Building Dependability Arguments for Software-Intensive Systems

Robert Seater
February 3rd, 2009

Irradiate!
England, 1980
How it works
How it works

- React mode
- Cat open
- Cool closed
- Level = 30%
- Reaction chamber
- Coolant
How it works
How it works
How it works
Why it works
Why it works

if ERROR SIGNAL then
  status CAT @t = status CAT @t+1
  status COOL @t = status COOL @t+1
else if SHUTDOWN MODE then
  CAT closed @t+1
  COOL closed @t+1
else if REACT MODE then
  if CAT closed @t => CAT open @t+1
  if CAT open @t => COOL open @t+1
    and CAT open @t+1

if CAT open, COOL closed
  level @t = (level @t+1) +10%
if CAT closed, COOL open
  level @t = (level @t+1) -10%
else
  level @t = level @t+1
What happened

If ERROR SIGNAL then
status CAT @t = status CAT @t+1
status COOL @t = status COOL @t+1
else if SHUTDOWN MODE then
CAT closed @t+1
COOL closed @t+1
else if REACT MODE then
if CAT closed @t => CAT open @t+1
if CAT open @t => COOL open @t+1
and CAT open @t+1

If CAT open, COOL closed
level @t = (level @t+1) +10%
if CAT closed, COOL open
level @t = (level @t+1) -10%
else
level @t = level @t+1

Release <=>
level = 100%

Component needs attention
=> no release

React mode

CAT open

COOL closed

Error signal

Needs attention
What happened

if ERROR SIGNAL then
  status CAT @t = status CAT @t+1
  status COOL @t = status COOL @t+1
else if SHUTDOWN MODE then
  CAT closed @t+1
  COOL closed @t+1
else if REACT MODE then
  if CAT closed @t => CAT open @t+1
  if CAT open @t => COOL open @t+1
  and CAT open @t+1

if CAT open, COOL closed
  level @t = (level @t+1) +10%
else if CAT closed, COOL open
  level @t = (level @t+1) -10%
else
  level @t = level @t+1
What happened

if ERROR SIGNAL then
  status CAT @t = status CAT @t+1
  status COOL @t = status COOL @t+1
else if SHUTDOWN MODE then
  CAT closed @t+1
  COOL closed @t+1
else if REACT MODE then
  if CAT closed @t => CAT open @t+1
  if CAT open @t => COOL open @t+1
  and CAT open @t+1

if CAT open, COOL closed
  level @t = (level @t+1) +10%
if CAT closed, COOL open
  level @t = (level @t+1) -10%
else
  level @t = level @t+1

release

VENT

level = 100%

CAT open

COOL closed

ERROR SIGNAL

needs attention

release <=> level = 100%

component needs attention => no release

ERROR SIGNAL <=> needs attention
Who to blame?

If ERROR SIGNAL then
status CAT @t = status CAT @t+1
status COOL @t = status COOL @t+1
else if SHUTDOWN MODE then
CAT closed @t+1
COOL closed @t+1
else if REACT MODE then
if CAT closed @t ⇒ CAT open @t+1
if CAT open @t ⇒ COOL open @t+1
and CAT open @t+1

If CAT open, COOL closed
level @t = (level @t+1) + 10%
If CAT closed, COOL open
level @t = (level @t+1) - 10%
else
level @t = level @t+1

VENT
release
level = 100%

CAT open
COOL closed
ERROR SIGNAL

need attention

component needs attention
⇒ no release

needs attention
Who to blame?

what would have prevented the problem?

not a matter of better components

- redundant & resilient components
- software analysis & testing

not obvious which spec is wrong

- SW should not simply halt on error
- tank should trigger coolant
- vent should pipe to enclosed tank

lack of system level understanding

- undocumented assumptions
- poor system-level analysis
Contributions

- **Requirement Progression**
  systematically derive specifications from requirements

- **CDAD framework**
  Composite Dependability Argument Diagrams
  structure end-to-end arguments

- **2 Case Studies**
  - **Proton Therapy**
    link requirement progression to code analysis
  - **Electronic Voting**
    link fidelity to secrecy, auditability
Traffic Light
Traffic Light
Traffic Light

Control Unit

Light Unit

Cars
Traffic Light Domains

- Control Unit
- Light Unit
- Cars
Traffic Light Phenomena

Control Unit
- signal pulses (sent)
- internal state

Light Unit
- signal pulses (received)
- lights (on/off)
- light observations (possible)

Cars
- car positions
- car directions
- light observations (made)
Traffic Light
Shared Phenomena
Traffic Light Requirement

- Control Unit
  - signal pulses
  - to Light Unit
- Light Unit
  - light observations
  - to Cars
- Car directions
- Car locations
- Safety Requirement
  - cars do not collide
Traffic Light
Spec & Assumptions

- **Control Unit**
  - Machine Specification: signal pulses sent in certain pattern

- **Light Unit**
  - Light Unit Assumptions: signal pulses determine observations made

- **Cars**
  - Car Assumptions: cars observe & obey status of lights
What if something is missing?

identify elements of implication

- Machine Spec $\land$ Light Unit Assumptions $\land$ Car Assumptions $\Rightarrow$ Requirement

what if something is missing?

- nothing missing: check implication
- one missing: identify weakest
- several missing: decompose
What is typically done?

unstructured approach
- put engineers in room together
- document the resulting design

resulting specification document is
- too vague - missing explanations
- too weak - implicit assumptions
- too strong - implementation bias
A new approach: 
requirement progression

systematic process to guide human 
proposing meaningful assumptions

explicitly document assumptions 
using precise language

automatically check proposals
Light Unit Behavior

South Unit

Controller

North Unit
Light Unit Behavior

South Unit -> Controller -> North Unit

south red signal pulse
Light Unit Behavior

South Unit

Controller

North Unit

north green signal pulse
Light Unit Behavior

South Unit → Controller → North Unit

South red signal pulse
Light Unit Behavior

South Unit -> Controller

Controller -> North Unit

north red signal pulse
Car Behaviors
Car Behaviors
Requirement Progression
frame diagram

The control unit will send a certain pattern of red and green signals...

...and since the signal pulses to the light determine if cars observe green lights...

...and since cars are only in the intersection if they observe a green light in their direction...

...then cars moving in opposite directions will never be in the intersection at the same time...
Requirement Progression

Formal requirement

\[
\text{no } t \text{ time } | \text{some } c_1, c_2 : \text{Cars} | \text{CarDirection}(c_1, t) = \text{north} \text{ and } \text{CarDirection}(c_2, t) = \text{south} \text{ and } \text{CarOnSegment}(c_1, t) \text{ and } \text{CarOnSegment}(c_2, t)
\]
Requirement Progression
add domain assumption

Diagram:

- Control Unit
  - NRpulse
  - NGpulse
  - SRpulse
  - SGpulse

- Light Unit
  - NRobserve
  - NGobserve
  - SRobserve
  - SGobserve

- Cars
  - CarDirection
  - CarOnSegment

Formulas:

- no t: time | some c1, c2 : Cars | CarDirection(c1, t) = north and CarDirection(c2, t) = south and CarOnSegment(c1, t) and CarOnSegment(c2, t)

- all t: time | ! NGobserve(t) => no c: Cars | CarDirection(c, t) = north and CarOnSegment(c, t)

- all t: time | ! SGobserve(t) => no c: Cars | CarDirection(c, t) = south and CarOnSegment(c, t)
Requirement Progression
rephrase requirement

no t: time \ \land \ \neg \text{NGobserve}(t) \land \neg \text{SGobserve}(t)

no t: \text{time} \ \land \ \exists c_1, c_2 : \text{Cars} \ \land \ \text{CarDirection}(c_1, t) = \text{north} \ \land \ \text{CarDirection}(c_2, t) = \text{south} \ \land \ \text{CarOnSegment}(c_1, t) \ \land \ \text{CarOnSegment}(c_2, t)

\forall t: \text{time} \ \land \ \neg \text{NGobserve}(t) \Rightarrow \nexists c: \text{Cars} \ \land \ \text{CarDirection}(c, t) = \text{north} \ \land \ \neg \text{CarOnSegment}(c, t)

\forall t: \text{time} \ \land \ \neg \text{SGobserve}(t) \Rightarrow \nexists c: \text{Cars} \ \land \ \text{CarDirection}(c, t) = \text{south} \ \land \ \neg \text{CarOnSegment}(c, t)
Requirement Progression implication holds

- Control Unit
  - NRpulse
  - NGpulse
  - SRpulse
  - SGpulse

- Light Unit
  - NRobserve
  - NGobserve
  - SRobserve
  - SGobserve

- Cars
  - CarDirection
  - CarOnSegment
  - NGobserve
  - SGobserve

- no t: time l
  - NGobserve(t) and
  - SGobserve(t)

- all t: time l
  - NGobserve(t) =>
    - no c: Cars l
    - CarDirection(c, t) = north
    and CarOnSegment(c, t)

- all t: time l
  - SGobserve(t) =>
    - no c: Cars l
    - CarDirection(c, t) = south
    and CarOnSegment(c, t)
Requirement Progression

push requirement

Diagram:

- Control Unit
  - NRpuls
  - NGpulse
  - SRpuls
  - SGpuls

- Light Unit
  - NRobserve
  - NGobserve
  - SRObservable
  - SGObserv

- Cars
  - CarDirection
  - CarOnSegment
  - NGoverflow
  - SGoverflow

Rules:

1. no t: time l
   - NGobserve(t) and SGobserve(t)

2. all t: time l ! NGobserve(t) →
   - no c: Cars l
     - CarDirection(c, t) = north
     - CarOnSegment(c, t)

3. all t: time l ! SGobserve(t) →
   - no c: Cars l
     - CarDirection(c, t) = south
     - CarOnSegment(c, t)
Requirement Progression
add domain assumption
Requirement Progression
rephrase requirement
Requirement Progression implication passes

Control Unit
- NRpulse
- NGpulse
- SRpulse
- SGpulse

Light Unit
- NGobserve
- SGobserve
- NGpulse
- SGpulse

Cars
- CarDirection
- CarOnSegment
- NGobserve
- SGobserve

Formulas:

For all time t:
- NGobserve(t) \iff \text{odd}(NGpulse, t)
- SGobserve(t) \iff \text{odd}(SGpulse, t)

For all time t:
- \text{no c}: Cars \implies \text{no c}: Cars

For all time t:
- CarDirection(c, t) = \text{north} \land \text{CarOnSegment}(c, t)
Requirement Progression

push requirement

Control Unit
- NGpulse
- SGPulse

Light Unit
- NGobserve
- SGPulse
- NGpulse

Cars
- CarDirection
- CarOnSegment
- NGobserve
- SGPulse

all t: time \[\text{NGobserve}(t) \iff \text{odd}(\text{NGpulse}, t) \text{ and } \text{SGobserve}(t) \iff \text{odd}(\text{SGpulse}, t)\]

all t: time \[\text{NGobserve}(t) \implies \text{no c: Cars} \]
- CarDirection(c, t) = north
- CarOnSegment(c, t)

all t: time \[\text{SGobserve}(t) \implies \text{no c: Cars} \]
- CarDirection(c, t) = south
- CarOnSegment(c, t)
Requirement Progression
der-to-end guarantee
Progression Toolkit

- Add Breadcrumb
- Rephrase Requirement
- Push Arc
- Split/Merge Arc
- Heuristics
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement
- Push Arc
- Split/Merge Arc
- Heuristics
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement new ∧ breadcrumb ⇒ old
- Push Arc
- Split/Merge Arc
- Heuristics
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement new ∧ breadcrumb ⇒ old
- Push Arc if phenomena are shared
- Split/Merge Arc
- Heuristics
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement new ∧ breadcrumb ⇒ old
- Push Arc if phenomena are shared
- Split/Merge Arc nothing else changes
- Heuristics
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement new ∧ breadcrumb ⇒ old
- Push Arc if phenomena are shared
- Split/Merge Arc nothing else changes
- Heuristics walk req. towards machine add / rephrase / push guided by informal “frame” push multiple arcs separately
Features

guided decomposition
  local reasoning, global guarantee
  result is structured argument

precise language
  unambiguous when dispatched
  automatic checks

explicit (vs implicit) domain assumptions
  “add what you need to make progress”
  tends to produce minimal assumptions
  focuses auditing and analysis
  reduces implementation bias
Voting Requirements

fidelity: count votes correctly

secrecy: protect anonymity

auditability: public can check fidelity
Voting Requirements

fidelity: count votes correctly
  • design analysis
  • requirement progression

secrecy: protect anonymity
  • knowledge/inference model
  • leverage fidelity argument

auditability: public can check fidelity
  • statistical audits
  • but which parts should be audited?
Electronic Voting
Prêt à Voter

- fidelity
- secrecy
- auditability
Fidelity
Electronic Voting
Prêt à Voter

✓ fidelity
☐ secrecy
☐ auditability
Modeling Secrecy Attacks
Modeling Secrecy Attacks

Information

initial
Modeling Secrecy Attacks
Modeling Secrecy Attacks

- Information
- Initial
- Incognito
- Inferences
Modeling Secrecy Attacks

- Information
- initial
- incognito

inferences
Modeling Secrecy Attacks
Modeling Secrecy Attacks
Modeling Secrecy Attacks
Modeling Secrecy Attacks

Information

extend fidelity model

sig Voter {
    intention: set Candidate,
    voterBallot: set Ballot,
}

sig Voter {
    intention: set Candidate,
    known_intention: Candidate -> Time,
    voterBallot: set Ballot,
    known_voterBallot: Ballot -> Time,
}
Modeling Secrecy Attacks

Initial Data

write constraints on first time step

//no telepathy
no known_intention.first

//tear-off receipt, alternate ballots
no known_ballotArrangement.first

//encrypted onions
no known_onionArrangement.first

//re-encryption steps
no known_mix.first

intention, mix, ballotArrangement, onionArrangement
Modeling Secrecy Attacks

Incognito Data

assert secret data is learned

Successful Attack:
The adversary (eventually) knows what ballot was given to some voter and the candidate indicated by that ballot.

some v: Voter |
some v.(known_voterBallot.last)
  .(known_ballotCandidate.last)
Modeling Secrecy Attacks

Inferences
derive from fidelity assumptions

Breadcrumb
Voters mark their ballots to match their intentions.

all v: RegisteredVoter | let b = v.voterBallot | b.ballotReceipt.receiptMarked
  .(b.ballotArrangement) = v.intention

Inference (Secrecy)
If you know a voter’s intention, then you can infer how that voter’s ballot is marked (and vice versa).
Modeling Secrecy Attacks

Analysis

pred attack [] {
    // restrictions on knowledge
    no known_intention.first
    no known_ballotArrangement.first
    no known_onionArrangement.first
    no known_mix.first

    // only learn via inferences
    explainAdditions[]

    // malicious goal
    some v: Voter |
        some v.(known_voterBallot.last)
        .(known_ballotCandidate.last)
}
run attack for 4
Electronic Voting
Prêt à Voter

✓ fidelity
✓ secrecy
✎ auditability
How to get auditability

without secrecy - easy
- reveal full fidelity argument (design)
- reveal full operation (implementation)
- e.g. reveal all votes cast

with secrecy - tricky
- cannot reveal all details of operation
- can perform statistical audits
- what to audit?
How to get auditability

assume fidelity
  breadcrumbs written Alloy model

assume secrecy
  extension of Alloy fidelity model

what should be audited?
  note set of hidden phenomena (secrecy)
  audit referencing assumptions (fidelity)
What to audit?

audit assumptions referencing non-initial information

intention, mix, ballotArrangement, onionArrangement
What to audit?

voters
all v: RegisteredVoter | let b = v.voterBallot |
  b.ballotReceipt.receiptMarked.(b.ballotArrangement) = v.intention

ballots
all b: Ballot |
  b.ballotArrangement = b.ballotReceipt.receiptOnion.onionArrangement

voting board
all input: mix.Record | let output = input.mix {
  input.receiptOnion.arrangement = output.recordArrangement
  input.receiptMarked = output.recordMarked
}
What to audit?

voters
   - "voters mark ballots according to intention"
   - user study on comprehension, execution

ballots
   all b: Ballot |
   b.ballotArrangement = b.ballotReceipt.receiptOnion.onionArrangement

voting board
   all input: mix.Record | let output = input.mix { 
   input.receiptOnion.arrangement = output.recordArrangement
   input.receiptMarked = output.recordMarked
   }
What to audit?

voters
- “voters mark ballots according to intention”
- user study on comprehension, execution

ballots
- “onion arrangements match their ballots”
- randomly sample, discard

voting board

```javascript
all input: mix.Record | let output = input.mix {
  input.receiptOnion.arrangement
  = output.recordArrangement
  input.receiptMarked = output.recordMarked
}
```
What to audit?

voters
- “voters mark ballots according to intention”
- user study on comprehension, execution

ballots
- “onion arrangements match their ballots”
- randomly sample, discard

voting board
- “re-encryption preserves ordering/marking”
- multiple re-encryption steps
- partial random reveal of each step
Fidelity Timetable

- **Fidelity**: 5 days, 130 lines Alloy
- **Secrecy**: 4 days, 1000 lines Alloy
- **Audit List**: 1 day, 0 lines Alloy

Graph:
- Interviews to understand system overview and needs: 4 days, 30 lines Alloy, 50 lines comments
- Interpret assumptions into cryptographic properties: 1 hour, 40 lines prose
- Cryptographic theorems: 0/? days
- Requirement progression: 1 day human, 5 seconds analysis, 100 lines Alloy, 150 lines comments
- Cryptographic proofs: 0/? days

The graph illustrates the stated properties on various levels: module, component, system, and context.
Secrecy Timetable

fidelity: 5 days
130 lines Alloy

secrecy: 4 days
1000 lines Alloy

audit list: 1 day
0 lines Alloy
Discoveries

dependability argument
- articulate, analyze, document
- separate system/crypto arguments
- confirmed fidelity, secrecy, auditability

build arguments in tandem: easier, thorough
- **fidelity** with req. prog.
- **secrecy** based on fidelity
- **audit list** read off of fidelity/secrecy

discoveries
- no need to absorb marking into onion
- do need alternate ballots in booth
Interests

link high level goals to low level specs
- talk to managers/users/customers
- talk to diff. types of engineers
- help articulate goals, capabilities
- document argument in usable form

interpreting technical info for user
- requirement progression guidance
- show secrecy attack traces
- role of constraint in model
- magic layout for Alloy
Questions are the enemy of the Daleks
Supplemental Slides
Providing Dependability

what must a dependability argument do?

- justify system requirement
  - cross-cutting, not monolithic

- justify component assumptions
  - hard evidence / clear sign-off

- connect requirement to assumptions
  - end-to-end argument
  - disciplined checked
Why “Dependability”?

avoids undesired connotations

reliability suggests...
- avoiding downtime

safety suggests
- avoiding human risk
- avoiding disaster

correctness suggests...
- absolutes (vs. reasonable confidence)
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement
  new ∧ breadcrumb ⇒ old
- Push Arc
  if phenomena are shared
- Split/Merge Arc
  nothing else changes
- Heuristics
  add / rephrase / push
  walk towards machine
  guided by informal “frame”
Add Breadcrumb
on phenomena of 1 domain

Rephrase Requirement
new ∧ breadcrumb ⇒ old

Push Arc
if phenomena are shared

Split/Merge Arc
nothing else changes

Heuristics
add / rephrase / push
walk towards machine
guided by informal “frame”
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement new \& breadcrumb \Rightarrow old
- Push Arc if phenomena are shared
- Split/Merge Arc nothing else changes
- Heuristics add / rephrase / push walk towards machine guided by informal “frame”
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement new \& breadcrumb \implies old
- Push Arc if phenomena are shared
- Split/Merge Arc nothing else changes
- Heuristics add / rephrase / push walk towards machine guided by informal “frame”
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement
  new ∧ breadcrumb ⇒ old
- Push Arc
  if phenomena are shared
- Split/Merge Arc
  nothing else changes
- Heuristics
  add / rephrase / push
  walk towards machine
  guided by informal “frame”
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement if phenomena are shared
- Push Arc if phenomena are shared
- Split/Merge Arc nothing else changes
- Heuristics add / rephrase / push walk towards machine guided by informal “frame”
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement: new ∧ breadcrumb ⇒ old
- Push Arc: if phenomena are shared
- Split/Merge Arc: nothing else changes
- Heuristics: add / rephrase / push walk towards machine guided by informal “frame”
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement new ∧ breadcrumb ⇒ old
- Push Arc if phenomena are shared
- Split/Merge Arc nothing else changes
- Heuristics add / rephrase / push walk towards machine guided by informal “frame”
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement new ∧ breadcrumb ⇒ old
- Push Arc if phenomena are shared
- Split/Merge Arc nothing else changes
- Heuristics add / rephrase / push walk towards machine guided by informal “frame”
Progression Toolkit

- **Add Breadcrumb** on phenomena of 1 domain
- **Rephrase Requirement**
  - new \( \land \) breadcrumb \( \Rightarrow \) old
- **Push Arc**
  - if phenomena are shared
- **Split/Merge Arc**
  - nothing else changes
- **Heuristics**
  - add / rephrase / push
  - walk towards machine
  - guided by informal “frame”
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement: new ∧ breadcrumb → old
- Push Arc if phenomena are shared
- Split/Merge Arc: nothing else changes
- Heuristics: add / rephrase / push walk towards machine guided by informal “frame”
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement new ∧ breadcrumb ⇒ old
- Push Arc if phenomena are shared
- Split/Merge Arc nothing else changes
- Heuristics add / rephrase / push walk towards machine guided by informal “frame”
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement if phenomena are shared
  new \land breadcrumb \Rightarrow old
- Push Arc nothing else changes
- Split/Merge Arc add / rephrase / push walk towards machine guided by informal “frame”
Progression Toolkit

- Add Breadcrumb on phenomena of 1 domain
- Rephrase Requirement new ∧ breadcrumb ⇒ old
- Push Arc if phenomena are shared
- Split/Merge Arc nothing else changes
- Heuristics add / rephrase / push walk towards machine guided by informal “frame”
Traffic Light: Alloy Description

sig Cars {
    onSeg: time,
    dir: Direction -> time
}
pred CarOnSegment(c: Cars, t: time) { t in c.onSeg }
fun CarDirection(c: Cars, t: time) : Direction { (c.dir).t }

abstract sig Direction {}
one sig north, south extends Direction {}
sig time {}

sig NGObs, SGObs in time {}
pred NGO(t: time) {t in NGObs}
pred SGO(t: time) {t in SGObs}
Traffic Light: Alloy Check

pred old() {
  no t: time | some c1, c2 : Cars |
  CarDirection(c1, t) = north and
  CarDirection(c2, t) = south and
  CarOnSegment(c1, t) and
  CarOnSegment(c2, t)
}

pred new() {
  no t: time |
  NGO(t) and
  SGO(t)
}

pred breadcrumb() {
  all t: time |
  ! NGO(t) =>
  (no c: Cars |
   CarDirection(c, t) = north and
   CarOnSegment(c, t))

  all t: time |
  ! SGO(t) =>
  (no c: Cars |
   CarDirection(c, t) = south and
   CarOnSegment(c, t))
}

assert valid {
  new() and breadcrumb() => old()
}
check valid for 10
## Alternate Approaches

<table>
<thead>
<tr>
<th></th>
<th>Explicit</th>
<th>Checked</th>
<th>Decomposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Communication</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>State Machine Models</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Automatic Logic Inference</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Patterns</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Requirement Progression</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
CDADs
Composite Dependability Argument Diagrams
Interpretation

stated as property on...

recast as properties on...

context

system

component

module

block

context

arguments

statements
Traffic Light

stated as property on...

- context
- system
- component
- module
- block

recast as properties on...

we know that cars do not collide because of our assumptions about the procedure specs
Traffic Light

stated as property on...

recast as properties on...

- cars do not collide
- car's obey lights
- lights obey signals sent
- signals sent in certain pattern

we know that cars will not collide because of our assumptions about the components

we know that signals are sent in a certain pattern because of our assumptions about the procedure specs
Program Analysis

stated as property on...

recast as properties on...

context

system

component

module

block

JML, Z, VDM, B

UML, OCL, Javadoc

Fürge, Karun, ESC, code review, algorithmic proof

design patterns, design review

Astree

code statements, abstracted calls
Target: Dependability

stated as property on...

recast as properties on...

context       block

component     module

system

RE

PA

Goal
Testing

stated as property on...

recast as properties on...

context

system

component

module

block

- unit tests
- regression tests, full component tests
- usability testing, integration testing
- deployment testing
Burr Proton Therapy Center

MGH BPTC facility
- treating cancer patients since 2002
- high energy proton beam
- irradiate tumor, low collateral dmg
- precise enough for brain/eye/child
BPTC Requirements

delivery
- intensity
- location
- patient

bookkeeping
- dose logging
- session logging
- privacy

operational
- throughput
- cost
- safe failure

Our Focus: Dose Delivery
The patient received a dose of radiation with the intensity recorded in the Rx database under that patient’s ID.
Problem Diagram

- **Therapist**
  - selection
  - nameInfo

- **Patient**
  - nameInfo
  - dose margin
  - dose
  - dose settings
  - interpretation

- **HW Beam Equipment**
  - dose

- **TM Treatment Manager**
  - queryDosesRequest
  - queryDosesResult
  - queryListRequest
  - queryListResult
  - readIDmsg
  - sendLISTmsg

- **GUI Interface**
  - selection
  - map
  - sendIDmsg
  - readLISTmsg

- **Messages on Network**
  - sendIDmsg
  - readLISTmsg

- **DB Prescription Database**
  - namesInfo
  - doses inactive
  - queryDosesRequest
  - queryDosesResult
  - queryListResult

The patient receives the dose that is associated with the patient's name in the prescription database.
The ID from the message received from the GUI is the same ID used to query the DB for dose information.

correctness
- the ID is read from the message and stored in the same global used to generate the DB dose query

separability
- that global is not overwritten before the query is generated
- permits analysis of less code
Correctness

examine designation

A message is received as a pair of parameters, data and arg. The data parameter contains a message which is an array of identifiers. The 0th slot of that array indicates the screen that was displayed when the message was generated. The 1th slot indicates the button that was pressed to trigger the message. The arg parameter is a lump of data containing state information about the gui. Part of that lump of data is the identity of the patient being treated.

data.data__msg.mixed_array_index[0] == SCR_A1_PATIENT_SELECTION && data.data__msg.mixed_array_index[1] == W_PATIENT_SELECT BTN => current_id_patient == arg.scrCrtPatientData.dbs_patient_type__id_patient

interpret into code terminology

data.data__msg.mixed_array_index[0] == SCR_A1_PATIENT_SELECTION && data.data__msg.mixed_array_index[1] == W_PATIENT_SELECT BTN => current_id_patient == arg.scrCrtPatientData.dbs_patient_type__id_patient

pass to Forge (Java API)

final ForgeExpression correct_result =
current_id_patient.eq(arg.join(scrCrtPatientData)
.join(dba_patient_type__id_patient))
.and(
  one.join(data.join(data__msg).join(mixed_array_index))
  .eq(W_PATIENT_SELECT BTN))
.and(
  zero.join(data.join(data__msg).join(mixed_array_index))
  .eq(SCR_A1_PATIENT_SELECTION));
Forge

automatic analysis
  relational claims (Alloy)
  counter-example traces
  iterative precondition discovery

  JForge, CForge
  manual translation to FIR

scalability: needs to add specs for
  library calls
  proprietary code
  cutoff points (reduce volume)
  reduce 2.5 Mloc → 1 Kloc
CDAD for TM

stated as property on...

designations, problem diagram

hospital needs

requirement progression

trace extraction

forge analysis

code statements, abstracted calls

specification for trace manager

specification for trace

module

component

system

context

block

recast as properties on...
Timetable

Total: 2 months (5 days/week)
With CForge: 5 weeks (est.)
Achievements

safety argument for critical property
- traceable end-to-end argument
- found & documented assumptions
- reasonable cost - est. 1 year more

current vulnerabilities
- sql injection (separability)
- message delays
- patient identification process

future vulnerabilities
- network (dropped, reordered msg)
- database (format, units)
- GUI generation (authentic i/o)
- code (structure, redundant, globals)
- new firing mode
Limitations

vulnerabilities not errors
- sometimes find errors in process
- limited by quality of assumption confirmation
- hard to assess confidence of human domains

requires skilled analysis & support
- support from domain specialists
- physicist, doctor, therapist, programmer, manager, operator (not patients)
- trouble with staff changes
- relational logic is easy...to me

code analysis requires abstraction to scale
- poorly structured code
- scaling limitations of analysis
Burr Proton Therapy Center

MGH BPTC facility
- treating cancer patients since 2002
- high energy proton beam
- irradiate tumor, low collateral dmg
- precise enough for brain/eye/child

![Graph showing photon and proton depth doses](image)
The patient receives the dose that is associated with the patient's name in the prescription database.
Identification Procedure

sharing assumptions: NameInfo
- unique name
- correctly read from tag

sharing assumptions: selection
- unique
- confirmed (mis-click)
- GUI SW error

domain assumptions
- memory while back turned
- alternate spelling
- duplicate entries on scroll list
CODE
TRANSLATION
BOOLEAN tpcrInSelectPatient(
    /* IN */     T_INT4 screenId,
    /* IN */     T_INT4 sizeofList,
    /* IN */     DATA_MSG_GROUP_ARRAY msgList,
    /* IN_OUT */  DBASCR_SCR_DATA_PTR_TYPE pscrData)
{
    T_INT4 num;
    /* Process list of message */
    for(num = 0; num < sizeofList; num++) {
        if(msgList[num].property != PDEF_CONTENTS_PROPERTY) {
            SW_ERROR_MSG(ERR_APP_WRONG_PROPERTY_TYPE);
            return FALSE;
        }
        switch(msgList[num].widgetGroupId) {
            case WG_PATIENT:
                switch(msgList[num].widgetId) {
                    case W_PATIENT_ID:
                        if(strlen(msgList[num].value) > DBA_PATIENT_ID_LEN) {
                            SW_ERROR_MSG(ERR_APP_WRONG_STR_VALUE);
                            return FALSE;
                        }
                        strcpy(pscrData -> scrCrtPatientData.id_patient, msgList[num].value);
                        break;
                    default: /* Error happens */
                        SW_ERROR_MSG(ERR_APP_WRONG_WIDGET_ID);
                        return FALSE;
                } /* widgetId */
            default: /* Error happens */
                SW_ERROR_MSG(ERR_APP_WRONG_WIDGET_GROUP_ID);
                return FALSE;
            } /* widgetGroupId */
        }
    } /* call events function correspond to "Select Patient" button */
    if(!eventsTPCRSelectPatient(screenId, pscrData)) { /* Error happens */
        TRACE_ERROR_MSG();
        return FALSE;
    }
    return TRUE;
}
BOOLEAN tpcrInSelectPatient(T_INT4 screenId, T_INT4 sizeofList, DATA_MSG_GROUP_ARRAY msgList, DBASCR_SCR_DATA_PTR_TYPE pscrData) {
    T_INT4 num;
    for(num = 0; num < sizeofList; num++) {
        if(msgList[num].property != PDEF_CONTENTS_PROPERTY) {
            SW_ERROR_MSG(ERR_APP_WRONG_PROPERTY_TYPE);
            return FALSE;
        }
        switch(msgList[num].widgetGroupId) {
            case WG_PATIENT:
                switch(msgList[num].widgetId) {
                    case W_PATIENT_ID:
                        if(strlen(msgList[num].value) > DBA_PATIENT_ID_LEN) {
                            SW_ERROR_MSG(ERR_APP_WRONG_STR_VALUE);
                            return FALSE;
                        }
                        strcpy(pscrData -> scrCrtPatientData.id_patient, msgList[num].value);
                        break;
                    default:
                        SW_ERROR_MSG(ERR_APP_WRONG_WIDGET_ID);
                        return FALSE;
                }
                break;
            default:
                SW_ERROR_MSG(ERR_APP_WRONG_WIDGET_GROUP_ID);
                return FALSE;
        }
    }
    if(!eventsTPCRSelectPatient(screenId, pscrData)) {
        TRACE_ERROR_MSG();
        return FALSE;
    }
    return TRUE;
}
void tpcrInSelectPatient(
    T_INT4 screenId,
    T_INT4 sizeofList,
    DATA_MSG_GROUP_ARRAY msgList,
    DBASCR_SCR_DATA_PTR_TYPE pscrData)
{
    T_INT4 num;
    for(num = 0; num < sizeofList; num++) {
        if (msgList[num].widgetGroupId = WG_PATIENT)
            if (msgList[num].widgetId = W_PATIENT_ID)
                if (strlen(msgList[num].value) > DBA_PATIENT_ID_LEN)
                    ERROR;
                else
                    strcpy(pscrData -> scrCrtPatientData.id_patient, msgList[num].value);
    } 
    eventsTPCRSelectPatient(screenId, pscrData);
}
void define__tpcrInSelectPatient() {
    //signature
    final LocalVariable
    screenId = program.newLocalVariable("screenId", T_INT4),
    sizeofList = program.newLocalVariable("sizeofList", T_INT4),
    msgList = program.newLocalVariable("msgList", DATA_MSG_GROUP_ARRAY),
    pscrData = program.newLocalVariable("pscrData", DBASCR_SCR_DATA_PTR_TYPE);
    tpcrInSelectPatient = program.newProcedure(
        "tpcrInSelectPatient",
        Arrays.<LocalVariable>asList(screenId, sizeofList, msgList, pscrData),
        Arrays.<LocalVariable>asList(pscrData));
    //body
    final LocalVariable num = program.newLocalVariable("num", T_INT4);
    final AssignStmt initializenum = tpcrInSelectPatient.newAssign(num, zero);
    final LocalVariable current = program.newLocalVariable("current", DATA_MSG_GROUP);
    final AssignStmt assign_current = tpcrInSelectPatient.newAssign(current, num.join(msgList.join(msg_grp_array_index)));
    final BranchNode loop_head_condition = tpcrInSelectPatient.newBranch(num.lt(sizeofList));
    final BranchNode check_widgetGroupId = tpcrInSelectPatient.newBranch(current.join(widgetGroupId_field).eq(WG_PATIENT));
    final BranchNode check_widgetId = tpcrInSelectPatient.newBranch(current.join(widgetId_field).eq(W_PATIENT_ID));
    final BranchNode check_length = tpcrInSelectPatient.newBranch(((current.join(value_field)).join(strlen)).gt(DBA_PATIENT_ID_LEN));
    final AssignStmt flag_error = tpcrInSelectPatient.newAssign(error_has_occurred, program.trueLiteral());
    final AssignStmt assign__id_patient = tpcrInSelectPatient.newAssign(dba_patient_type__id_patient, dba_patient_type__id_patient.override(
        (pscrData.join(scrCrtPatientData)).product(current.join(value_field))));
    final AssignStmt loop_end_num_increment = tpcrInSelectPatient.newAssign(num, num.plus(one));
    final CallStmt call_eventsTPCRSelectPatient = tpcrInSelectPatient.newCall(
        eventsTPCRSelectPatient,
        Arrays.<ForgeExpression>asList(screenId, pscrData),
        Arrays.<ForgeVariable>asList());
    //linkups
    initializenum.setEntry();
    initializenum.setNext(loop_head_condition);
    loop_head_condition.setThen(assign_current);
    assign_current.setNext(check_widgetGroupId);
    check_widgetGroupId.setThen(check_widgetId);
    check_widgetId.setThen(check_length);
    check_length.setThen(flag_error);
    flag_error.setNext(loop_end_num_increment);
    check_length.setElse(assign__id_patient);
    assign__id_patient.setNext(loop_end_num_increment);
    check_widgetGroupId.setElse(loop_end_num_increment);
    check_widgetId.setElse(loop_end_num_increment);
    loop_end_num_increment.setNext(loop_head_condition);
    loop_head_condition.setElse(call_eventsTPCRSelectPatient);
    call_eventsTPCRSelectPatient.setNext(tpcrInSelectPatient.exit());
}
proc tpcrInSelectPatient (screenId, sizeofList, msgList, pscrData) : (pscrData) {
Node55: num := 0 goto Node56
Node56: if (num < sizeofList) then Node57 else Node64
Node57: current := (num . (msgList . msg_grp_array_index)) goto Node58
Node58: if ((current . widgetGroupId_field) = WG_PATIENT) then Node59 else Node63
Node59: if ((current . widgetId_field) = W_PATIENT_ID) then Node60 else Node63
Node60: if (((current . value_field) . strlen) > DBA_PATIENT_ID_LEN) then Node61 else Node62
Node61: error_has_occurred := true goto Node63
Node63: num := (num plus 1) goto Node56
Node62: dba_patient_type__id_patient := (dba_patient_type__id_patient ++
   ((pscrData . scrCrtPatientData) -> (current . value_field))) goto Node63
Node64: eventsTPCRRselectPatient(screenId, pscrData) : () goto Node54
Node54: terminate
}
Fidelity

stated as property on...

context

system

component

module

recast as properties on...

fidelity requirement, designations, problem diagram

component assumptions

mathematical theorems

cryptographic proofs

requirement progression
Fidelity

The onion on each receipt faithfully encodes the arrangement of candidates on that receipt's ballot.

Every receipt corresponds to one record that indicates the same candidate.

Stack of Ballots --- receipts and onions --- Voting Board

ballots and receipt

Voters

Public Record

Each registered voter marks her ballot's receipt next to the candidate that she wants to win.

Thus each candidate's score is the number of registered voters who wanted that candidate to win.

Each candidate's score is the number of receipts marked in favor of that candidate.
all c: Candidate I

c.score = #(RegisteredVoter & intention.c)
Fidelity

Goal #1
all c: Candidate
  c.score = #(ballotCandidate.c & Ballot)
Fidelity
Fidelity
Fidelity
Fidelity

Goal #3
all c: Candidate
  c.score = #(|Receipt.mix & recordCandidate|, c)

all a b: Ballot I
  b.ballotReceipt := b'.ballotReceipt
all b: Ballot I
  b.ballotReceipt = b'.ballotReceipt
all r: Receipt I
  r.receiptCandidate = (r.receiptMarked).r.receiptOnion.onionArrangement
all c: Candidate I
  c.score = #(|Receipt.mix & recordCandidate|, c)
A Secrecy Attack

- Candidate
  - ballot arrangement
  - Position

- Candidate
  - given
  - Ballot
    - intention
  - indicate
  - Candidate
    - onion arrangement
  - Onion
    - annotated
    - onion arrangement
  - Candidate
    - Position

- Position
  - marked
  - Receipt
    - perforated
    - marked
    - Record
      - re-encrypted
      - record arrangement
      - Position
        - marked
A Secrecy Attack

- **Candidate**
  - ballot arrangement
  - given
  - perforated
  - marked

- **Position**
  - marked

- **Ballot**
  - intention
  - indicate

- **Voter**
  - given

- **Candidate**
  - onion arrangement

- **Receipt**
  - annotated
  - re-encrypted

- **Record**
  - record arrangement

- **Onion**
A Secrecy Attack

- Candidate
  - Position
- Ballot
  - given
  - intention
  - indicate
- Candidate
  - marked
  - re-encrypted
  - annotated
  - perforated
- Receipt
  - marked
- Onion
  - onion arrangement
- Record
  - marked
  - record arrangement
- Candidate
  - Position
  - marked
  - re-encrypted
  - annotated
A Secrecy Attack

- **Candidate** → **ballot arrangement** → **Voter** intention → **Candidate**
- **Candidate** → **Position**
- **Ballot** → **indicate** → **Candidate**
- **Position** → **marked** → **Receipt**
- **Receipt** → **annotated** → **Onion**
- **Onion** → **onion arrangement** → **Candidate**
- **Candidate** → **Position**
- **Position** → **marked** → **Record**
- **Record** → **record arrangement**
Modeling Secrecy Attacks

Voter
(Victim, known_RegisteredVoter, RegisteredVoter)

Ballot
bArrange: Position->Candidate
ballotCandidate: Candidate

Receipt
receiptCandidate: Candidate

Record
dArrange: Position->Candidate
kn_dArrange: Position->Candidate
recordCandidate: Candidate

Position

Onion
oArrange: Position->Candidate

onionArrangement_inference_3
(comingAttractions, pre)
candidate: Candidate
onion: Onion
position: Position
Modeling Secrecy Attacks
Modeling Secrecy Attacks
Modeling Secrecy Attacks
Auditability
Future Work

Requirement Progression
  tool support (automatic suggestions)

Code Analysis
  lower human cost, better scalability

Human Analysis
  systematic assessment of assumptions

Case Studies
  lightweight, automatic, lower confidence

CDAD
  denote confidence, cost
How to invest effort?

stated as property on...

recast as properties on...

context block module component system context

system

component

module

block

hazard analysis

requirement progression

automatic code analysis
Idea: Water Glass Model

Budget
Idea: Water Glass Model

- automatic code analysis
- requirement progression
- hazard analysis
Idea: Water Glass Model

- Minimum
- Automatic code analysis
- Requirement progression
- Hazard analysis
Idea: Water Glass Model

- Minimum
- Automatic code analysis
- Requirement progression
- Hazard analysis
Idea: Water Glass Model
wrong requirements
no substantiation
no link

minimum

- automatic code analysis
- no link
- hazard analysis
low budget, heavy weight techniques
high budget, lightweight techniques

minimum

informal review  patterned argument  hazard analysis
marginal returns

diminishing returns
high overhead
heavyweight
lightweight
Interests

diagrams & techniques to interpret technical information and guide analysis

doctoral (requirements engineering)
  requirement progression guidance
  show secrecy attack traces

masters (modeling)
  role of constraint in model

side projects (information layout)
  magic layout for Alloy
  information representation in games
Questions?
Building PFs