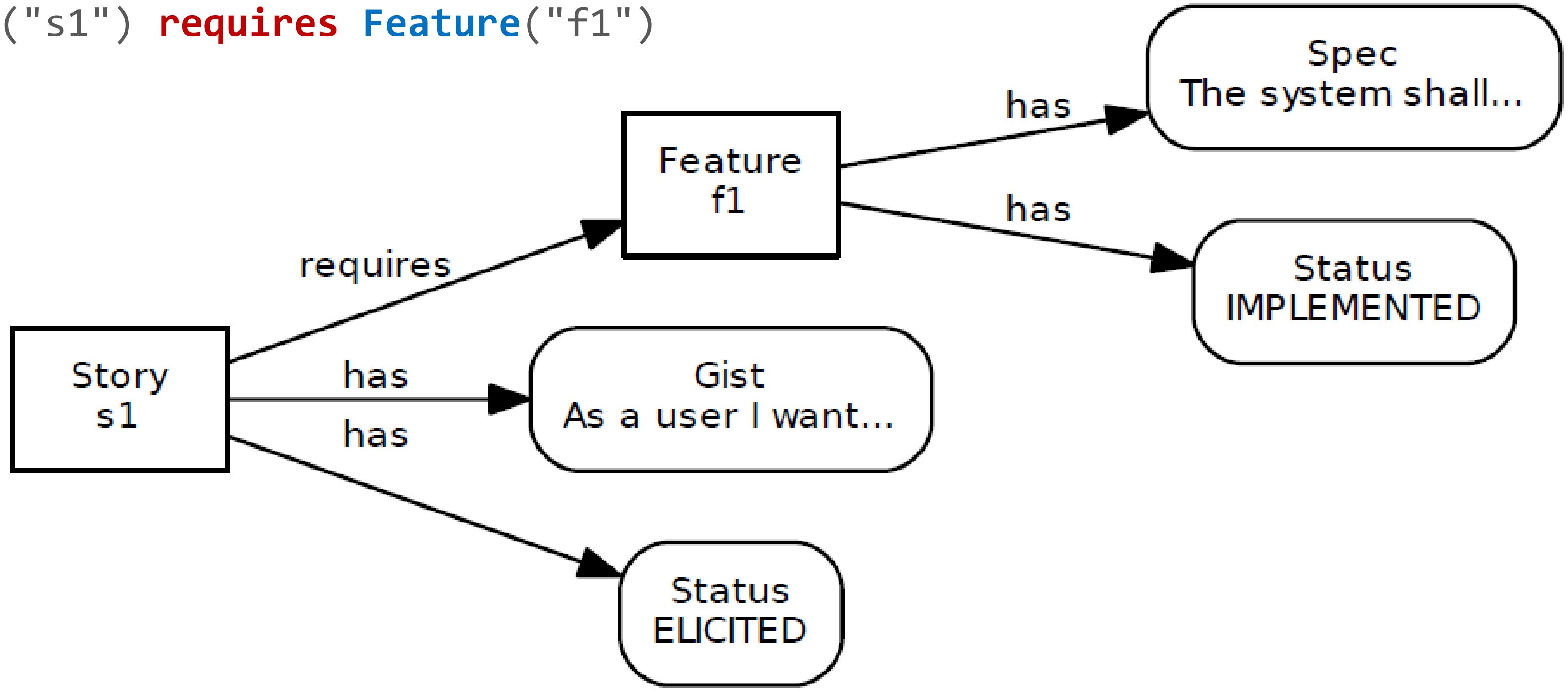
Product("reqT") has Feature("toTable")



	A	B	C	D	E
1	Goal("accuracy").has/	Spec	Our pre-calculations shall hit within 5%		
2	Feature("quotation").has/	Spec	Product shall support cost recording and quotation with experience data		
3	Function("experienceData").has/	Spec	Product shall have recording and retrieval functions for experience data		
4	Design("screenX").has/	Spec	System shall have screen pictures as shown in Fig. X		
5					
6					
7					
8					
9					
10					
11					

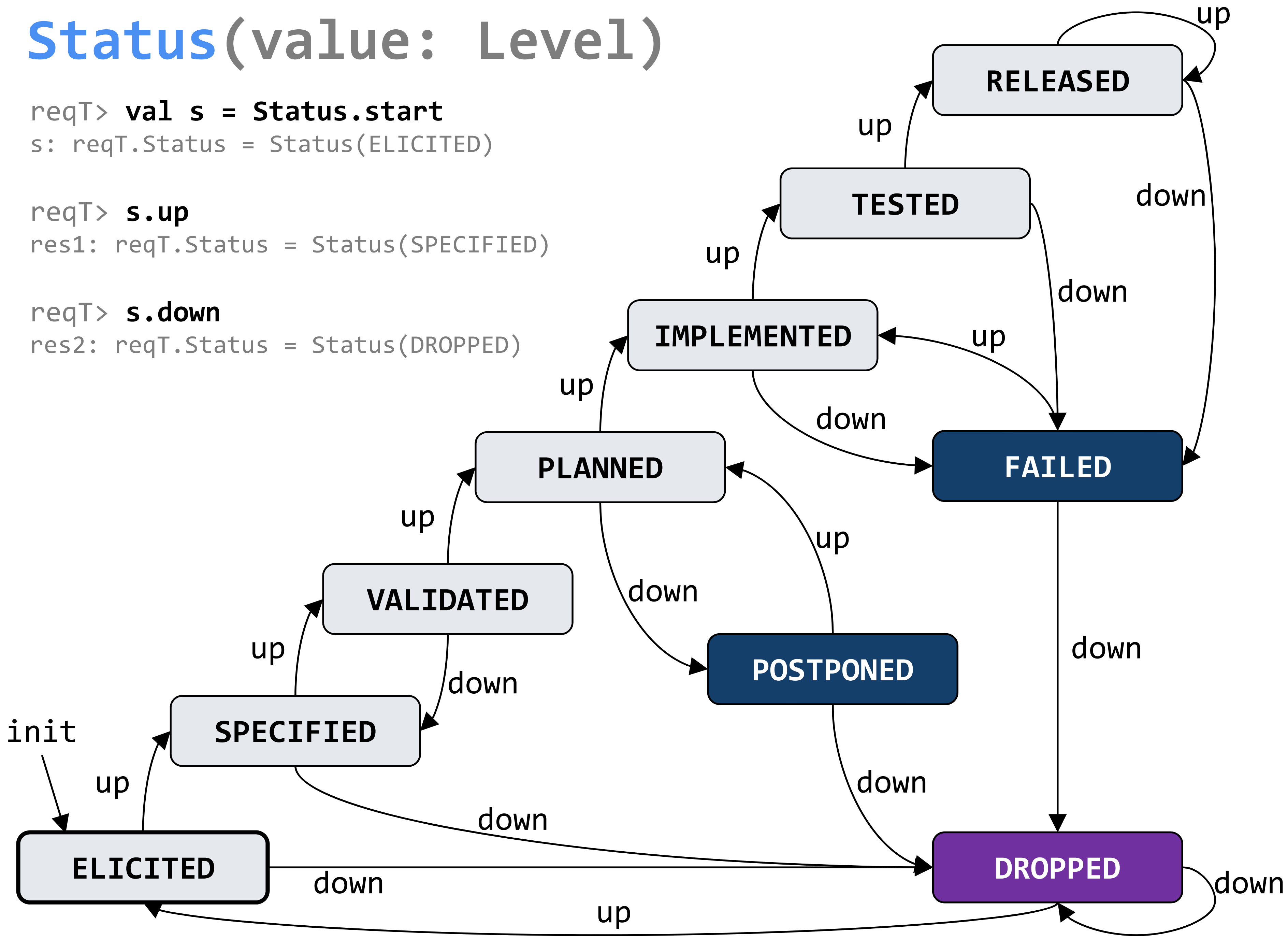
Product("reqT") has Feature("toGraph")

```
Model(
  Feature("f1") has (
    Spec("The system shall..."),
    Status(IMPLEMENTED)),
  Story("s1") has (
    Gist("As a user I want..."),
    Status(ELICITED)),
  Story("s1") requires Feature("f1")
)
```



Status(value: Level)

```
reqT> val s = Status.start
s: reqT.Status = Status(ELICITED)

reqT> s.up
res1: reqT.Status = Status(SPECIFIED)

reqT> s.down
res2: reqT.Status = Status(DROPPED)
```



Example: variability model

```
Model(
  Component("apperance") has (
    VariationPoint("color") has (
      Min(0), Max(2),
      Variant("blue"), Variant("red"), Variant("green")),
    VariationPoint("shape") has (
      Min(1), Max(1), Variant("round"), Variant("square")),
    VariationPoint("payment") has (
      Min(1), Max(2), Variant("cash"), Variant("credit")),
    VariationPoint("payment") requires Variant("cash"), /* mandatory */
    Variant("round") excludes Variant("red"),
    Variant("green") requires Variant("square")),
  Component("apperance") requires VariationPoint("shape"), /* mandatory */
  App("free") requires Component("apperance"),
  App("free") binds (
    VariationPoint("shape") binds Variant("round")),
  App("premium") requires Component("apperance"),
  App("premium") binds ( /* violating variability constraints */
    VariationPoint("color") binds (Variant("red"), Variant("green")),
    VariationPoint("shape") binds (Variant("round"), Variant("square")),
    VariationPoint("payment") binds Variant("cash")))
```

Summary: The power of reqT

- Scalable data structure from 1 to 10E4
- Scriptable with the power of Scala
- CLI + GUI for power users
- Works with git and similar tools
- Constraint solving with JaCoP
- Methodology agnostic: 'bag of concepts'
- Open metamodel => metaprogramming

The future of reqT?

- Growing requirements engineering laboratory
  - **Visualizer** by integrating some graph lib
  - **Analyzer** with metrics and checking
    - Product line engineering variability model checking
  - Integrate **testing** concepts to merge RE & VV
  - Integrate **risk** modelling
  - Integrate **NLP** and **IR** techniques
- Growing an OOS **Community**
  - Your pull requests are welcome!
- Master thesis projects
  - Front-end + back-end cloud app in HTML5?

Discussion

- What are **your requirements** on a requirements engineering open source software (OSS) tool?
  - For **research**?
  - For **teaching** ?
- How to best **involve** the academic RE community in taking a collective ownership of and contribution to a wide-spread OSS RE tool?
  - How should the reqT OSS project be **governed** to make it easy for you to contribute?

Thanks!

bjorn.regnell@cs.lth.se

Download latest reqT.jar at: http://reqT.org/
Run in terminal: java -jar reqT.jar