# 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 14, 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 exress qualities
- Formalizing theories with formal notations
- Providing automated tools to assist with the above

# State of The Art

## Ross's Semantic Approach (Ross et al., 2012)

- **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]

**Prescriptive Semantic Basis for Change-type Ilities**

In response to "perturbation" in "context", desire "agent" to make some "change" in "system" that is "valuable"

| Perturbation | Context | Phase | Agent | Impetus Change | | | | | Mech | Outcome Change | | | | | System | Valuable* (this category is not complete) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

In response to "perturbation" in "context" during "phase" desire "agent" to make some "nature" impetus to the system "parameter" from "origin(s)" to "destination(s)" in the "aspect" using "mechanism" in order to have an "effect" to the outcome "parameter" from "origin(s)" to "destination(s)" in the "aspect" of the "abstraction" that are valuable with respect to thresholds in "reaction", "span", "cost" and "benefits"

| | | | | Impetus* (optional) | | | | | Mech | Outcome | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Perturbation | Context | Phase | Agent | Nature | Parameter | Origin | Destination | Aspect | Mechanism | Effect | Parameter | Origin | Destination | Aspect | Abstraction | Reaction | Span | Cost | Benefit |
| optional | circumstantial: required; general: optional | null | optional | null | required | optional | optional | null* (this is implied by "parameter") | Optional | null | required | optional | optional | null* | optional | required | required | required | required |
| "name" | "name(s)" | | "name(s)" | | "parameter" | "state(s)" | "state(s)" | | "name" | | "parameter" | "state(s)" | "state(s)" | | "name" | "threshold w/units" | "threshold w/units" | "threshold w/units" | "threshold w/units" |
| none | circumstantial | pre-ops | none | decrease | level | one | one | form | | decrease | level | one | one | form | architecture | sooner | shorter | less | more |
| disturbance | general | ops | internal | same | set | few | few | function | | same | set | few | few | function | design | later | longer | same | same |
| shift | <empty> | inter-LC | external | increase | <empty> | many | many | operations | | increase | <empty> | many | many | operations | system | always | same | more | less |
| <empty> | | <empty> | either | not-same | | <empty> | <empty> | <empty> | | not-same | | <empty> | <empty> | <empty> | <empty> | <empty> | <empty> | <empty> | <empty> |
| | | | <empty> | <empty> | | | | | | <empty> | | | | | | | | | |

| | | | | Impetus* (optional) | | | | | Mech | Outcome | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Perturbation | Context | Phase | Agent | Nature | Parameter | Origin | Destination | Aspect | Mechanism | Effect | Parameter | Origin | Destination | Aspect | Abstraction | Reaction | Span | Cost | Benefit | Ility Label |
| shift | | ops | | | | | | | | same | "Value" | | few | | | | | | | Value Robustness |
| disturbance | | ops | | | | | | | | same | "Value" | | few | | | | | | | Value Survivability |
| shift | | ops | | | | | | | | same | | | few | | | | | | | Robustness |
| shift | | ops | | not-same | | | | | | same | | | few | | | | | | | Active Robustness |
| shift | | ops | | same | | few | few | | | same | | | few | | | | | | | Passive Robustness |
| shift | | ops | none | same | | few | few | | | same | level | | few | form | system | | | | | Classical Passive Robustness |
| disturbance | | ops | | | | | | | | same | | | few | | | | | | | Survivability |
| | | | either | not-same | | | | | | 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 | | | | | | | | | Modifiability |
| | | ops | either | not-same | | | | | | increase | set | | | | | | | | | Extensibility |
| | | | | not-same | | | | | | not-same | | | | | | | shorter | | | Agility |
| | | | | not-same | | | | | | not-same | | | | | | sooner | | | | Reactivity |
| | | ops | | same | "Element set" | one | one | form | | not-same | "Link set" | | | form | | | | | | Form Reconfigurability |
| | | ops | | same | "Element set" | one | one | operations | | not-same | "Order set" | | | operations | | | | | | Operational Reconfigurability |
| | | ops | | same | | one | one | form/ops | | not-same | set | | few/many | function | | | | | | Functional Versatility |
| | | ops | | same | | one | one | form/fnct | | not-same | set | | few/many | operations | | | | | | Operational Versatility |
| | | ops | | same | | one | one | fnct/ops | | not-same | set | | few/many | form | | | | | | Substitutability |

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

[1] Figures from (Ross et al., 2012)

# 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 (Boehm & Kukreja, 2015)

- **Problem:**
  System designs are deficient in balancing system ilites
- **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

[2] Figure from (Boehm & Kukreja, 2015)

# 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]

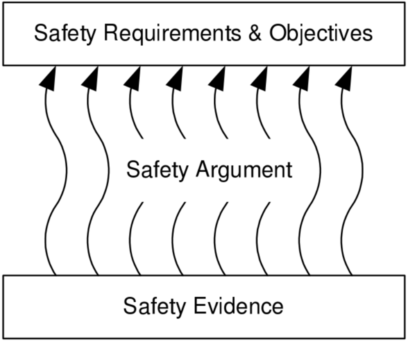


Figure: The relationship among safety case elements

[3] Figure from (Kelly & Weaver, 2004)

# State of The Art

Kelly's Goal Structuring Notation (Kelly & Weaver, 2004)

- **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

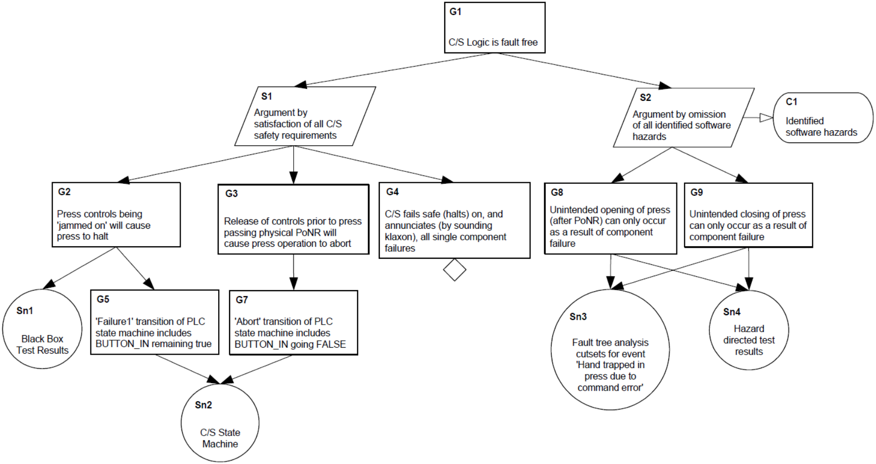


Figure: Example GSN (Figure from (Kelly & Weaver, 2004))

# 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 (Rushby, 2010)

- **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 mechnized 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 (Graydon, Knight, & Strunk, 2007)

- **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 (Basir, Denney, & Fischer, 2009)

- **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 (van Gurp, Karhinen, & Bosch, 2006)

- **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 (Lundberg, Bosch, Hggander, & Bengtsson, 1999)

- **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 (Graydon & Knight, 2008)

- **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
- Integrating Boehm's theory and Ross's approach
- 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

# Framework Architecture

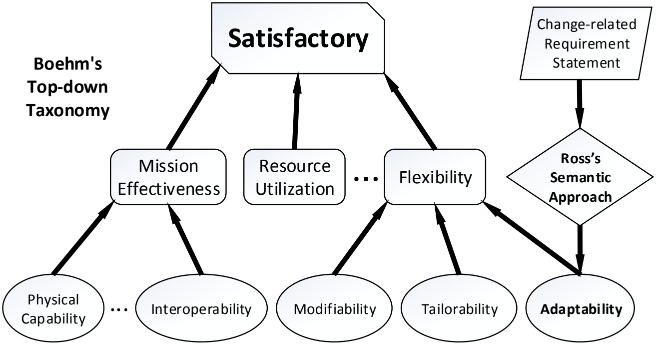


Figure: The overall architecture of our framework

# Top-Most System Value – Satisfactory

Class **Satisfactory** ($System$: Set) ($Stakeholder$: Set) ($Context$: Set) := {
sys: $System$

; physicalCapability : $System \to Stakeholder \to Context \to$ Prop
; cyberCapability : $System \to Stakeholder \to Context \to$ Prop
; humanUsability : $System \to Stakeholder \to Context \to$ Prop
......
; adaptability : $System \to Context \to$ Prop

; me: **MissionEffective** $System$ $Stakeholder$ $Context$ sys physicalCapability cyberCapability humanUsability speed endurability maneuverability accuracy impact scalability versability interoperability
; ru: **ResourceUtilization** $System$ $Context$ sys cost duration keyPersonnel otherScareResources manufacturability sustainability
; dp: **Dependable** $System$ $Context$ sys security safety reliability maintainability availability survivability robustness
; fl: **Flexible** $System$ $Context$ sys modifiability tailorability adaptability

# *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.

# Second-Level Property – Mission Effective

Inductive ***MissionEffective*** $(System\colon \texttt{Set})$ $(Stakeholder\colon \texttt{Set})$ $(Context\colon \texttt{Set})$ $(sys\colon System)$
$(physical\_capable\colon System \rightarrow Stakeholder \rightarrow Context \rightarrow \texttt{Prop})$
$(cyber\_capable\colon System \rightarrow Stakeholder \rightarrow Context \rightarrow \texttt{Prop})$
$(human\_usable\colon System \rightarrow Stakeholder \rightarrow Context \rightarrow \texttt{Prop})$
$(speed\colon System \rightarrow Stakeholder \rightarrow Context \rightarrow \texttt{Prop})$
$(endurable\colon System \rightarrow Stakeholder \rightarrow Context \rightarrow \texttt{Prop})$
......
$(interoperable\colon System \rightarrow Stakeholder \rightarrow \texttt{Context} \rightarrow \texttt{Prop})$
$: \texttt{Prop} :=$

*mk_mission_eff* :

***PhysicalCapable*** *System Stakeholder Context sys physical_capable* $\rightarrow$
***CyberCapable*** *System Stakeholder Context sys cyber_capable* $\rightarrow$
***HumanUsable*** *System Stakeholder Context sys human_usable* $\rightarrow$
***Speed*** *System Stakeholder Context sys speed* $\rightarrow$
***Endurable*** *System Stakeholder Context sys endurable* $\rightarrow$
***Maneuverable*** *System Stakeholder Context sys maneuverable* $\rightarrow$
......
***MissionEffective*** *System Stakeholder Context sys mission_effective physical_capable cyber_capable human_usable speed endurable maneuverable accurate impact scalable versatile interoperable.*

# Second-Level Property – Flexible

```
Inductive Flexible (System: Set) (Context: Set) (sys: System)
                   (flexible: System → Context → Prop)
                   (modifiable: System → Context → Prop)
                   (tailorable: System → Context → Prop)
                   (adaptable: System → Context → Prop)
                   : Prop :=
  mk_flexibility:
  Modifiable System Context sys modifiable →
  Tailorable System Context sys tailorable →
  Adaptable System Context sys adaptable →
  Flexible System Context sys flexible modifiable tailorable adaptable.
```

# Leaf Property – Adaptable

```
Inductive Adaptable (System: Set) (Context: Set) (sys: System)
                    (adaptable: System → Context → Prop)
                    : Prop :=
  mk_adaptability:
    (∀ cx: Context, adaptable sys cx) →
      Adaptable System Context sys adaptable.
```

# Example – Smart Home

Define System, Stakeholder, and Context for a Smart Home

```
Require Import Satisfactory.
Require Import Changeable.

Definition Smart_Home_System := Datatypes.unit.
Inductive Smart_Home_Stakeholder := investor | end_user |
developer | maintainer | public.
Inductive Smart_Home_Context := normal.
```

# Example – Smart Home

Create a Specific Adaptability Requirement using Ross's Approach

Definition smart_home_system_adaptability_requirement :
**changeStatement** :=
mk_changeStatement
(perturbation_shift "low temperature")
(context_circumstantial "late at night")
phase_preOps
(agent_internal "controller")
(mk_change direction_increase (parameter_level "knob angle")
(origin_one "degree") (destination_one "degree") aspect_function)
(mechanism_description "regulating the airflow")
(mk_change direction_increase(parameter_level "temperature")
(origin_one "degree") (destination_one "degree") aspect_function)
(abstraction_architecture " ")
valuable_simple

# Example – Smart Home

**Corresponding requirement statement:**
In response to (Perturbation_shift) *low temperature* (Context_circumstantial) *late at night*, during (Phase_preOps) of system, desire (Agent_internal) *controller* to be able to (Direction_increase) the (Parameter_level) of *knob angle* from (Origin_one) state(s) to (Destination_one) state(s) in the system (Aspect_function) through (Mechanism_description) *regulating the airflow* that results in the effect of (Direction_increase) the (Parameter_level) of *temperature* from (Origin_one) state(s) to (Destination_one) state(s) in the system (Aspect_function) for a (Abstraction_architecture) that is (Valuable_simple).

# Example – Smart Home

Check a given system meets the adaptability requirement

```
Inductive systemMeetsSpecificAdaptabilityRequirement:
Smart_Home_System → changeStatement → Prop :=
  systemMeetsSpecificAdaptabilityRequirement_proof:
    ∀ s: Smart_Home_System, ∀ c: changeStatement,
      In adaptability (tipeAssignment c) →
systemMeetsSpecificAdaptabilityRequirement s c.
```

# Example – Smart Home

Check a given system has adaptability quality

```
Inductive adaptability (sys: Smart_Home_System) (cx:
Smart_Home_Context): Prop :=
  adaptability_proof:
systemMeetsSpecificAdaptabilityRequirement sys
smart_home_system_adaptability_requirement →
      adaptability sys cx.
```

# Example – Smart Home

Formalize two properties with trivial proofs

`Inductive` **systemCanControlFurnaceOnOffSwitch**:
Smart_Home_System → `Prop` :=
systemCanControlFurnaceOnOffSwitch_proof: ∀ $s$:
Smart_Home_System, **systemCanControlFurnaceOnOffSwitch** $s$.

`Inductive` **systemCanControlGarageDoorOpener**:
Smart_Home_System → `Prop` :=
systemCanControlGarageDoorOpener_proof: ∀ $s$:
Smart_Home_System, **systemCanControlGarageDoorOpener** $s$.

# Example – Smart Home

Check a given system has Physical Capability quality

`Inductive` **physicalCapability** $(sys$: Smart_Home_System) $(sh$: **Smart_Home_Stakeholder**) $(cx$: **Smart_Home_Context**): `Prop` :=
physicalCapability_proof: **systemCanControlFurnaceOnOffSwitch** $sys \wedge$ **systemCanControlGarageDoorOpener** $sys \rightarrow$ **physicalCapability** $sys\ sh\ cx$.

# 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*

# Conclusion

- To be added

# Bibliography

Basir, N., Denney, E., & Fischer, B. (2009). Deriving safety cases from automatically constructed proofs.

Boehm, B., & Kukreja, N. (2015). An initial of an ontology for system ilities. In *To be presented in 25th annual incose international symposium.*

Graydon, P. J., & Knight, J. C. (2008). *Success arguments: Establishing confidence in software development.*

Graydon, P. J., Knight, J. C., & Strunk, E. A. (2007, Jun). Assurance based development of critical systems. In *37th annual IEEE/IFIP international conference on dependable systems and networks (dsn 2007).*

Kelly, T., & Weaver, R. (2004). The goal structuring notation–a safety argument notation. In *Proceedings of the dependable systems and networks 2004 workshop on assurance cases.*

Lundberg, L., Bosch, J., Hggander, D., & Bengtsson, P.-O. (1999). Quality attributes in software architecture design. In *Proceedings of the iasted 3rd international conference*