UNIVERSITY OF Southampton

Mastering design complexity through formal modelling and verification

Michael Butler
users.ecs.soton.ac.uk/mjb

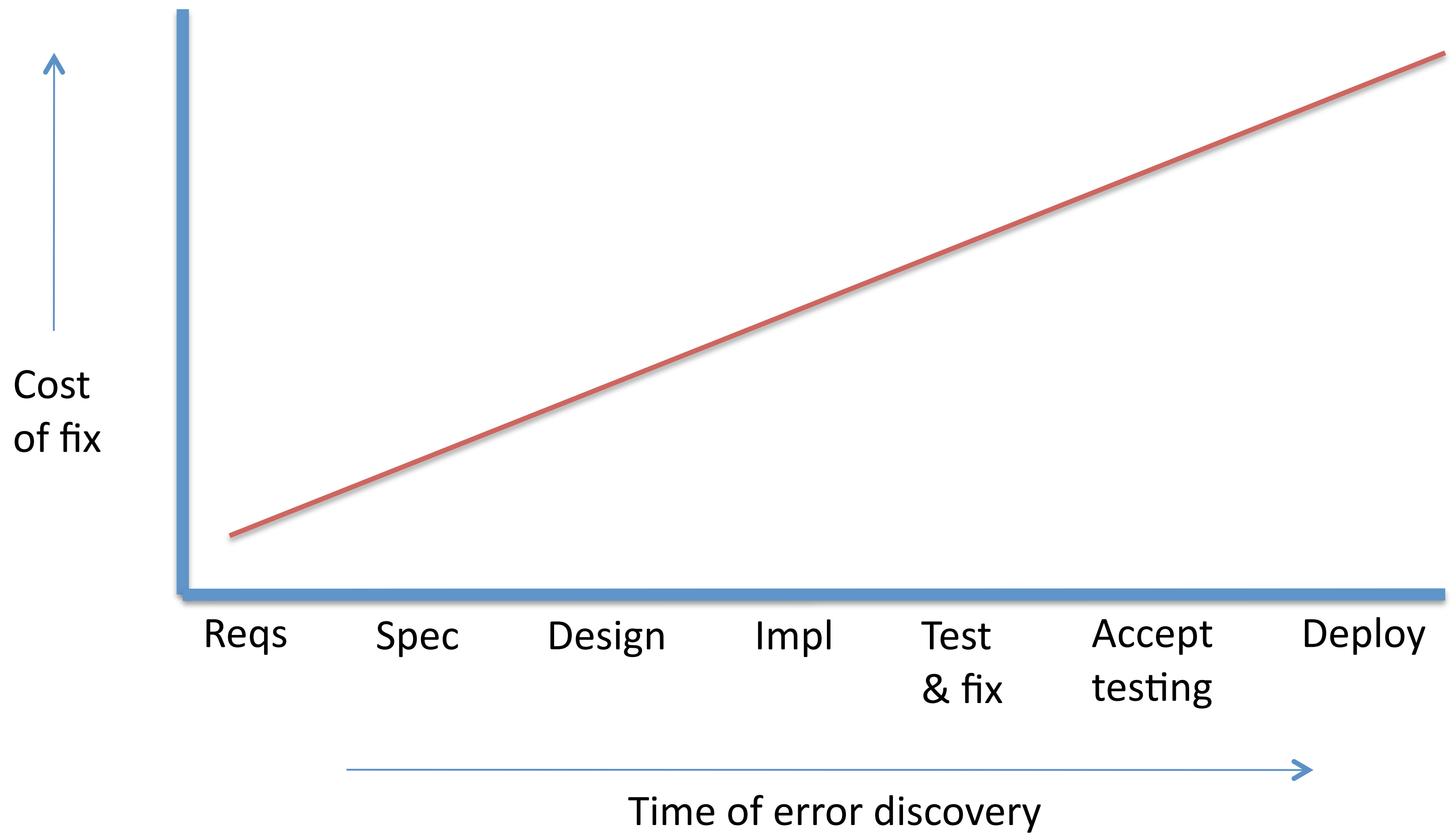
Dependable Systems and
Software Engineering Group (DSSE)

School of Electronics and Computer Science

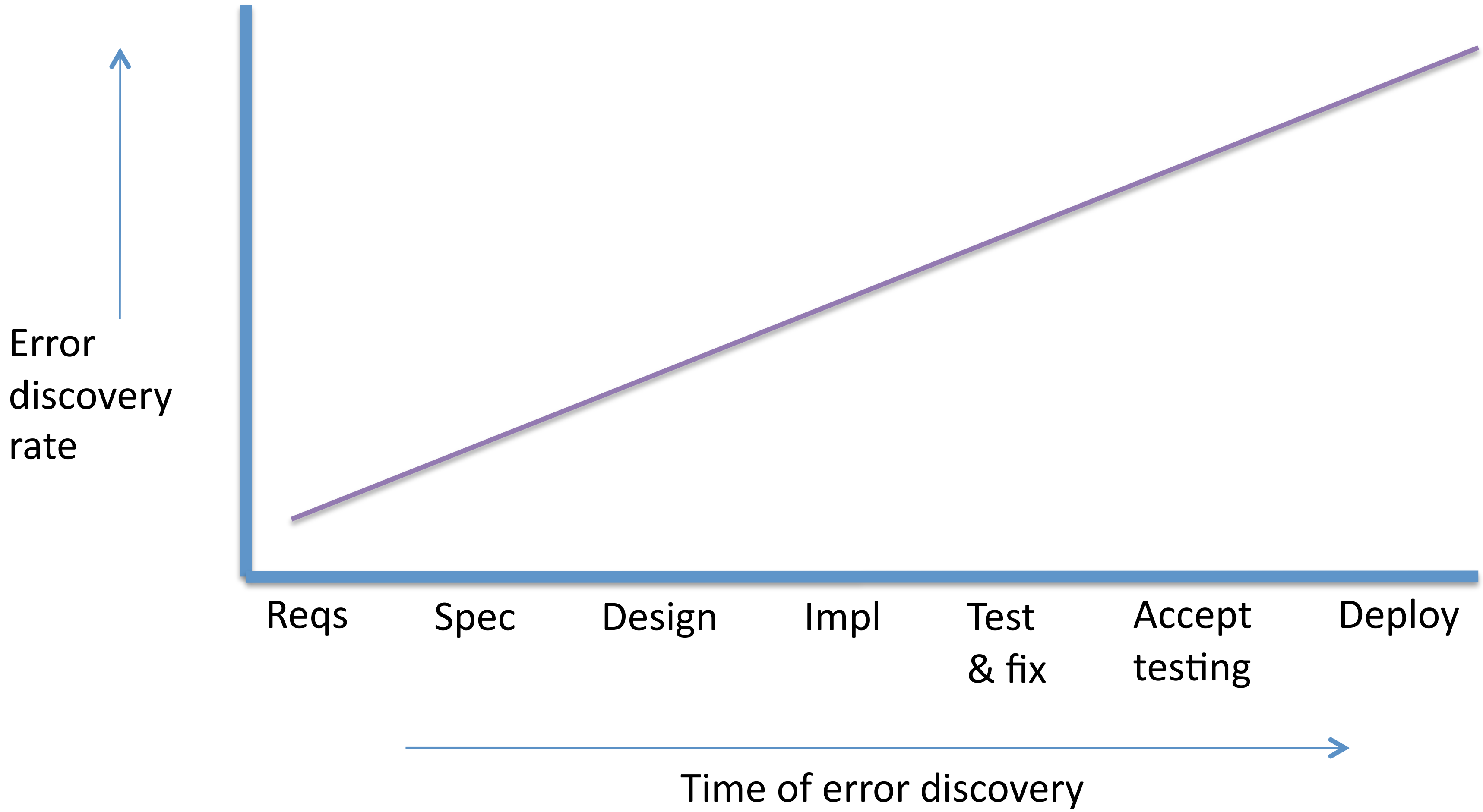
Contents

- Motivation
  - cost of fixing errors
  - difficulty of discovering errors
- Formal methods overview
  - impact on lifecycle
  - some industrial experiences
- Our approach with formal methods
  - abstraction
  - refinement
  - automated analysis
- Rodin toolset
- Current industrial collaboration

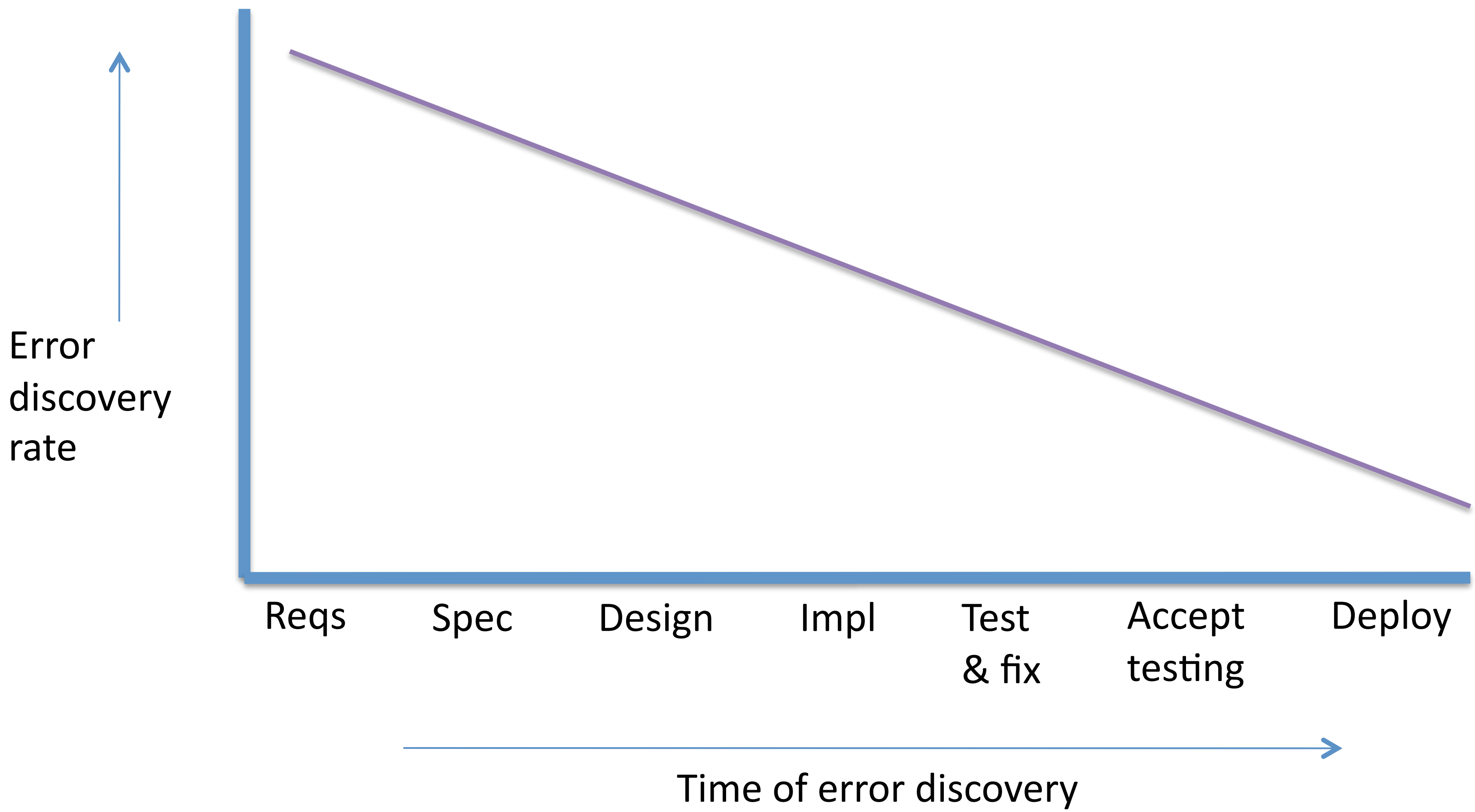
Cost of error fixes



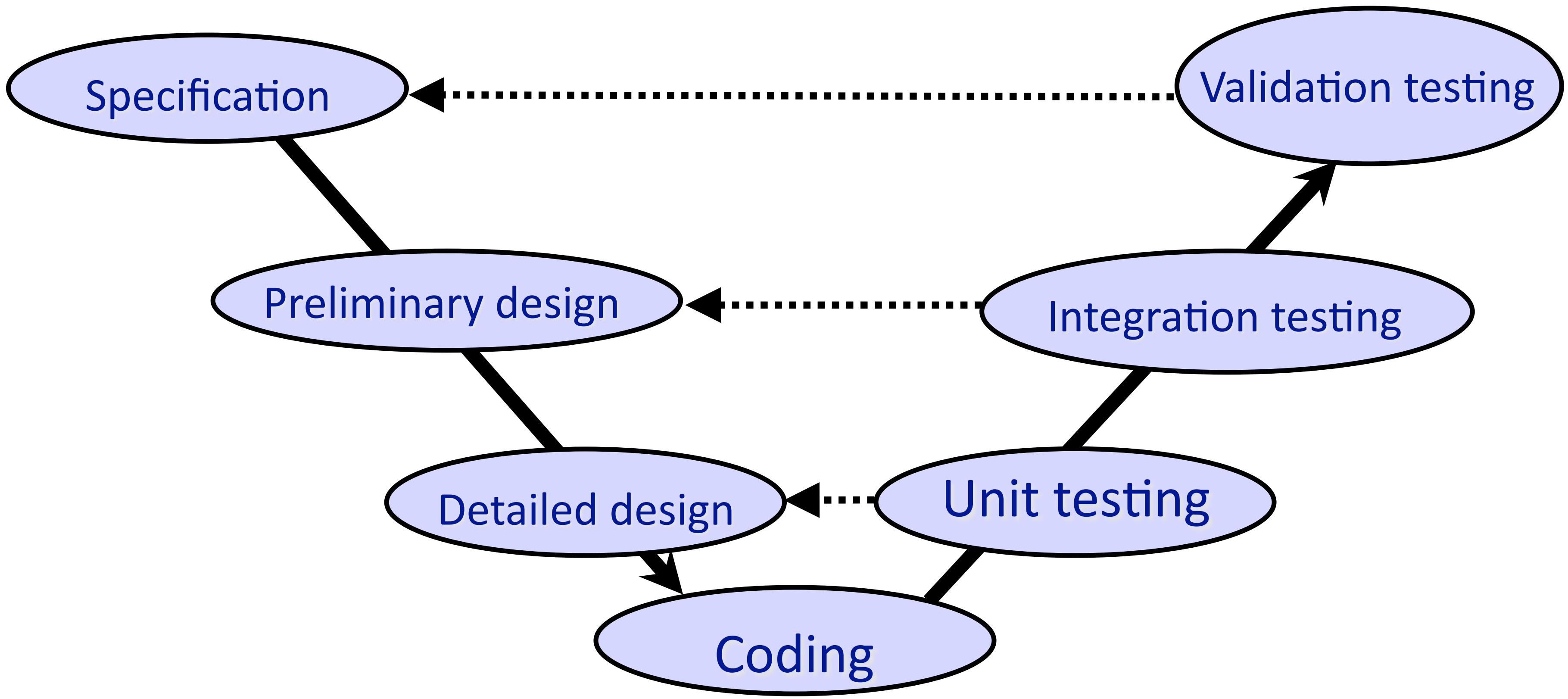
Rate of error discovery



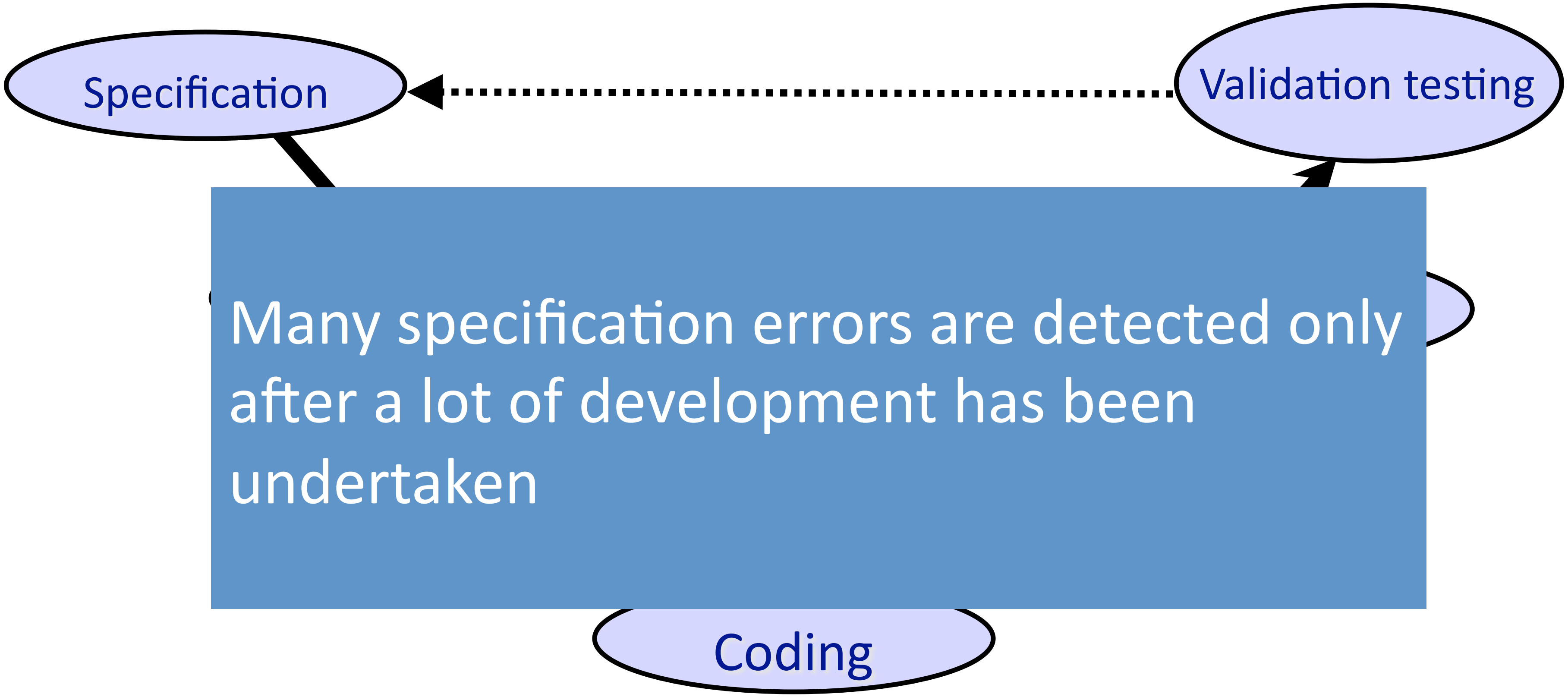
Invert error identification rate?



What's wrong with the V model?



What's wrong with the V model?



Many specification errors are detected only after a lot of development has been undertaken

Why is it difficult to identify errors?

- Lack of precision
  - ambiguities
  - inconsistencies

- Too much complexity
  - complexity of requirements
  - complexity of operating environment
  - complexity of designs

Need for precise models/blueprints

- Early stage analysis
  - Precise descriptions of intent
  - Amenable to analysis by tools
  - Identify and fix ambiguities and inconsistencies as early as possible
- Mastering complexity
  - Encourage abstraction
  - Focus on *what* a system does
  - Early focus on *key / critical* features
  - Incremental analysis and design

Contents

- Motivation
  - cost of fixing errors
  - difficulty of discovering errors
- Formal methods overview
  - impact on lifecycle
  - some industrial experiences
- Our approach with formal methods
  - abstraction
  - refinement
  - automated analysis
- Rodin toolset
- Current industrial collaboration

Formal Methods

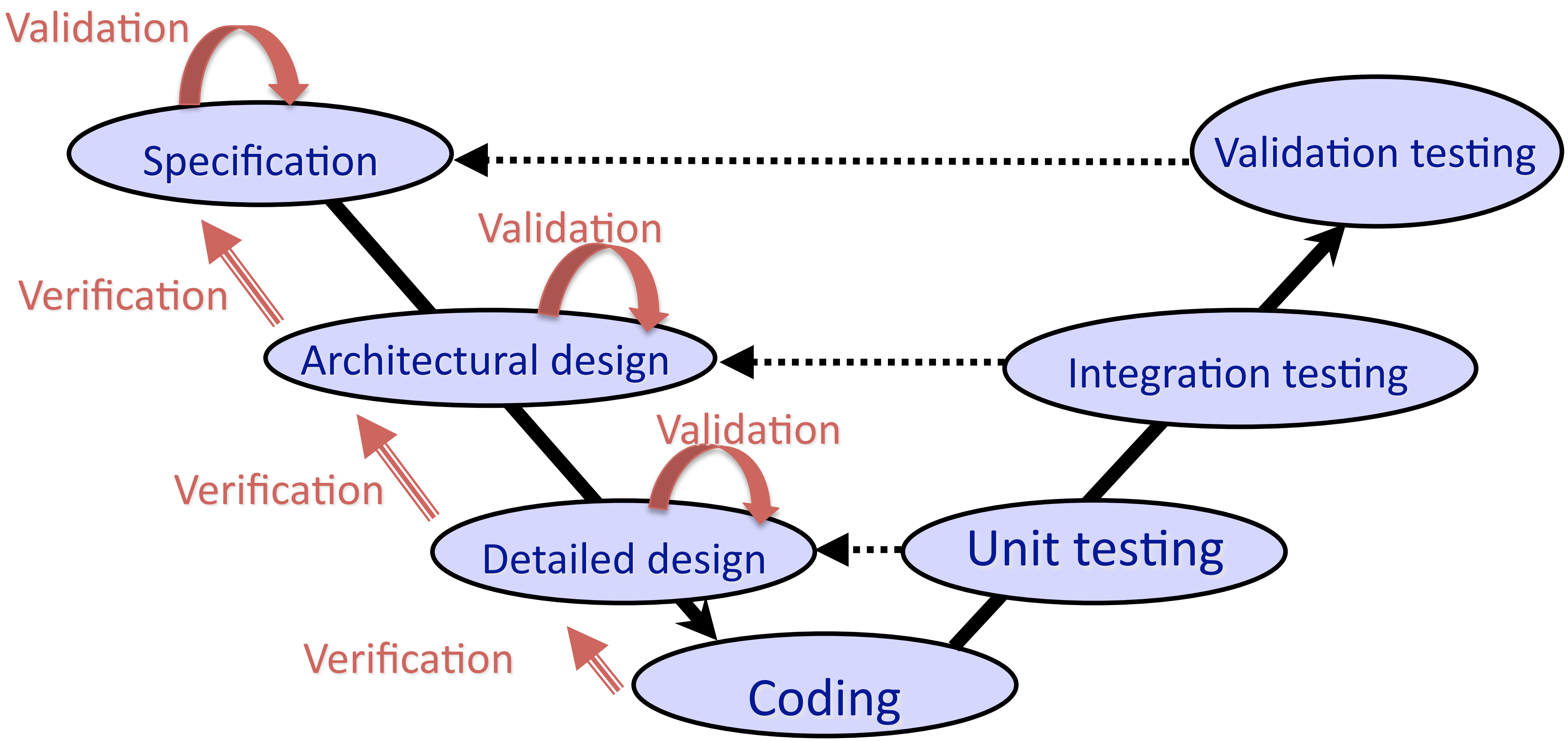
- Mathematical techniques for formulation and analysis of systems

- Formal methods facilitate:
  - Clear specifications (contract)
  - Rigorous *validation* and *verification*

Validation: does the contract specify the right system?
- answered informally

Verification: does the finished product satisfy the contract?
- can be answered formally

Early stage analysis



B Method

- *Model* using set theory and logic
- *Analyse* using proof, model checking, animation
- Refinement:
  - verify conformance between
    higher-level and *lower-level* models
  - chain of refinements
- Code generation from low-level models
- Commercial tools (*Atelier-B, B-Toolkit*)

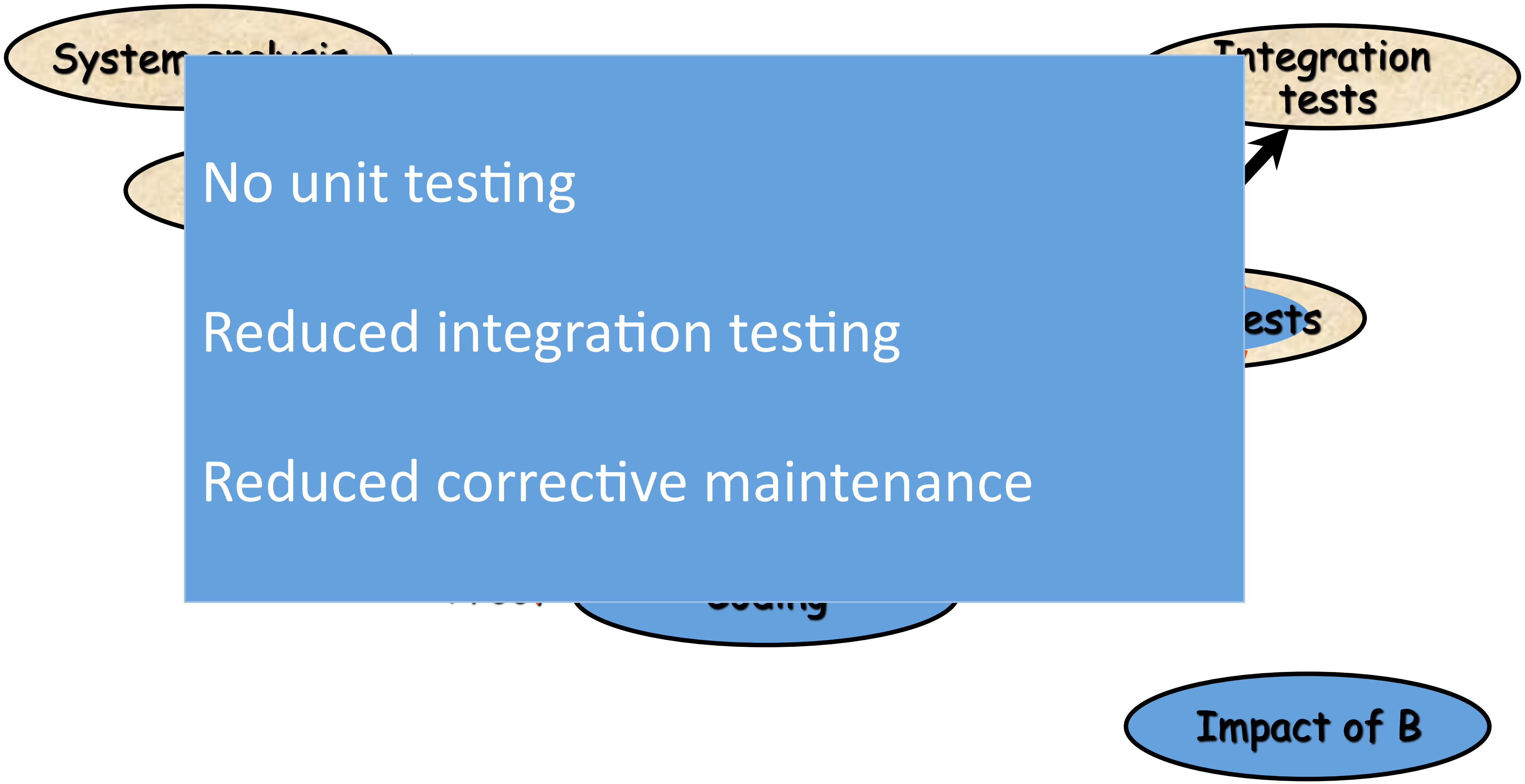
Industrial use of B in Railways

- Meteor: Paris Line 14 - Driverless
  - 117 kloB
  - 29 K proofs
  - 87 kloc (auto generated Ada)

- Canarsie: New York Line L – Mixed mode
  - 273 kloB
  - 83 K proofs
  - 110 kloc (auto generated Ada)

Source: Siemens Transportation Systems (STS)

STS development cycle with B



No unit testing

Reduced integration testing

Reduced corrective maintenance

Source: STS

Contents

- Motivation
  - cost of fixing errors
  - difficulty of discovering errors
- Formal methods overview
  - impact on lifecycle
  - some industrial experiences
- Our approach with formal methods
  - abstraction
  - refinement
  - automated analysis
- Rodin toolset
- Current industrial collaboration

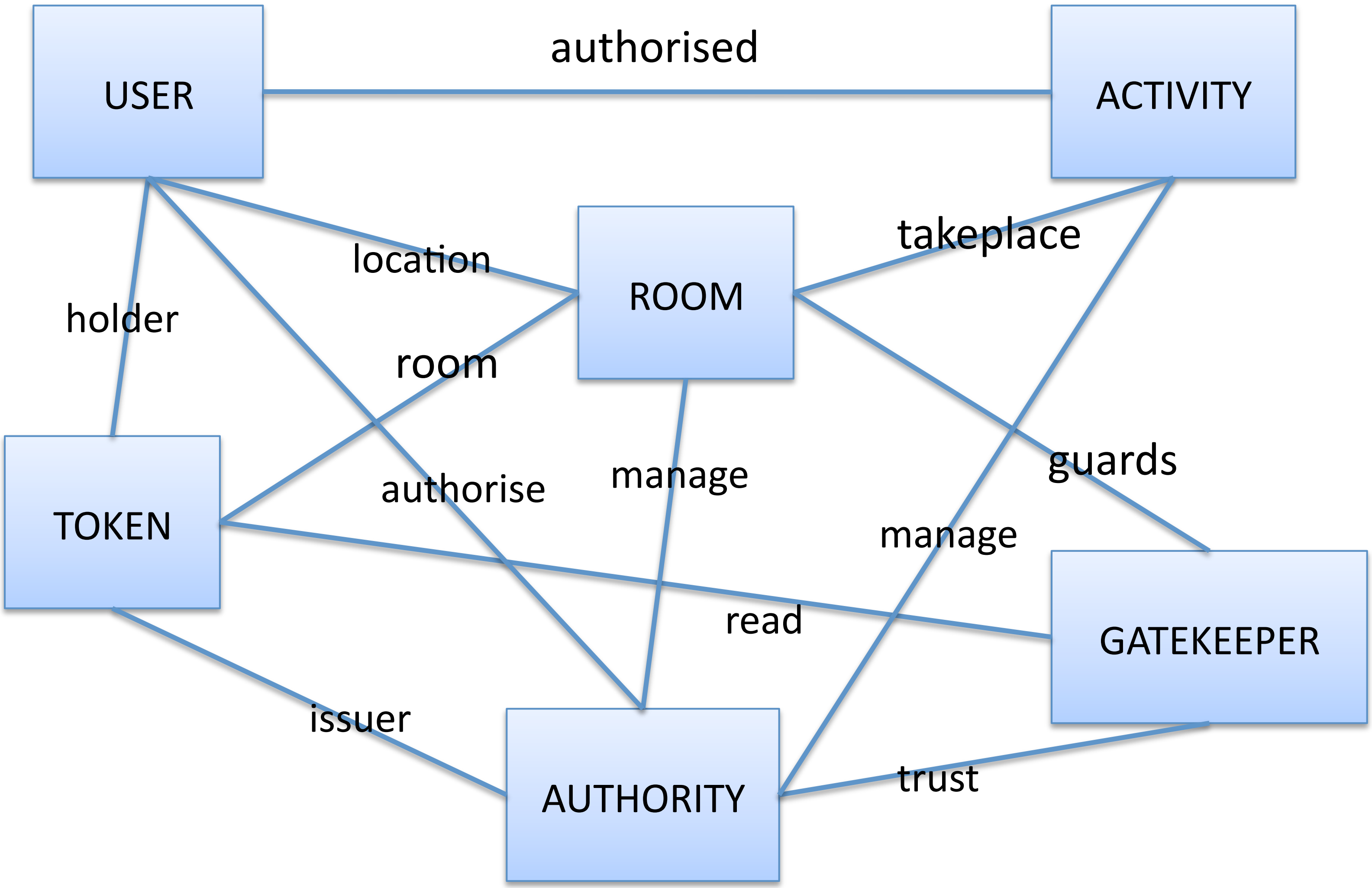
Example: authorisation system

- Example intended to give a feeling for:
  - modelling language
  - abstraction and refinement
  - mathematical analysis

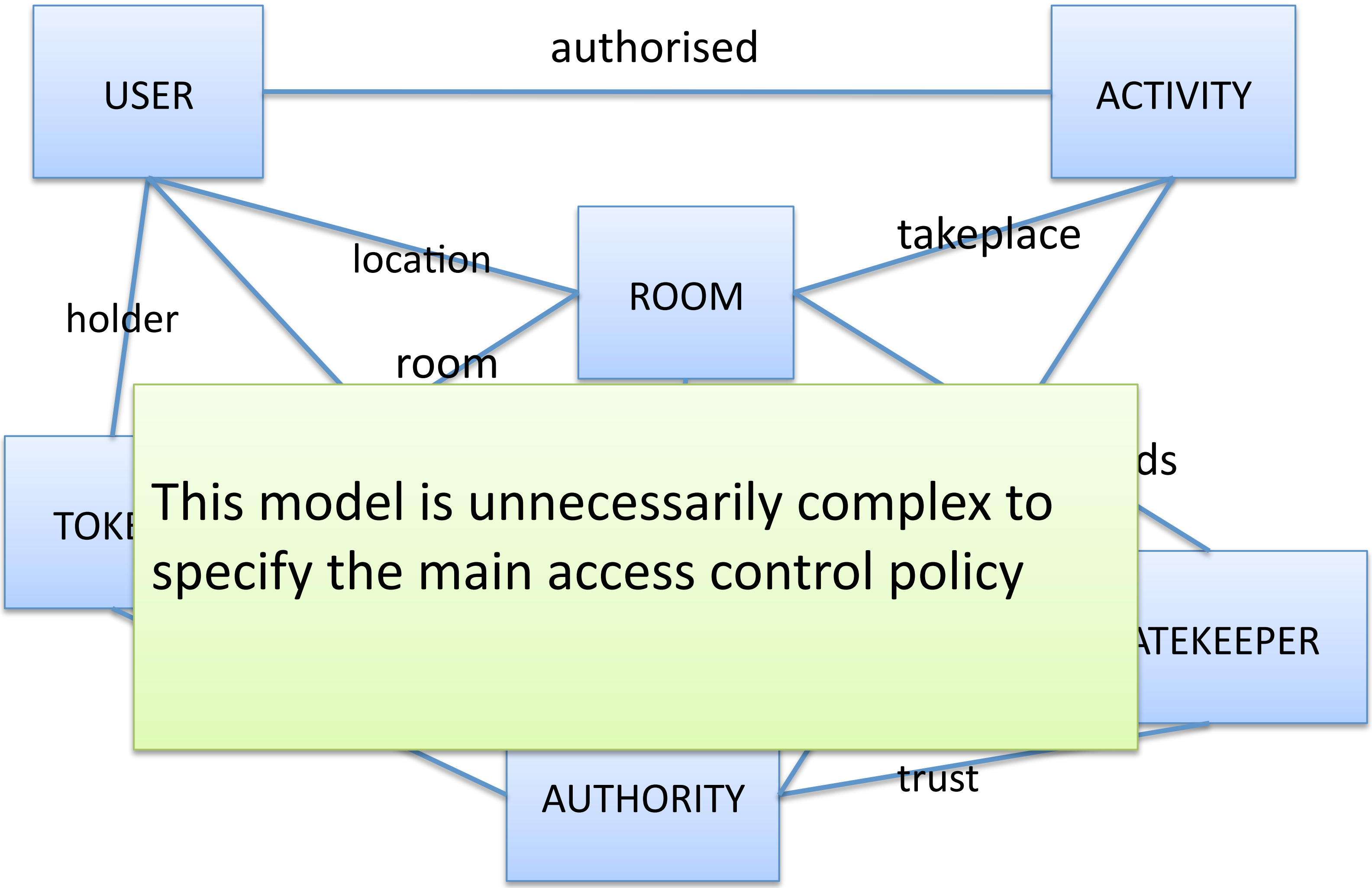
Access control system

- Users are authorised to engage in activities
- User authorisation may be added or revoked
- Activities take place in rooms
- Users gain access to a room using a one-time token provided they have authority to engage in the room activities
- Tokens are issued by a central authority
- Tokens are time stamped
- A room gateway allows access with a token provided the token is valid

Class diagram



Class diagram



This model is unnecessarily complex to specify the main access control policy

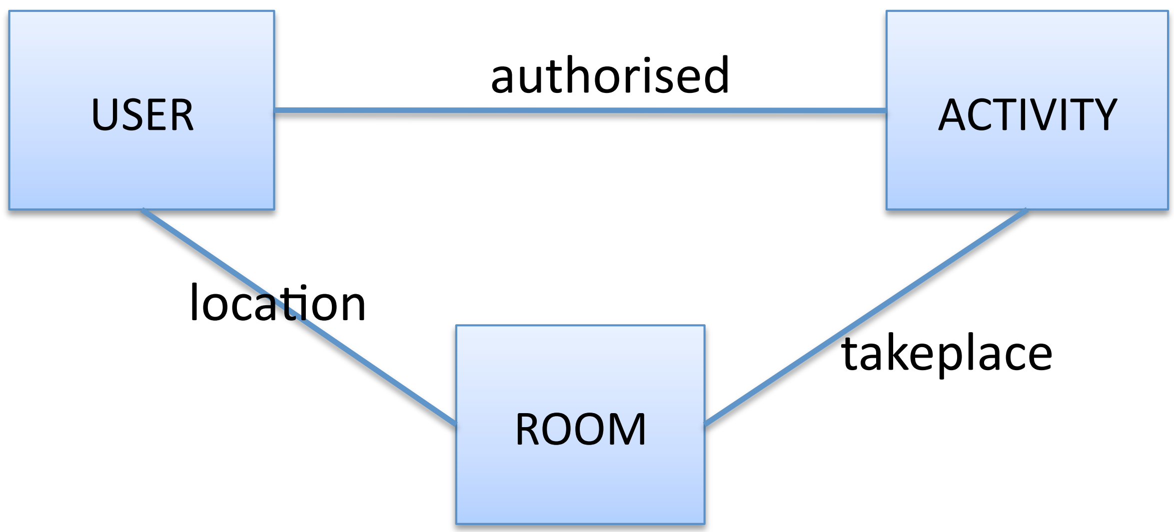
Extracting the essence

- Access Control Policy: *Users may be in a room only if they are authorised to engage in all activities that may take place in that room*

- To express this we only require Users, Rooms, Activities and relationships between them

- Abstraction: focus on key entities in the problem domain

Diagrammatic representation of an abstract model



Variables and invariants of Event-B model

Variables of Event-B model

@inv1 authorised $\in$ User $\leftrightarrow$ Activity // relation

@inv2 takeplace $\in$ Room $\leftrightarrow$ Activity // relation

@inv3 location $\in$ User $\mapsto$ Room // partial function

Access control *invariant*:

if user *u* is in room *r*,

then *u* must be authorised to engaged in all activities that can take place in r

@inv4 $\forall$ u,r . u $\in$ dom(location) $\land$ location(u) = r $\Rightarrow$
takeplace[r] $\subseteq$ authorised[u]

State snapshot as tables

User	Activity
u1	a1
u1	a2
u2	a2

authorised

Room	Activity
r1	a1
r1	a2
r2	a1

takeplace

User	Room
u1	r1
u2	r2
u3	

location

Event for entering a room

Enter $\triangleq$

when

grd1 : $u \in User$

grd2 : $r \in Room$

grd3 : $takeplace[r] \subseteq authorised[u]$

then

act1 : $location(u) := r$

end

Does this event maintain the security invariant?

Role of invariants and guards

- Invariants: specify properties of model variables that should also remain true
  - violation of invariant is undesirable
  - use (automated) proof to verify invariant preservation
- Guards: specify conditions under which events may occur
  - should be strong enough to ensure invariants are maintained
  - but not so strong that they prevent desirable behaviour

Remove authorisation

RemoveAuth(u,a) $\triangleq$

when

grd1 : $u \in User$

grd2 : $a \in Activity$

grd3 : $u \mapsto a \in authorised$

then

act1 : $authorised := authorised \setminus \{ u \mapsto a \}$

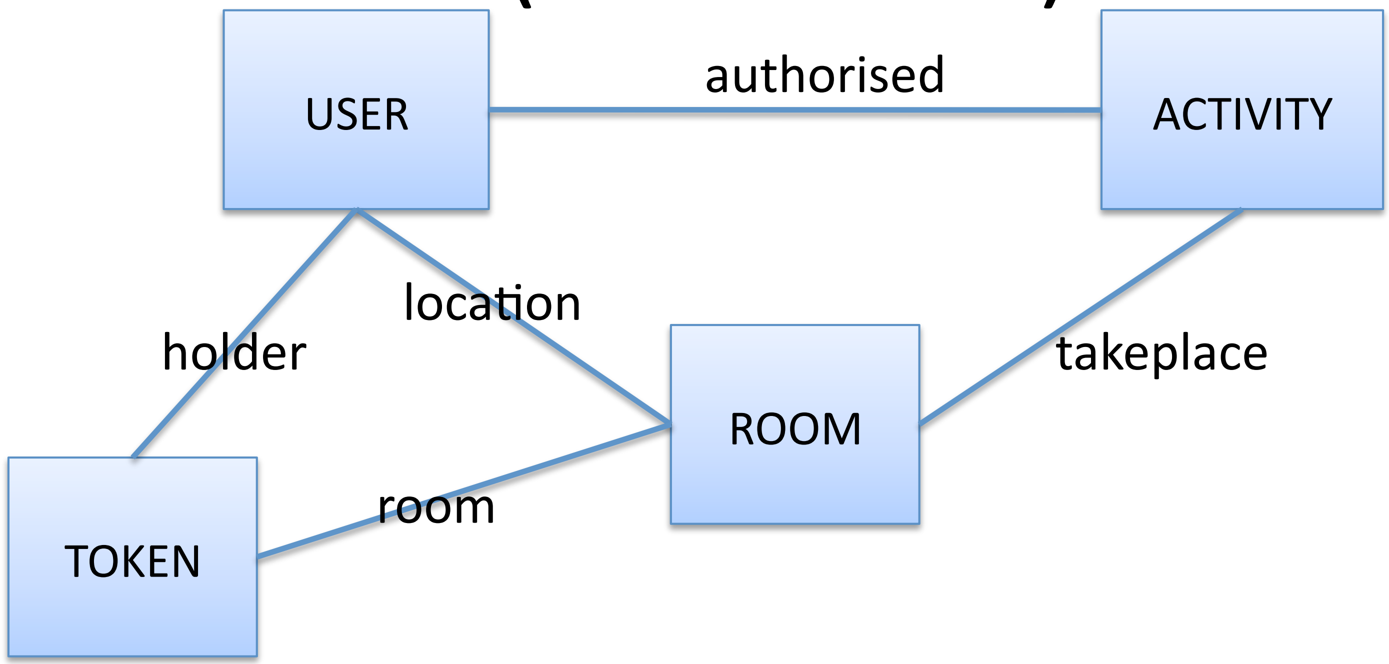
end

Does this event maintain the security invariant?

Rodin demo

- Illustrate interplay between modelling and verification

Now we construct a new model (refinement)



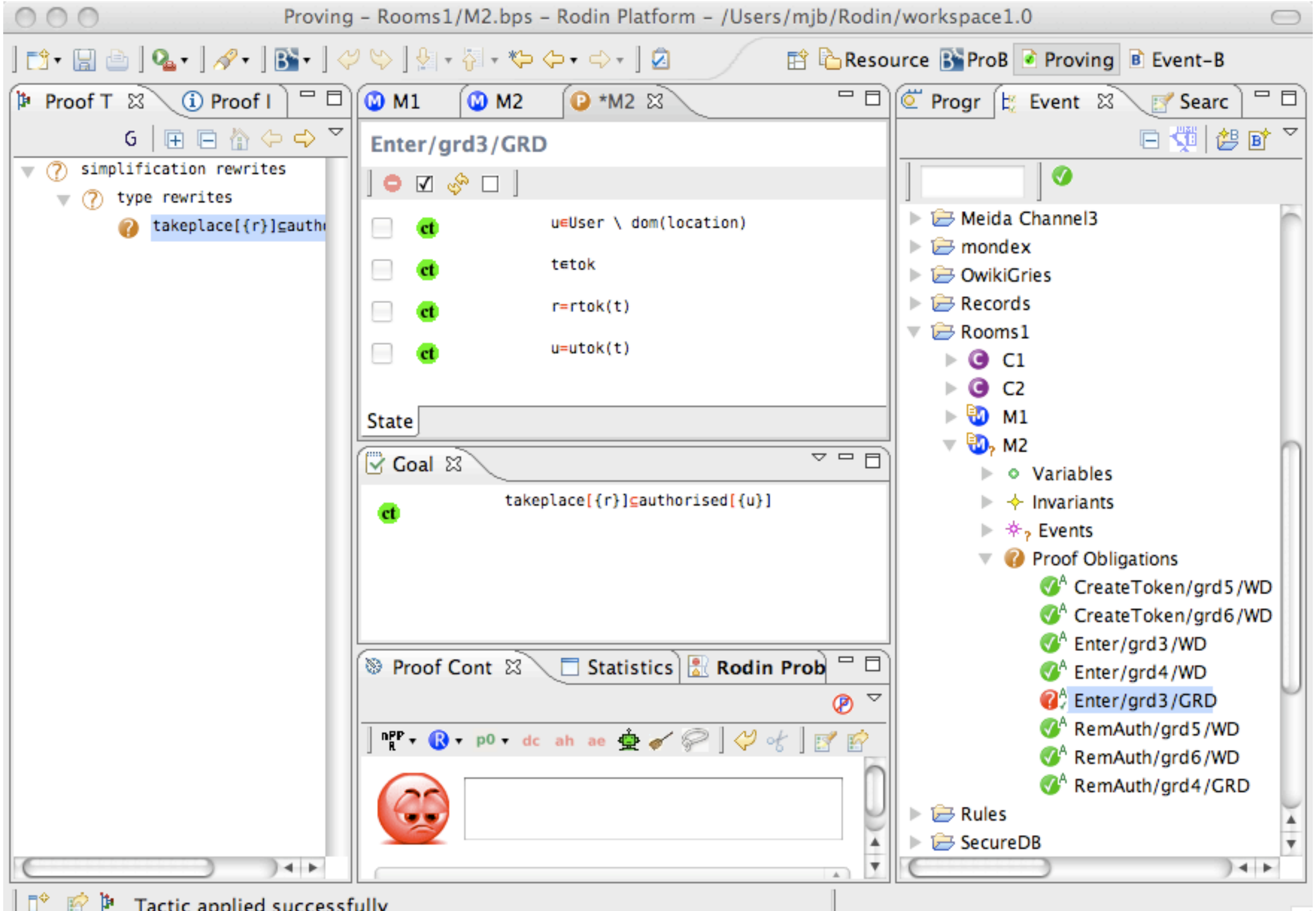
Abstract guard on a user and room for entering

grd3: takeplace[r] $\subseteq$ authorised[u]

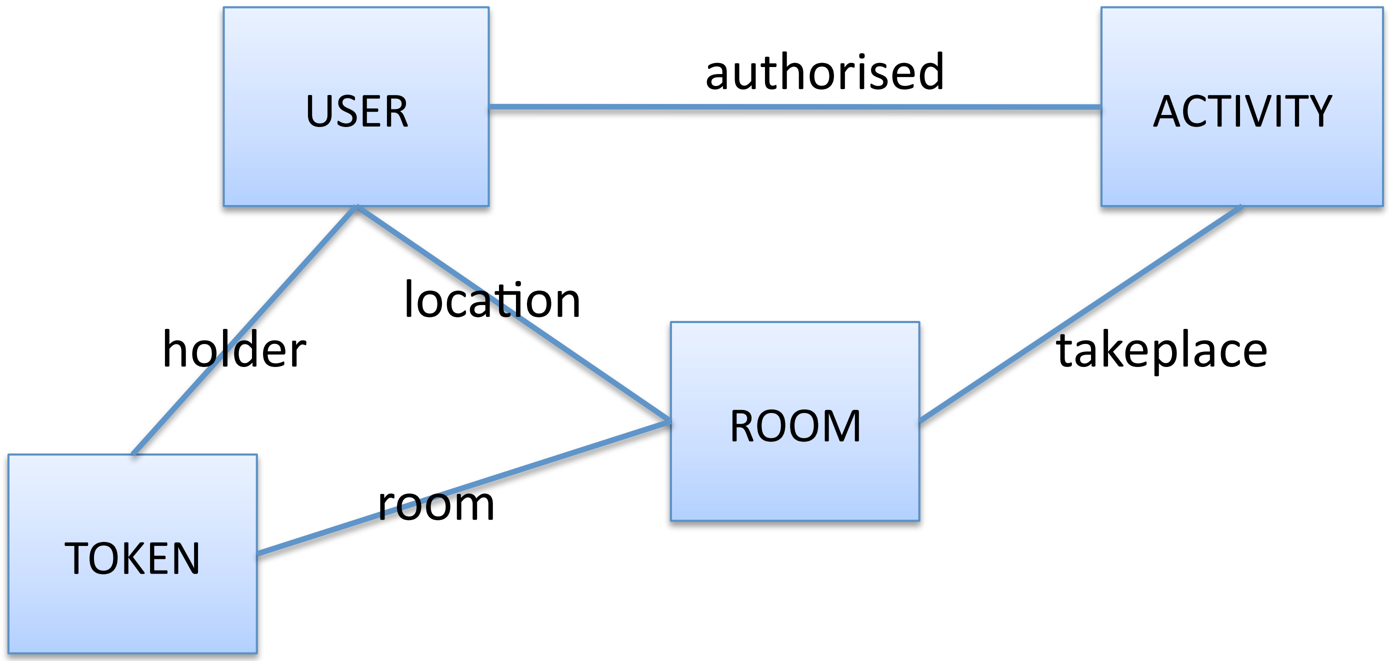
is replaced by a guard on a token

grd3b: t $\in$ valid $\wedge$ room(t) = r $\wedge$ holder(t) = u

Failing refinement proof



Gluing invariant



To ensure consistency of the refinement we need invariant:

inv 6: $t \in valid$

$\Rightarrow$

$takeplace[\, room(t)\,] \subseteq authorised[\, holder(t)\,]$

Rational design – what, how, why

- *What* does it achieve?

  if user *u* is in room *r*,

  then *u* must be authorised to engaged in all activities that can take place in r

- *How* does it work?

  Check that a user has a valid token

- *Why* does it work?

  For any valid token t, the holder of t must be authorised to engage in all activities that can take place in that room

What, how, why written in B

- *What* does it achieve?

  inv4: $u \in dom(location) \land location(u) = r$
  $\Rightarrow$
  $takeplace[r] \subseteq authorised[u]$

- *How* does it work?

  grd3b: $t \in valid \land r = room(t) \land u = holder(t)$

- *Why* does it work?

  inv5: $t \in valid$
  $\Rightarrow$
  $takeplace[room(t)] \subseteq authorised[holder(t)]$

Abstraction

- Abstraction can be viewed as a process of simplifying our understanding of a system.
- The simplification should
  - focus on the intended purpose of the system
  - ignore details of how that purpose is achieved.
- The modeller should make judgements about what they believe to be the key features of the system.

Abstraction (continued)

- If the purpose is to provide some service, then
  - model what a system does from the perspective of the service users
  - ‘users’ might be computing agents as well as humans.
- If the purpose is to control, monitor or protect some phenomenon, then
  - the abstraction should focus on those phenomenon
  - in what way should they be controlled or protected?
  - why should they be monitored?

Refinement

- Refinement is a process of enriching or modifying a model in order to
  - augment the functionality being modelled, or
  - explain how some purpose is achieved

- In a refinement step we refine one model M1 to another model M2:
  - M2 is a refinement of M1
  - M1 is an abstraction of M2
  - Don’t throw M1 away

Refinement (contined)

- We can perform a series of refinement steps to produce a series of models M1, M2, M3, ...
- Facilitates abstraction: we can postpone treatment of some system features to later refinement steps
- Event-B provides a notion of consistency of a refinement:
  - We use proof to verify the consistency of a refinement step
  - Failing proof can help us identify inconsistencies in a refinement step

Proof obligations in Event-B

- Well-definedness
  - e.g, avoid division by zero, out of bounds access
- Invariant preservation
  - each event maintains invariants
- Guard strengthening
  - Refined event only possible when abstract event possible
- Simulation
  - update of abstract variable correctly simulated by update of concrete variable

Proof and model checking

- Model checking: force the model to be finite state and explore state space looking for invariant violations
  - completely automatic
  - powerful debugging tool (counter-example)
- (Semi-)automated proof: based on logical deduction rules
  - no restrictions on state space
  - leads to discovery of invariants that deepen understanding
  - not completely automatic

Event-B is not the full solution

- Event-B is a general purpose formalism
- Particular domains/paradigms require additional guidelines, patterns and language extensions
  - some results on this in Deploy
- Not tied to any specific requirements engineering approach
  - possible to link with approaches, e.g., Problem Frames
- Can use alternative syntax such as UML
  - UML-B (class diagrams, state machine diagrams)
  - Integration with SAP UML-like language and tool
- Not tied to any specific programming language
  - Classical B has automatic generation of Ada and C
  - In Deploy working on code generation from Event-B (Ada and C)
- No support for continuous or stochastic reasoning in Event-B
  - some on-going work

Important Messages

- Formal modelling can be applied to *systems*
- Role of formal modelling:
  - increase understanding
  - decrease errors
- Role of refinement:
  - manage complexity through multiple levels of abstraction
- Role of verification:
  - improve quality of models (consistency, invariants)
- Role of tools:
  - make verification as automatic as possible, pin-pointing errors and even *suggesting* improvements
- Event-B can and should be linked with complementary methods

Contents

- Motivation
  - cost of fixing errors
  - difficulty of discovering errors
- Formal methods overview
  - impact on lifecycle
  - some industrial experiences
- Our approach with formal methods
  - abstraction
  - refinement
  - automated analysis
- Rodin toolset
- Current industrial collaboration

Rodin Open Tool Platform

- Extension of Eclipse IDE
- Repository of structured modelling elements
- Rodin Eclipse Builder manages:
  - Well-formedness + type checker
  - Consistency/refinement PO generator
  - Proof manager
  - Propagation of changes

- Extension points

www.event-b.org

Rodin Plug-ins

- Linking UML and Event-B
- ProB model checker: animation, consistency and refinement checking
- Graphical model animation
- Requirements management
- Code generation
- …

Contents

- Motivation
  - cost of fixing errors
  - difficulty of discovering errors
- Formal methods overview
  - impact on lifecycle
  - some industrial experiences
- Our approach with formal methods
  - abstraction
  - refinement
  - automated analysis
- Rodin toolset
- Current industrial collaboration ←

DEPLOY Integrated Project

Industrial deployment of advanced system engineering methods for high productivity and dependability

Strategic Objective ICT-2007.1.2:
Service and Software Architectures, Infrastructures and Engineering

2008 to 2010

www.deploy-project.eu

Industrial deployment partners

The industrial deployment is in 4 major sectors

- Bosch: automotive

- Siemens: rail transportation

- Space Systems Finland: space systems

- SAP: business information

DEPLOY Goals

- Understand and justify the role of formal engineering methods in building dependable systems
- Address the barriers to deploying formal engineering methods in industry
- Achieve deployment of formal engineering methods
- Scale and professionalise Rodin technology

DEPLOY Associates

- AeS, Sao Paulo
  - Rail system pilot

- Critical Software Technologies, Southampton
  - Avionics display pilot

Concluding

- Mastering complexity through formal modelling and analysis
  - Encourage abstraction
  - Focus on *what* a system does
  - Focus on *key / critical* features
  - Incremental analysis and design

- DEPLOY + Rodin
  - Industrial deployment of methods and tools
  - focus on early stage design