how to prevent disasters

Daniel Jackson, MIT
Siren//NL, Veldhoven • November 2, 2010
a civil engineering disaster
kansas city hyatt regency, 1981

New York Times
the design

beam supports one walkway

illustrations from Matthys Levy and Mario Salvadori, Why Buildings Fall Down
how it failed

beam supports two walkways

as designed  as implemented  what happened
therac 25

no argument for success
› AECL fault tree (1983) did not include software
› \( P(\text{computer selects wrong energy}) = 10^{-11} \)

hard to extract any lessons
› Leveson & Turner (1993): so many flaws, nothing clear

so doomed to fail again
› 17 deaths from similar machine in Panama (2001)
› 621 target/dose/patient errors (2001-9, NY state)

my conclusions

civil engineers
› argue why structure should stand
› failure occurs when argument is flawed

software engineers
› build and hope for the best
› when failure occurs, no story
› can’t assign blame or learn for future
WHY SOFTWARE STANDS UP
The Strength of Architecture

MARIO SALVADORI
a new approach

write down critical properties $R$

write down domain assumptions $A$

design a specification $S$

check that $A \wedge S \implies R$

reduce goal

rely on more

fix design

no

yes

build machine $M$

check that $M \Rightarrow S$

DEPENDABILITY CASE:
claimed properties
assumptions
design & specs
correctness argument
the door interlock problem
problem: design an interlock

a textbook problem

› see, eg, *Engineering a Safer World* [Leveson, 2010]
actually, a real problem

The Worlds First Microwave Test Oven

Here's a picture of the world's first commercial microwave during its first field test. I am on the left, my brother on the right. We used to defeat the door interlock and point it at the end of the countertop where we left a plate of eggs. They exploded like little hand grenades. Drove my mom nuts!

http://www.thescubalady.com/Keith%20Lamb%20History.htm

Statistics indicate that five to ten arc-flash accidents that involve a fatality or serious injury to an employee occur every day in the United States.

http://www.iaei.org/magazine/?p=1163
step 1: requirement

- Safe: touch event does not occur in state Live
- Safe: no touching live power source

Diagram:
- Operators
- Safe
- Power Source
- Touch
step 2: domain assumptions

Operators

- Safe
  - Safe: touch event does not occur in state Live

Exposed

- Door
  - open, close
  - when close occurs, Exposed becomes false

- Sensor
  - open, close
  - when open occurs, Closed becomes false

- Power Source
  - Live
    - when off occurs, Live becomes false

- Switch
  - no touch unless Exposed is true
step 3: machine specification

- Operators
- Door: open, close
  - when close occurs, Exposed becomes false
  - when open occurs, Closed becomes false
- Sensor: open, close
- Exposed: no touch unless Exposed is true
- Power Source
  - when off occurs, Live becomes false
- Live
- Switch
- Controller
  - every step, send off if Closed became false
  - send on only when Closed is true
- Safe: touch event does not occur in state Live
  - when close occurs, Exposed becomes false
  - when open occurs, Closed becomes false
  - when off occurs, Live becomes false
  - every step, send off if Closed became false
  - send on only when Closed is true
step 4: checking the system argument

domain assumptions $\land$ machine spec $\implies$ requirement

one sig Sensor extends Domain {
  Closed: set Time
}

one sig PowerSource extends Domain {
  Exposed, Live: set Time
}

sig Open extends Event {
  not Sensor.Closed.after
}

one sig Controller extends Domain {
  all t: Time - (first + last) |
  not Sensor.Closed.at [t]
  and Sensor.Closed.at [t.prev]
  implies Off.happensAt [t]
}

one sig Safe extends Requirement {
  all t: Touch |
  not PowerSource.Live.before [t]
}
counterexample!

problem: forgot initial conditions
solution: record them

one sig PowerSource extends Domain {
Exposed, Live: set Time }
{
not Live.initially
not Exposed.initially
}
counterexample again!

problem: controller turns power off too late

solution: new domain assumption

```
sig Touch extends Event {} {
  PowerSource.Exposed.before
  no o: Open | this.follows [o]
}
```
no more counterexamples

Alloy’s analysis is
› fully automatic
› large bounded space
› here, analyzed $2^{366}$ cases
**summary**

- **Safe**: Touch event does not occur in state **Live**.
- **Door**: When close occurs, **Exposed** becomes false.
- **Exposed**: Exposed is initially false.
- **Power Source**: When off occurs, **Live** becomes false.
- **Live**: Live is initially false.
- **Closed**: When open occurs, Closed becomes false.
- **Controller**: Every step, send off if Closed became false.
- **Switch**: When on occurs, Live becomes false.
- **Sensor**: No touch unless Exposed is true.
- **Operators**: Touch does not follow within 1 step of open.
- **Controller**: Send on only when Closed is true.
dependability cases we’ve worked on

Burr Proton Therapy Center
› correct dose [Robert Seater]
› emergency stop [with Andrew Rae]
› treatment door interlock [Eunsuk Kang, Joe Near, Aleks Milicevic]

Voting systems
› Pret a Voter [Robert Seater]
› Scantegrity [Eunsuk Kang]

Tokeneer
› ongoing analysis [Eunsuk Kang]
tokeneer

- commissioned by NSA as exemplar
- built by Praxis using Z and SPARK-Ada
- not just open source!
Problem diagram

- Users
  - Access
- Enclave
  - Access
  - Accessible, Blocked
- Card Reader
  - Insert, Remove
  - Read Token
- Fingerprint Reader
  - Attach, Detach
  - Read Fingerprint
- Door
  - Lock, Unlock
  - Locked, Unlocked
- Controller
  - Lock, Unlock
- Latch

- Privilege
  - Access

- Access Enclave
  - Have Privilege
analyzing the design

what Praxis did
› formal spec in Z (about 120 pages); informal reasoning
› code verification with SPARK-Ada

defects found to date
› 5 code-level defects
› requirements issues (using Alloy for test case generation) [Aydal & Woodcock 2009]
› no defects yet found in design

what we’re doing
› translating design to Alloy (about 1000 lines so far)
› automatic analysis: design ∧ assumptions ⇒ security
sample argument fragments

```plaintext
sig DoorLatchAlarm {
    currentTime : Time,
    currentDoor : Door,
    currentLatch : Latch,
    doorAlarm : Alarm,
    latchTimeout : Time,
    alarmTimeout : Time
}

-- latch is locked when timed out
currentState = Locked iff gte[currentTime, latchTimeout]

-- door alarm goes off when the door is open but the latch is locked
doorAlarm = Alarming iff
    (currentDoor = Open and
     currentState = Locked and
     gte[currentTime, alarmTimeout])
```

```plaintext
-- property 2 : unlock at allowed time (pg. 10, Doc 40_4)
assert UnlockAtAllowedTime {
    all s : Step |
    let s' = s.next,
    ut = IDStation.userToken.s,
    config = IDStation.config.s,
    curr = IDStation.time.s |
    -- if the latch is unlocked, then
    (some w, w' : ControlledWorld | latchUnlocked[w, w', s']) implies {
        -- the user must have a token that has "recently" been validated for an entry
        let token = ut.currentUserToken.t {
            validToken[token] |
            some recentTime : timesRecentTo[curr, config.tokenRemovalDuration] |
            recentTime in
            config.entryPeriod[token.privCert.role][token.privCert.clearance]
    }
}
```
results so far

bug in security property
› if door is opened, user must hold token with recently validated fingerprint or valid authorization certificate

bug in spec for UnlockDoor
› timer not checked if token withdrawn after timeout
proton therapy
proton therapy treatment room
correct dose requirement
correct dose case

extraction of models
› Alloy models of messaging infrastructure
› C code translated to Java, then to Alloy using Forge

resulting insights
› very long message delay might cause bad dose
› patient identification relies on distinct patient names
› SQL injection attack vulnerability
Door interlock requirement

Opening door causes beam stop being inserted

- **Door Safety Requirement**
- **Beam** Stop signal causes beam stop being inserted
- **Nozzle**

- **Safety Control Unit**
- DoorOpen signal

- **DataDaq**
  - frameOut causes (msg : RTWorks.msgs)
    - msg.type = RTMsgTypeMap[frameOut]
    - msg.dest = RTMsgDestMap[frameOut]

- **TCP/IP**
  - frameln

- **RPC**
  - evlReport
  - BCU.beamInsert causes BeamStop signal

- **Beam Control Unit**
  - insertBeamStop
  - (req : reqs) causes call: dest.calls
    - req.dest = BCU and
    - req.type = InhibitBeamStop

- **Treatment Control Unit**
  - inhibitBeam
  - (cb : callbacks) causes (msg : RTWorksmsgs)
    - msg.dest = BeamManager
    - msg.type = ACT_INHIBIT_BEAM

- **System Manager**
  - logEvent

- **Treatment Manager**
  - callback(rtdaqinDoorOpen)

- **RPC (evlReport)**
  - (msg : msgs) causes (cb : dest.callbacks)
    - dest = msg.dest
    - cb = CallbackMap[msg.type]
door interlock case

high level analysis in Alloy
› by modelling each component
› simple chain of events

code analysis
› to identify side conditions
› to extract control paths
› but hard due to missing code

approach
› lightweight extraction of control flow
› abstract interpretation of state
› user provides specs for library calls
tracing call paths

tool and analysis by Aleks Millicevic
tracing calls within a component
results so far

entanglement
› door safety entangled with logging
› if logging fails, safety action is aborted
› (but hardware safety system...)

how to cheat
identifying the trusted base

touch does not follow within 1 step of open

when close occurs, Exposed becomes false

exposed is initially false

when open occurs, Closed becomes false

Closed

Sensor

Controller

Operators

Power Source

Switch

Door

Exposed

Live

Off

touch does not occur in state Live

when off occurs, Live becomes false

Live is initially false

on, off

Controller

keep touch unless Exposed is true

Exposed is initially false

when close occurs, Exposed becomes false

no touch unless Exposed is true

Exposed is initially false

touch does not follow within 1 step of open

every step, send off if Closed became false

send on only when Closed is true

Safe: touch event does not occur in state Live
reducing the trusted base

simpler design ⇒ simpler argument
analysis with trusted bases

one sig Sensor extends Domain {
    Closed: set Time
}
sig Open extends Event { } {
    Sensor in OK implies not Sensor.Closed.after
}

one sig Controller extends Domain { } {
    this in OK implies
    all t: Time - (first + last) |
        not Sensor.Closed.at [t]
        and Sensor.Closed.at [t.prev]
        implies Off.happensAt [t]
}

one sig Safe extends Requirement {} {
    this in OK iff
    all t: Touch | not PowerSource.Live.before [t]
    trustedBase = Switch + Controller + Sensor + Door + Operators
}

assert BaseSufficient {
    all r: Requirement | r.trustedBase in OK implies r in OK
}
reducing the trusted base: examples
designing emergency stop

pendant with emergency stop button
existing design

Emergency Stop works

Hand Pendant
Event Registration
Beam Block
UI Agent
Event Queue
Controller
File System
Operating System
alarm clock

... It’s only job is to wake you up in the morning, and I believe you'll find that it does it’s job perfectly.

Most other alarm clock applications choose to play the alarms/music via iTunes (via AppleScript). I deliberately decided against this... Consider...

- The alarm is set to play a specific song, but the **song was deleted**.
- The alarm is set to play a specific playlist, but you renamed the playlist, or deleted it.
- The alarm is set to play a **radio station**, but the internet is down.
- iTunes was recently upgraded, and requires you to **reagree to the license** next time you launch it.
- The alarm application launches it for the alarm...
- You had iTunes set to play to your airTunes speakers, but you left your airport card turned off.
- You had the iTunes **preference panel open**. (Which prevents AppleScript from working)
- You had a "Get Info" panel open. (Which also prevents AppleScript from working)

alarm clock

example: voting

All cast ballots are counted

Scanner computes tally based on ballots

Voters accurately record choice on a ballot

Optical Scanner computes tally based on records

Reports tally from scanner to public

Election Official

Scantegrity design, relying on voters and 3rd party tabulators

Gives one ballot per voter

Gives one ballot per voter

Voters checks their receipts

Voters checks their receipts

Auditor checks independent tallies

Auditor checks independent tallies

Independent tallies match

Independent tallies match

Tabulator

Tabulator

Standard design, relying on scanner

Election Official

Optical Scanner

Optical Scanner

Check-in Desk

Check-in Desk
conclusions
what's typically (not) done

-触碰在打开后的第一步骤内不适用

-现象未指定
-领域假设未记录

-当关闭时，露出变为false
-初始化未记录

-在打开时，关闭变为false
-没有触碰除非露出为true

-控制器
-参考规格

-每一步，发送off如果关闭变为false
-发送on只在关闭为true时

-安全：触碰事件不发生在Live状态
-无触碰除非露出为true

-关键特性未明确
-无系统分析
observations

on dependability cases
› if you can’t say why it works, it probably doesn’t

on design
› a principle: design for simple argument

on formal methods
› two benefits: clarity of requirements, mechanical checks

on cost
› key to low cost is upfront investment, non-uniformity
too hard to argue, unsafe to build

The direction and amount of the complicated strains throughout the trussing [would] become **incalculable** as far as all practical purposes are concerned...

Stephenson, explaining why he rejected a suspension design
a research question

- Touch does not follow within 1 step of open
- When close occurs, Exposed becomes false
- When close occurs, Closed becomes true
- When open occurs, Closed becomes false
- Sensor
- When close occurs, Closed becomes true
- Power Source
  - When off occurs, Live becomes false
- Switch
  - Live is initially false
- Live
- Controller
  - Every step, send off if Closed became false
  - Send on only when Closed is true
  - Send on when Closed becomes true

Safe: Touch event does not occur in state Live

-redundant' properties
should they be included?
if so, how?
acknowledgments

joint work with my students
› Eunsuk Kang, Joe Near, Aleks Millicevic

phenomenology
› Michael Jackson, *Problem Frames* (2001)

dependability cases study
› ‘Sufficient Evidence’ (NAS, 2007)

related work by many
› van Lamsweerde, Kelly, etc (goal structuring)
› Rushby, Knight, Bloomfield (assurance cases)
› ...

support from NSF, Northrop Grumman, Mass General
A paper about this approach

A Direct Path to Dependable Software, CACM, March 2009
wordle thanks to Jonathan Feinberg, IBM Research, Cambridge
backup slides
**designations**

- **events**
  - open: operator opens door fully or partially
  - close: operator closes door fully
  - touch: operator touches power
  - on: controller issues command to switch to turn on
  - off: controller issues command to switch to turn off

- **states**
  - Exposed: power source is exposed
  - Live: power in live state
  - Closed: sensor is in state that reports door closed
what if analysis finds no flaws?

informal problems
› wrong domain assumption
› missing phenomena or interactions
› wrong or badly expressed requirement

formal problems
› scope not large enough
› inconsistent axiomatization
› analysis tool is broken
› ... or system is actually safe
module domains

abstract sig Domain {}

abstract sig Property {}

abstract sig Requirement extends Property {
    trustedBase: set Domain
}

sig OK in Domain + Property {}

assert BaseSufficient {
    all r: Requirement | r.trustedBase in OK implies r in OK
}
module events

open util/ordering[Time] as time

sig Time {}

abstract sig Event {
    pre, post: Time
}

fact Traces {
    all t: Time - last | some e: Event | e.pre = t and e.post = t.next
    all t: Time - last | lone e: Event | e.pre = t
}
examining side conditions
on software risks

“We have become dangerously dependent on large software systems whose behavior is not well understood and which often fail in unpredictable ways.”

President's Information Technology Advisory Committee, 1999

“The most likely way for the world to be destroyed, most experts agree, is by accident. That’s where we come in. We’re computer professionals. We cause accidents.”

Nathaniel Borenstein, Programming as if People Mattered, Princeton University Press, 1991
on accidents

“Accidents are signals sent from deep within the system about the vulnerability and potential for disaster that lie within”

Richard Cook and Michael O’Connor
Thinking About Accidents And Systems (2005)
“There probably isn’t a best way to build the system, or even any major part of it; much more important is to avoid choosing a terrible way, and to have a clear division of responsibilities among the parts.”

Butler Lampson
Hints for computer system design (1983)