

An Artificial Science for System Value Engineering and Assurance

Chong Tang, Kevin Sullivan, Ke Dou, Koleman Nix
Department of Computer Science
University of Virginia



May 12, 2015

Problems

Weak engineering foundations

- ▶ Weak understanding of system qualities
- ▶ Weak ability to engineer system qualities
- ▶ Poor understanding of relationships among qualities
- ▶ Lacking frameworks for reasoning about tradeoffs
- ▶ Weak ability to manage qualities across lifecycle

These engineering problems are in turn rooted in weak science.

Problems

Weak scientific foundations

- ▶ Research has often been informal and imprecise
- ▶ Theories expressed in natural language, tables, graphics
- ▶ No use of math, computational, and logical notations

Consequently we also lack foundations for automated tools for assisting with such issues.

Purpose

To provide a framework for:

- ▶ Communicating, reasoning about system qualities
- ▶ Reasoning about tradeoffs among system qualities
- ▶ Building formal languages to express qualities
- ▶ Formalizing theories with formal notations
- ▶ Providing automated tools to assist with the above

State of The Art

Ross's Semantic Approach (?, ?)

- ▶ **Problem:**

No precise understanding of particular system properties

- ▶ **Key Idea:**

A semantic approach for defining change-related ility terms

- ▶ **Main Contributions:**

- ▶ *Informal grammar* for changeability requirements
- ▶ Rules for *classifying statements* by *ility*
- ▶ Providing *semantics* to ility terms

Ross's semantic basis approach ¹[illegible]

Perturbation	Context	Phase	Agent	Inst* (optional)				Mechanism		Outcome				Abstraction	Reaction	Span	Cost	Details
				Nature	Parameter	Origin	Destination	Aspect	Mechanism	Effect	Parameter	Origin	Destination					
shift		opt							same	"Value"		few						By Label
disturbance		opt							same	"Value"		few						Value Robustness
shift		opt							same	"Value"		few						Value Survivability
shift		opt		not same					same			few						Robustness
shift		opt		not same					same			few						Active Robustness
shift		opt		same					same			few						Passive Robustness
shift		opt	none	same		few	few		same	level		few	form	system				Classical/Passive Robustness
disturbance		opt							same			few						Survivability
									not same									Changeability
shift	general	inter-LC		not same					not same					architecture				Evolvability
			internal	not same					not same									Adaptability
			external	not same					not same									Flexibility
				not same					not same	level								Scalability
				not same					not same	set								Mobility
		opt	either	not same					increase	set								Extensibility
				not same					not same									Agility
				not same					not same									Reactivity
		opt		Threaten per	one	one	form	operations	not same	"Link per"			form		sooner			Form Recognoizability
		opt		Threaten per	one	one	operations	operations	not same	"Order per"			operations					Operational Recognoizability
		opt		same	one	one	form/ops	not same	set			few/many	function					Functional Versatility
		opt		same	one	one	form/mot	not same	set			few/many	operations					Operational Versatility
				same	one	one			not same	set		form						Substitutability

Figure: Ross's prescriptive semantic basis for change-type ilites

Ross's semantic basis approach

► **Pros:**

Defining change-related ilities requirements statements

► **Cons:**

Informal, not computable, hard to evaluate and evolve

State of The Art

Boehm's top-down Taxonomy (?, ?)

► Problem:

System designs are deficient in balancing system ilities

► Key Ideas:

- Defining language grammer for full range of ilities
- Balancing ility values for the system's stakeholders

► Main Contributions:

- Proposing a stakeholder-value based property hierarchy
- An ontology for reasoning about a system's ilities
- Studied Synergies and Conflicts among key properties

Boehm's top-down Taxonomy ²

Stakeholder Value-Based QA Ends	Contributing QA Means
Mission Effectiveness	Stakeholders-satisfactory balance of Physical Capability, Cyber Capability, Human Usability, Speed, Endurability, Maneuverability, Accuracy, Impact, Scalability, Versatility, Interoperability
Resource Utilization	Cost, Duration, Key Personnel, Other Scarce Resources; Manufacturability, Sustainability
Dependability	Security, Safety, Reliability, Maintainability, Availability, Survivability, Robustness
Flexibility	Modifiability, Tailorability, Adaptability
Composite QAs	
Affordability	Mission Effectiveness, Resource Utilization
Resilience	Dependability, Flexibility

Figure: Stakeholder-value based property means-ends hierarchy

²Figure from (?, ?)

Boehm's top-down Taxonomy

► Pros:

- Clarifying the nature of system ilities
- Reasoning about the tradeoffs among ilities
- Addressing stakeholder value conflicts

► Cons:

Informal, difficult to validate, hard to apply

State of The Art

Assurance Cases

- ▶ Claim - Assertion about key requirements and properties
- ▶ Evidence
 - ▶ Testing, Proofs, Process and people, Review and analyses
- ▶ Argument - How the evidences support the claims
 - ▶ Inference rules: deterministic, probabilistic, qualitative
- ▶ Inductive reasoning
 - ▶ Providing evidence, not proof that the claim is certain

Assurance Cases ³

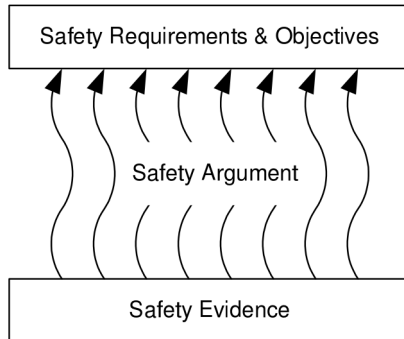


Figure: The relationship among safety case elements

³Figure from (?, ?)

State of The Art

Kelly's Goal Structuring Notation (?, ?)

► Problem:

Safety arguments are often poorly communicated

► Key Idea:

Develop safety cases in a reader-friendly manner

► Main Contributions:

- Using graphical notations to annotate the assurance cases
- Applying *inductive* argumentation to safety cases

Kelly's Goal Structuring Notation

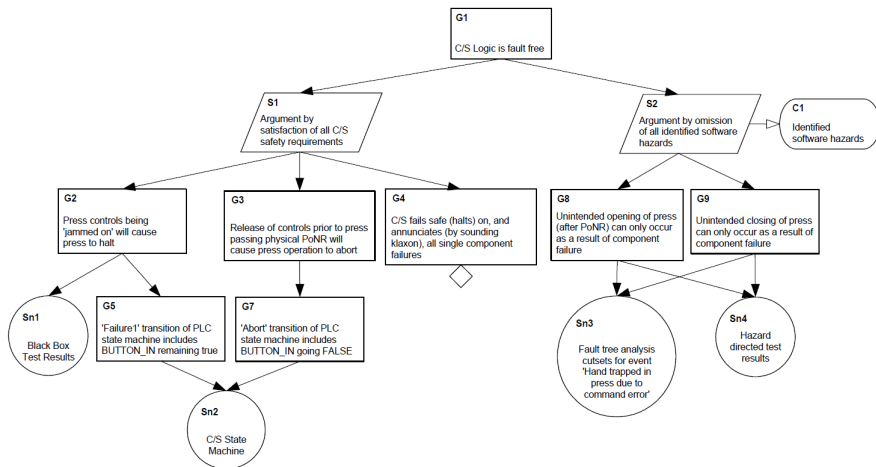


Figure: Example GSN (Figure from (?, ?))

Kelly's GSN safety argument notation

- **Pros:**

Facilitate comprehension and communication of arguments

- **Cons:**

Informal, syntax rules are defined in prose text, not scale

State of The Art

Rushby's Theory (?, ?)

► Problem:

Increasing confidence in the soundness of a given case

► Key Ideas:

- Applying formalism to safety cases
- Eliminating logic doubt and focusing on epistemic logic

► Main Contributions:

- Formalizing parts of a safety argument into deductive logic
- Providing mechanized support for assurance case argument
- Helping engineers focus on evidence instead of argument

Rushby's Theory

- **Pros:**

Improving efficiency and cost of safety argument checking

- **Cons:**

No empirical evidence

State of The Art

Knight's Assurance Based Development (?, ?)

► Problem:

Assurance cases often fail to guide developers' decisions

► Key Idea:

Co-developing the software system and its assurance case

► Main Contributions:

- Integrating assurance into development process.
- Assurance requirements drive development decisions

Knight's Assurance Based Development

- **Pros:**

Detecting the assurance difficulties from the earliest stages

- **Cons:**

Hard to validate that their approach is optimal

State of The Art

Basir's Automatically Generated Argument (?, ?)

- ▶ **Problem:**

Formal proofs are complex and machine-oriented

- ▶ **Key Idea:**

Automatically generating a safety argument by converting natural deduction style proofs

- ▶ **Main Contributions:**

- ▶ helps human understand the formal proofs

Basir's Automatically Generated Argument

► Pros:

Providing easier-to-understand proofs

► Cons:

- No benefit over an hand-generated, informal argument
- Far from satisfactory as the proofs contain too many details

State of The Art

Bosch's Mobile Service Oriented Architectures (?, ?)

- **Problem:**

It's hard to achieve success in realizing mobile services

- **Key Idea:**

Defining the architecture drivers that make success

- **Main Contributions:**

- Identified the goals for mobile service oriented architectures
- Identified ilities that influence the success of mobile services
- Predicted future trends of mobile service

State of The Art

Lundberg's Architecture Design Guidelines (?, ?)

► **Problem:**

There are conflicts between modifiability and performance

► **Key Idea:**

Providing guidelines in software architecture design

► **Main Contributions:**

- A taxonomy for performance and modifiability related QA
- Four software architecture design evaluation approaches
- Four architecture design transformation strategies
- Eight guidelines in software architecture design

Lundberg's Architecture Design Guidelines

► Pros:

- Revealed the relationships among architecture, quality attributes, and implementation
- The guidelines are extracted from real industry experience

► Cons:

- Only focus on performance and modifiability
- Such studies may not fit domains other than software design

State of The Art

Knight's Success Arguments (?, ?)

► Problem:

Failure rate of software development efforts is high

► Key Idea:

Defining success argument to establish confidence

► Main Contributions:

- Structuring and documenting the argument
- Recording the argument and exposing it to examinations

Knight's Success Arguments

► Pros:

- Helps structure the reasoning and expose it to criticism
- Helps explain the evidence to the reviewers

► Cons:

- Informal, Hard to validate

Our approach

- ▶ Combining Bosch's innovation experiment systems theory
- ▶ Using rigorous formal specification and software synthesis
- ▶ Refining and expressing quality theories using Coq
- ▶ Building web-based tools to implement the theory concepts
- ▶ Driving theory testing, evolution, and validation with tools

Top-Most System Value – Satisfactory

```

Class Satisfactory (System: Set) (Stakeholder: Set) (Context: Set) := {
  system: System

  ; mission_effective: System → Stakeholder → Context → Prop
  ; resource_utilization : System → Context → Prop
  ; dependable : System → Context → Prop
  ; flexible : System → Context → Prop

  ; ME: MissionEffective System Stakeholder Context system mission_effective
  ; RU: ResourceUtilization System Context system resource_utilization
  ; DP: Dependable System Context system dependable
  ; FL: Flexible System Context system flexible
  ; Affordable: Affordable System Stakeholder Context system ME RU
  ; Resilient: Resilient System Context system DP FL
}.

```

Mission Effectiveness in QA Taxonomy [Boehm, to app]

Mission Effectiveness: a System has achieved a
 Stakeholders-satisfactory balance of
 Physical Capability, Cyber Capability, Human Usability,
 Speed, Endurability, Maneuverability, Accuracy, Impact,
 Scalability, Versatility, and Interoperability.

Mission Effective

```

Inductive MissionEffective (System: Set) (Stakeholder: Set) (Context: Set) (sys: System)
  (mission_effective: System → Stakeholder → Context → Prop)
  (physical_capable: System → Stakeholder → Context → Prop)
  (cyber_capable: System → Stakeholder → Context → Prop)
  (human_usable: System → Stakeholder → Context → Prop)
  (speed: System → Stakeholder → Context → Prop)
  (endurable: System → Stakeholder → Context → Prop)
  .....
  : Prop :=

mk_mission_eff:
  PhysicalCapable System Stakeholder Context sys physical_capable →
  CyberCapable System Stakeholder Context sys cyber_capable →
  HumanUsable System Stakeholder Context sys human_usable →
  Speed System Stakeholder Context sys speed →
  Endurable System Stakeholder Context sys durable →
  Maneuverable System Stakeholder Context sys maneuverable →
  .....
  MissionEffective System Stakeholder Context sys mission_effective physical_capable
cyber_capable human_usable speed durable maneuverable accurate impact scalable versatile
interoperable.

```

Physical Capable

Inductive *PhysicalCapable* (*System*: **Set**) (*Stakeholder*: **Set**)
 (*Context*: **Set**) (*sys*: *System*)
 (*physical_capable*: *System* \rightarrow *Stakeholder* \rightarrow *Context* \rightarrow **Prop**)
 : **Prop** :=
mk_physical_capable:
 (\forall *cx*: *Context*, \forall *sh*: *Stakeholder*, *physical_capable sys sh cx*) \rightarrow
PhysicalCapable System Stakeholder Context sys physical_capable.

Example – Smart Home

Our Contributions

- ▶ A parameterizable hierarchy of qualities and relationships
- ▶ Quality-specific languages for expressing requirements
- ▶ Integration of the distinct, previously conflicting theories.
- ▶ Web-based software implementations of the theory concepts
- ▶ An approach for theory testing, evolution, and validation

The overall contribution of this work is a novel, rigorous, and promising new approach to developing, promulgating, testing, evolving, and validating the scientific theory that is needed to underpin rigorous new approaches to comprehensive system quality engineering.

Why do we think it will work?

- ▶ Replaces vague prose with *verifiable propositions*
- ▶ Every proposition has corresponding *assurance case*
- ▶ Practitioners never have to see formal specifications
- ▶ Web-based tools provide for *broad accessibility*
- ▶ Evolution of theory driven by *feedback from use*
- ▶ Social process of learning, testing, *theory validation*

Assessment

- Todo

Results and insights

- To be added

Conclusion

- To be added

Bibliography