A Dependability Case Language for a Radiation Therapy System

Michael Ernst, Dan Grossman, Jon Jacky, Calvin Loncaric, Stuart Pernsteiner, Zachary Tatlock, Emina Torlak, Xi Wang

University of Washington
end-to-end verification for safety critical systems
Memory Model

COMPCERT

Quark

IronClad

frenetic

sel4

SUPPORTED BY

TLA+

SLAM

Z3

ROSETTE

alloy

Isabelle

KODKOD

Dafny
Dependability cases

Integrate diverse sources of evidence

check interfaces of design, testing, proof, review

Argue end-to-end claim based on evidence

show claim holds across all layers of a system

Focus on physical system properties

eases validation and focuses verification effort
Dependability case engineering
Many large software systems display fragility or a lack of dependability caused by inattention to details at various stages of development (e.g., missing data, undocumented assumptions, lack of testing), resulting in a failure to catch errors. This technical note explains how to create a dependability case for a system that helps identify and keep track of such details. A dependability case is defined here as a structured argument providing evidence that a system meets its specified dependability requirements. The technical note describes how to structure the argument and present evidence to support it. A sample problem is presented, as well as issues raised by that problem and future goals.
Many large software systems display fragility or a lack of dependability caused by inattention to details at various stages of development (e.g., missing data, undocumented assumptions, lack of testing), resulting in a failure to catch errors. This technical note explains how to create a dependability case for a system that helps identify and keep track of such details. A dependability case is defined here as a structured argument providing evidence that a system meets its specified dependability requirements. The technical note describes how to structure the argument and present evidence to support it. A sample problem is presented, as well as issues raised by that problem and future goals.

Many large software systems display fragility or a lack of dependability caused by inattention to details at various stages of development (e.g., missing data, undocumented assumptions, lack of testing), resulting in a failure to catch errors. This technical note explains how to
### Dependability case engineering

Many large software systems display fragility or a lack of dependability caused by insufficient documentation, missing data, undocumented assumptions, lack of testing, resulting in a failure to catch errors. This technical note explains how to create a dependability case for a system that helps identify and keep track of such details. A dependability case is defined here as a structured argument providing evidence that a system meets its specified dependability requirements. The technical note describes how to structure the argument and present evidence to support it. A sample problem is presented, as well as issues raised by that problem and future goals.

<table>
<thead>
<tr>
<th>Difficult to develop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficult to check</td>
</tr>
<tr>
<td>Difficult to maintain</td>
</tr>
</tbody>
</table>
Many large software systems display fragility or a lack of dependability caused by inattention to details at various stages of development (e.g., missing data, undocumented assumptions, lack of testing), resulting in a failure to catch errors. This technical note explains how to create a dependability case for a system that helps identify and keep track of such details. A dependability case is defined here as a structured argument providing evidence that a system meets its specified dependability requirements. The technical note describes how to structure the argument and present evidence to support it. A sample problem is presented, as well as issues raised by that problem and future goals.

Dependability case engineering

Difficult to develop

Difficult to check

Difficult to maintain

SUPPORTED BY

?
Formal

End-to-end

Checkable Dep. Cases

Dependability Cases

COMPCERT
Quark
frenetic >> seL4

IronClad
Developing a Dependability Case Language
Developing a Dependability Case Language

Move from specific to general

avoid attempt to design “silver bullet”
Developing a Dependability Case Language

Move from specific to general

avoid attempt to design “silver bullet”
Developing a Dependability Case Language

1. Target specific system
2. Develop dep. claims
Developing a Dependability Case Language

1. Target specific system
2. Develop dep. claims
Developing a Dependability Case Language

1. Target specific system
2. Develop dep. claims
Developing a Dependability Case Language

1. Target specific system
2. Develop dep. claims
3. Gather evidence

Claims
Developing a Dependability Case Language

1. Target specific system
2. Develop dep. claims
3. Gather evidence
Developing a Dependability Case Language

1. Target specific system
2. Develop dep. claims
3. Gather evidence
Developing a Dependability Case Language

1. Target specific system
2. Develop dep. claims
3. Gather evidence
4. Design + build DCL
Developing a Dependability Case Language

1. Target specific system
2. Develop dep. claims
3. Gather evidence
4. Design + build DCL
Developing a Dependability Case Language

DCL

Claims

Design
Application
Platform
Env

1. Target specific system
2. Develop dep. claims
3. Gather evidence
4. Design + build DCL

Find general tradeoffs and patterns

*make simple easy and hard possible*

Impact real-world projects

*bring current PL tech to the trenches*
Developing a Dependability Case Language

1. Target specific system
2. Develop dep. claims
3. Gather evidence
4. Design + build DCL

Find general tradeoffs and patterns
make simple easy and hard possible

Impact real-world projects
bring current PL tech to the trenches
an end-to-end dependability case for CNTS
Checking safety of CNTS

Clinical Neutron Therapy System (CNTS) at UW

- 30 years of incident-free service.
- Controlled by custom software, built by CNTS engineering staff.
- Third generation of Therapy Control software now being built.
Checking safety of CNTS

Clinical Neutron Therapy System (CNTS) at UW

- Prescription
- Sensors
- Therapy Control Software
- Beam, motors, etc.
Checking safety of CNTS

Experimental Physics and Industrial Control System (EPICS) Dataflow Language

- Prescription
- Sensors

Therapy Control Software

Beam, motors, etc.
The Maximize Severity attribute is one of NMS (Non-Maximize Severity), MS (Maximize Severity), MSS (Maximize Status and Severity) or MSI (Maximize Severity if Invalid). It determines whether alarm severity is propagated across links. If the attribute is MSI only a severity of INVALID_ALARM is propagated; settings of MS or MSS propagate all alarms that are more severe than the record's current severity. For input links the alarm severity of the record referred to by the link is propagated to the record containing the link. For output links the alarm severity of the record containing the link is propagated to the record referred to by the link. If the severity is changed the associated alarm status is set to LINK_ALARM, except if the attribute is MSS when the alarm status will be copied along with the severity.
Checking safety of CNTS

Prescription → Sensors

Therapy Control Software

Beam, motors, etc.
Checking safety of CNTS

An end-to-end property that spans the entire system, not just software.

**CNTS Couch Safety Property:**

The beam will turn off if the couch rotation angle moves out of tolerances during treatment and the operator has not issued the manual override command.
An informal dependability case for couch safety

couch rotates out of tolerances and no manual override => beam shuts off

- Couch
- Treatment Motion Controller
- Therapy Control Software
- Programmable Logic Controller
- Hardwired Safety Interlock System

Ethernet Network
An informal dependability case for couch safety

couch rotates out of tolerances and no manual override => beam shuts off

Couch

TMC measures OOT rotation -> Treatment Motion Controller

TC receives OOT rotation and no manual override -> Therapy Control Software

TC sets Therapy Sum interlock -> Programmable Logic Controller

PLC disables Therapy Sum relay => beam shuts off

Hardwired Safety Interlock System
A formal dependability case for couch safety

couch rotates out of tolerances and no manual override => beam shuts off

PLC disables Therapy Sum relay => beam shuts off

evidence[“63c8d380", PLC_Analysis, ..., Proof] =>
all relayState: plc.relay2754 & RelayOpen |
one coilState: plc.sentMsgs & relayState.^next |
coilState.coilNumber = Coil1623
coilState.coilValue = False

all r: Couch.rotation |
(properties and
  r.angle not in Prescription.tolerance and
no Event.GantryCouch_Turntable_Override) =>
some Beam.state & BeamOff
Generating **evidence** for couch safety

A solver-aided verifier for the subset of EPICS used in CNTS.
Checking couch safety

Dependability Case Complier (DCC)

Expert Review Validator
EPICS Linter
EPICS Verifier
PLC Analyzer
EPICS-PLC Signal Tracer
Alloy Analyzer

Counterexample or bounded proof
Deep analysis with <2000 LOC of tool code ...

TC receives OOT rotation and no manual override => TC sets Therapy Sum interlock

Found a bug in the Therapy Control software (preventing beam shut off), masked by a bug in the EPICS runtime!

Therapy Control Software

EPICS Verifier

concrete counterexample
Thanks!

Recent Verification Successes

Dependability Cases

End-to-end

Formal