Building Dependability Arguments for Software-Intensive Systems

Irradiate!

Robert Seater
January 5th, 2009
How it works
How it works

- React Mode
- Cat open
- Cool closed
- Level = 30%
- Reaction Chamber
- Coolant
How it works
How it works

- React mode
- Cat open
- Cool open
- Level = 40%
- Reaction chamber
- Coolant
- Vent
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%
else if CAT closed, COOL open
    level @t = (level @t+1) -10%
else
    level @t = level @t+1

VENT

component needs attention => no release
release <=> level = 100%

ERROR SIGNAL => needs attention

CATALYST

COOLANT

REACTION CHAMBER
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
- level = 30\%
- REACTION CHAMBER
- ERROR SIGNAL \( <-> \) needs attention
- VENT
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
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
release ↔
level = 100%

VENT

component
needs attention
⇒ no release

ERROR SIGNAL ↔ needs attention

ERROR SIGNAL

CAT open

COOL closed

REACTION CHAMBER

level = 100%

CATALYST

COOLANT
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

level = 100%

release

release <=>
level = 100%

component
needs attention
=> no release

ERROR SIGNAL

CAT open

COOL closed

ERROR SIGNAL

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

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

- Control Unit
- Light Unit
- Cars

signal pulses → Light Unit ← light observations
Traffic Light Requirement

- Control Unit
  - signal pulses to Light Unit
- Light Unit
  - light observations to Cars
- Safety Requirement
  - cars do not collide
  - car directions
  - car locations
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 \ And \ Light Unit Assumptions
  \ And \ 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  south red signal pulse  Controller  North Unit
Light Unit Behavior

South Unit -> Controller: north green signal pulse

Controller -> North Unit
Light Unit Behavior

South Unit → Controller → North Unit

south red signal pulse
Light Unit Behavior

South Unit  →  Controller  →  North Unit

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

- Control Unit: The control unit will send a certain pattern of red and green signals...
- Light Unit: ...and since the signal pulses to the light determine if cars observe green lights...
- Cars: ...and since cars are only in the intersection if they observe a green light in their direction...
- Cars: ...then cars moving in opposite directions will never be in the intersection at the same time...
Requirement Progression

formal requirement

Control Unit
- NRpulse
- NGpulse
- SRpulse
- SGpulse

Light Unit

Cars
- NRobserve
- NGobserve
- SRobserve
- SGobserve

CarDirection
CarOnSegment

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

Control Unit

Light Unit

Cars

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

- Control Unit
  - NRpulse
  - NGpulse
  - SRpulse
  - SGpulse

- Light Unit
  - NRobserve
  - NGobserve
  - SRobserve
  - SGobserve

- Cars
  - CarDirection
  - CarOnSegment
  - NGobserve
  - SGobserve

- no t: time ⊢ NGobserve(t) and SGobserve(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 implication holds

no t: time \! \! NGobserve(t) and \! \! SGobserve(t)

all t: time \! \! NGobserve(t) \implies \! \! no c: Cars \! \! CarDirection(c, t) = north and CarOnSegment(c, t)

all t: time \! \! SGobserve(t) \implies \! \! no c: Cars \! \! CarDirection(c, t) = south and CarOnSegment(c, t)
Requirement Progression
push requirement

- **Control Unit**
  - NRpulse
  - NGpulse
  - SRpulse
  - SGpulse

- **Light Unit**
  - NRobserve
  - NGobserve
  - SRobserve
  - SGobserve

- **Cars**
  - CarDirection
  - CarOnSegment
  - NGobserve
  - SGobserve

- **Event**
  - no t: time I
    - NGobserve(t) and
    - SGobserve(t)

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

- **Rule**
  - all t: time I ! SGobserve(t) =>
    - no c: Cars I
  - CarDirection(c, t) = south
  - and CarOnSegment(c, t)
Requirement Progression
add domain assumption

- Control Unit
  - NRpulse
  - NGpulse
  - SRpulse
  - SGpulse

- Light Unit
  - NRobserve
  - NGobserve
  - NGpulse
  - SGobserve

- Cars
  - CarDirection
  - CarOnSegment
  - NGobserve
  - SGobserve

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

- all t: time !
  - NGobserve(t) ↔ odd(NGpulse, t) and SGobserve(t) ↔ odd(SGpulse, 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
Requirement Progression implication passes

Control Unit → Light Unit → Cars

- Control Unit:
  - NRpulse
  - NGpulse
  - SRpulse
  - SGpulse

- Light Unit:
  - NRobserve
  - NGobserve
  - SRobserves
  - SGobserve
  - NGpulse
  - SGpulse
  - NGobserve(t) ↔ odd(NGpulse, t) and SGobserve(t) ↔ odd(SGpulse, t)

- Cars:
  - CarDirection
  - CarOnSegment
  - NGobserve
  - SGobserve
  - all t: time \( t \) ! NGobserve(t) \( \Rightarrow \) no c: Cars
  - CarDirection(c, t) = north and CarOnSegment(c, t)
  - all t: time \( t \) ! SGobserve(t) \( \Rightarrow \) no c: Cars
  - CarDirection(c, t) = south and CarOnSegment(c, t)

- no t: time \( t \)
  - odd(NGpulse, t) and odd(SGpulse, t)
Requirement Progression
push requirement

Control Unit

Light Unit

Cars

NGpulse
SGpulse

NRpulse
NGpulse
SRpulse
SGpulse

NRobserve
NGobserve
SRobserve
SGobserve

CarDirection
CarOnSegment
NGobserve
SGobserve

all t: time \! NGobserve(t) \Rightarrow no c: Cars
CarDirection(c, t) = north and CarOnSegment(c, t)

all t: time \! SGobserve(t) \Rightarrow no c: Cars
CarDirection(c, t) = south and CarOnSegment(c, t)

no t: time \!
odd(NGpulse, t) and odd(SGpulse, t)
Requirement Progression
end-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 \(\land\) breadcrumb \(\Rightarrow\) 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
CDADs
Composite Dependability Argument Diagrams

- narrow
- broad
- shallow
- deep

- stated
- established
- argument
Interpretation

stated as property on...

recast as properties on...

context

system

component

module

block

statements

arguments

context
Traffic Light

stated as property on...

recast as properties on...

context

block

module

component

system

context

system

component

software procedure specs

module

block

cars do not collide

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

software procedure specs

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
Requirements Engineering

stated as property on...

- STAMP diagram, customer interview
- DFDs, OPM, problem diagram, use cases
- hazard analysis, paper prototyping
- state charts
- fault trees, event trees, FMEA, FMECA
- KAOS

recast as properties on...

- block
- module
- component
- system
- context
Program Analysis

stated as property on...

recast as properties on...

context

system

component

module

block

code statements, abstracted calls

JML, Z, VDM, B

UML, OCL, Javadoc

Forge, Karun, ESC, code review, algorithmic proof

design patterns, design review

Astree
Target: Dependability

stated as property on...

recast as properties on...

context

system

component

module

block

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
Composition

stated as property on...

recast as properties on...

context

system

component

module

block

context

system

component

module

block
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.
The patient receives the dose that is associated with the patient's name in the prescription database.
Argument Diagram

1. Patient is correctly selected
   namesInfo = selection

2a. Interpretation reflects msg
    map = readLISTmsg

2b. Id is interpreted and sent
    map.selection = sendIDmsg

3. Messages are transmitted authentically
   sendLISTmsg = readLISTmsg
   sendDmsg = sendIDmsg

4a. List info is sent
    queryListResult = sendLISTmsg

4b. Id from message is sent to db
    queryDosesRequest = readDmsg

4c. Queried dose is used to set equipment
    queryDosesResult = queryDosesResult

5a. Queries reflect db
    queryListResult = namesInfo - inactive
    queryDosesResult = queryDosesRequest.doses

0. Dose delivery
   (namesInfo, nameInfo).doses = dose

HW Beam Equipment

TM Treatment Manager

DB Prescription Database
Code Analysis of TM

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...

specification for treatment manager
specification for trace
code statements, abstracted calls
forge analysis
trace extraction
requirement progression
hazard analysis
designations, problem diagram
hospital needs
context
system
component
module
block

recast as properties on...
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
Electronic Voting

Prêt à Voter

Peter Ryan

http://www.pretavoter.com/
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

stated as property on...

context | block | module | component | system | context
---|---|---|---|---|---
context |  |  |  |  |  
system |  |  |  | fidelity requirement, designations, problem diagram |  
component |  |   | component assumptions |  |  
module |  | mathematical theorems |  | requirement progression |  
cryptographic proofs |  |  |  |  |
Fidelity

Goal #0
all c: Candidate
\text{score} = \#(RegisteredVoter \land \text{intention}, c)
Electronic Voting
Prêt à Voter

✓ fidelity
촉 secrecy
毫无 auditability
Modeling Secrecy Attacks
Modeling Secrecy Attacks
Modeling Secrecy Attacks

Information

initial

incognito
Modeling Secrecy Attacks

Information

initial

incognito

inferences
Modeling Secrecy Attacks
Modeling Secrecy Attacks
Modeling Secrecy Attacks

Information

initial

incognito

inferences
Modeling Secrecy Attacks

Diagram showing the flow of information from initial to incognito stages, with inferences depicted.
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
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.

\[
\text{all } v: \text{RegisteredVoter} \mid \text{let } b = v.\text{voterBallot} \mid \\
\text{b.ballotReceipt.receiptMarked} \mid \\
(b.\text{ballotArrangement}) = v.\text{intention}
\]

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
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
 ø "voters mark ballots according to intention"
 ø 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
Secrecy Timetable

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

Stated as property on:

- Building model framework, articulating attack goal: 2 days, 250 lines Alloy framework
- Interpreting assumptions into cryptographic properties: 1 hour human time, 250 lines Alloy checks
- Alloy model: 1000 total lines
- Deriving inferences from assumptions: 2 days, 500 lines Alloy inferences
- Alloy Analysis: 1 second
Discoveries

dependability argument
- articulate, analyze, document
- separate system/crypto arguments
- identify appropriate level of detail
- confirmed fidelity, secrecy, auditability

build arguments in tandem
- **fidelity** more thorough with req. prog.
- **secrecy** easier if based on fidelity
- **audit** more thorough if based on fid/sec

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

system

component

module

block

hazard analysis

requirement progression

automatic code analysis
Idea: Water Glass Model
Idea: Water Glass Model

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

- minimum
- automatic code analysis
- requirement progression
- hazard analysis

[Image of a water glass and a pitcher]
Idea: Water Glass Model

- minimum
- automatic code analysis
- requirement progression
- hazard analysis
Idea: Water Glass Model
wrong requirements
no substantiation
no link
low budget, heavyweight techniques

minimum

- theorem proving
- requirement progression
- hazard analysis
high budget, lightweight techniques
marginal returns

diminishing returns
high overhead
heavyweight
lightweight
Questions are the enemy of the Daleks
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)
Requirement Progression
informal requirement

Control Unit → signal pulses → Light Unit → light observations → Cars

Safety:
the road segment never contains a northward moving car and a southward moving car at the same time

car directions
car locations
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”
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 $\land$ 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 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 if phenomena are shared
- Push Arc nothing else changes
- Split/Merge Arc walk towards machine guided by informal “frame”
Progression Toolkit

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

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”
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>
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

Photon & Proton Depth Doses

![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 psrCrtData)
{
    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(psrCrtData->scrCrtPatientData.id_patient, msgList[num].value);
                        break;
                    default: /* Error happens */
                        SW_ERROR_MSG(ERR_APP_WRONG_WIDGET_ID);
                        return FALSE;
                } /* widgetId */
                break;
            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, psrCrtData)) { /* 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; } 
                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 BranchNode loop_head_condition = tpcrInSelectPatient.newBranch(num.lt(sizeofList));
    final AssignStmt assign_current = tpcrInSelectPatient.newAssign(current, num.join(msgList.join(msg_grp_array_index)));
    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))));

    //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);
    check_length.setElse(assign__id_patient);
    assign__id_patient.setNext(loop_end_num_increment);
    loop_end_num_increment.setNext(loop_head_condition);
    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: eventsTPCRSelectPatient(screenId, pscrData) : () goto Node54
    Node54: terminate
}
Operational Overview

<table>
<thead>
<tr>
<th>Arrangement</th>
<th>Receipt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Candidate E</td>
<td></td>
</tr>
<tr>
<td>Candidate D</td>
<td></td>
</tr>
<tr>
<td>Candidate F</td>
<td></td>
</tr>
<tr>
<td>Candidate B</td>
<td>✔️</td>
</tr>
<tr>
<td>Candidate C</td>
<td></td>
</tr>
<tr>
<td>Candidate A</td>
<td></td>
</tr>
</tbody>
</table>

Marking

Onion
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

Voting Board

Voters

Public Record

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

Receives and onions

Votes

Records

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
Fidelity

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

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

all b. Ballot - Voter.voterBallot | b.ballotReceipt.receiptMarked
all v. RegisteredVoter - one v.voterBallot
all v. Voter - RegisteredVoter | no v.voterBallot
all b. Ballot - one voterBallot.b
all v. RegisteredVoter | let b = v.voterBallot |
| b.ballotReceipt.receiptMarked.b.ballotArrangement = v.intention
all b. Ballot | b.ballotCandidate = (b.ballotReceipt).receiptMarked.b.ballotArrangement
Fidelity
Fidelity
Fidelity
A Secrecy Attack

Candidate

Voter

intention

given

Ballot

Candidate

indicate

Candidate

ballot arrangement

Candidate

Position

Perforated

Receipt

Candidate

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position

Candidate

Candidate

Position
A Secrecy Attack

Candidate → Ballot
  given

Candidate → Position

Ballot → Receipt
  perforated

Receipt → Record
  marked

Record → Onion
  re-encrypted

Onion → Candidate
  onion arrangement

Candidate → Position

Candidate → Position

Candidate → Position

Candidate → Position
A Secrecy Attack

Candidate

ballot arrangement

Position

Voter

intention

Candidate

indicating

Ballot

Perforated

Candidate

Onion

onion arrangement

Candidate

Position

Receipt

annotated

Candidate

Record

marked

Record

marked

Position

Position
A Secrecy Attack
Modeling Secrecy Attacks
Modeling Secrecy Attacks
Modeling Secrecy Attacks
Auditability

first re-encryption

receipt 1
receipt 2
receipt 3
receipt 4
receipt 5
receipt 6

second re-encryption

receipt 3
receipt 1
receipt 2
receipt 5
receipt 6
receipt 4
receipt 6

receipt 3
receipt 2
receipt 1
receipt 5
receipt 4
receipt 6
Lessons Learnt

systematic / discipline
- Req. Prog. saves time, deepens understanding
- designations and P.F. building guide interviews

scalability
- Problem Diagrams remain simple (4–8 domains)
- automatic analysis vs. manual review

CDAD
- STAMP promising link to broaden/integrate
- reveals gaps between academic fields

elicitation
- breadcrumbs good medium of comm.
- Alloy excerpts readable by engineers
Building PFs