A Direct Path to Dependable Software

Communications of the ACM Magazine, April 2009

Daniel Jackson

March 16, 2012
Sanghyun Yoon
Contents

- Introduction
- The Need for a Direct Approach
- Why Testing Isn’t Good Enough
- A Direct Approach
- Implications
- Closing Thoughts
Introduction

• The growing tendency to embed software in powerful systems, e.g., medicine, pacemakers. It can save lives but also kill.

• What would it take to make software more dependable?
  – Most approaches have been indirect, e.g., processes, tools or techniques.
  – Developers should instead produce direct evidence of their software’s dependability.

• The potential advantages of direct evidence:
  – Greater credibility
  – Reduced cost
    • Development resources can be focused where they have the most impact.
The Need for a Direct Approach (1)

• A dependable system is one you can depend on
  – That is, you can place your trust in it.
  – A rational person or organization only does with evidence that the system’s benefits far outweigh its risks.

• Industrywide data collection and analysis barely exist.
  – Cars, aviation: accidents data is collected and analyzed (NHTSA, NTSB in U.S.)
    • Result: Seatbelts and airbags are efficient
  – The software industry has no comparable mechanism,
  – And we have almost no data on the causes or effects of software failure.
  – > Improving designs and development strategies is difficult.
The Need for a Direct Approach (2)

- Over the past few decades we have developed approaches and technologies that can dramatically improve the quality of software.
  - Platforms, development infrastructure, processes, tools

- But, there is still no evidence compelling enough that simply using a given approach establishes with confidence the quality of the resulting system.

- Many certification standards have had not good effect.
  - They say which technique, not what the system satisfies.
  - Cost is very expensive. Too many demand
  - Certified systems sometimes fail catastrophically.

  - e.g.) DO178B, the safety standard used in U.S. for avionics systems, requires MCDC.
  - That is extremely costly and whose benefits studies have yet to substantiate.
The Need for a Direct Approach (3)

- Direct approach

  - Direct approach: “goal-based” or “case-based” certification
    - Developers provide direct evidence that satisfies the goal
    - Check whether the system satisfies the direct evidence

  - This approach is now gaining currency
    - In the U.K., the Ministry of Defence dramatically simplified its procurement standards for software under this approach, with contractors providing “Software reliability cases” to justify the system.

  - The direct approach recommended by the study (effectiveness and costs of existing approaches to software dependability) is the basis of this article.
Why Testing Isn’t Good Enough

• Testing is a crucial tool in the software developer’s repertoire.
  – There is a widespread folk belief that testing is sufficient evidence for dependability

• Testing can be used to show the presence of errors but not their absence. (by Edsger Dijkstra)
  – Test result is all TRUE -> no error?
    • No errors about the input data.
Why Testing Isn’t Good Enough

• It is very difficult even to achieve full code coverage.
  – Although, some small components can be tested exhaustively, but the proportion of scenarios executed in a typical test is vanishingly small.
  – Furthermore, user’s input data range is larger than developers.
    • Cannot expect all input data

• An alternative approach: generating tests that matches the expected usage
  – Too many test cases are needed
    • One input in a thousand, revealing no bugs -> 5,000 test cases
    • Revealing 10 bugs -> 20,000 test cases

• Testing cannot deliver the confidence required at a reasonable cost.
A Direct Approach  
- Fundamental questions

• What is a system?
  – Any other components: Software, hardware platform, peripheral devices, human operators and users
  – Designs should concern all of them, even if not economically feasible things.

• What does “dependable” mean?
  – A system is trusted to perform a particular task.
  – A system cannot be dependable without evidence.
A Direct Approach
- Characteristics of Dependability

• Dependability is a trade-off between benefits and risks.
  – The level of assurance being chosen to matching the risk at hand.

• Dependability can not be measured on simple numeric scale.
  – Different kinds of failures have very different consequences.

• The cost of preventing all failures will usually be prohibitive, so a dependable system will **not offer uniform levels of confidence across all functions.**
  – E-commerce site
    • May display advertisements incorrectly
    • Give bad search results
    • Perhaps lose shopping-cart items
  
  • Must never bill the wrong amount or leak customers’ credit card details
A Direct Approach

• First step: **drawing its boundaries**
  – Deciding system boundaries
  – Identifying the critical properties (dependability goal)
  – Determining what level of confidence is required.

• A critical property may be related with a set functions.

• Checking function vs. property
  – Full correctness of these functions would likely be neither necessary nor sufficient.
A Direct Approach
- Traditional terms

• Most software system interact with the physical world.
  – They should form the vocabulary for expressing critical properties.
  – There is long tradition of writing requirements in terms of interfaces closer to the software
    • It is easier
    • To divide labor from system-level concerns
  – The assumptions between software and real world may be a problem
A Direct Approach
- Dependability Case

• The evidence for dependability takes the form of a dependability case.

• Dependability case
  – An argument that the software, in concert with other components, establishes the critical properties.
  – Mix of informal and formal argument is appropriate, but certain features

• Dependability case should be
  – Auditable: a third party certifier can evaluate it without any expert knowledge
  – Complete: any assumptions should be noted
  – Sound: it should not make unwarranted assumptions that certain components fail independently
Implications

• Dependability case as product
  – By focusing on the case, the developer can make decisions that ease its construction, most notably by designing the system so that critical properties are easier to establish.
  – It demands a sea change in attitude.
  – Developers are called upon, to consider their decisions in the light of the system’s dependability and to view the evidence.
Implications

• Procurement
  – The goals set at the start must be realistic in terms both of their achievement and demonstration.
  – Fundamentally, our society recognize that functionality and dependability are inherently in conflict.

  – Indirect approach to certification: give detailed prescriptions
    • How the software should and should not be developed
    • Which technologies should be used.

  – Direct approach frees the developer to use the best available means to achieve the desired goal.
Implications

• Structuring requirements
  – Just listen carefully to the stakeholders
    -> the purpose they believe these functions will enable them to accomplish
    -> that the requirements be expressed precisely and succinctly
    -> that great care be taken to avoid making irrevocable decisions
  – Clearly record any assumptions made about the software’s operating environment.
  – List of function -> which requirement is critical?
    • e.g.) Down grading
Implications

- Decoupling and simplicity
  - Localize a critical property to a component
    - If then, attention can be focused on that component
      - e.g.) e-commerce system: billing and the other
    - Minimization of communication
    - Appropriate language features to protect against errors
  - Simplicity is not easy to achieve
    - Decoupling is an important way of securing simplicity
    - The cost of simplicity is high, but the cost of lowering the floodgate to complexity is higher.
Implications

- Process and culture
  - A rigorous process will be needed to ensure that attention is paid to the dependability case and to preserve the chain of evidence as it is constructed.
  - A rigorous process need not be a burdensome one.
    - Freeing the engineers to concentrate on the more creative aspects of the project.
  - A paradox: burdensome processes do seem to be correlated with more dependable software (a social effect)
  - Dependability cases will always contain informal elements that cannot be verified mechanically
    - The case produce credibility itself and also organization required for that.
Implications

• Robust foundations
  – A robust software system cannot be built on a foundation of weak tools and platforms
  – C# libraries, Strong typing
  – Complexity in a programming language can compromise dependability
    • It increase chance that the program will behave differently from what the programmer envisaged.

• Commodity platforms
  • Usually has the advantage of lower cost and larger pool of candidate developers
  • Not usually designed for critical applications
Implications

• The (in)significance of code
  – Many researches learn that the correctness of code is rarely that weakest link.
    • Cause 3% of the time
  – Nevertheless, the correctness of code is a vital link in the dependability chain.
Implications

- Testing and analysis
  - Testing cannot generally deliver the high levels of confidence that are required for critical systems.
  - Analysis is needed to fill the gap.
  - Analysis might involve any of a variety of techniques, depending on the kind of property being checked and the level of confidence required.
  - Techniques: theorem proving, static analysis, model checking, model finding
Implications

• Credibility of tools
  – Sound tool
    • No bugs and there is no possible execution that can violate a property.
  – Even if an analysis establishes a property with complete assurance, the question of whether the property itself is sufficient still remains.
  – Mathematical proof is generally believed to offer the highest level of confidence.
Closing Thoughts

• Software system is dependable -> good reason/evidence

• The system must concern all correctness of components
  – Including human error

• The cost of constructing a dependability case may be high.
  – Obtaining decoupled design may reduce the cost of maintenance later.

• When low levels of confidence, testing may be the most cost-effective way to establish dependability.

• The direct approach based on dependability cases gives developers an incentive to use whatever development methods and tools are most economic and effective.