Software Testing and/or Software Monitoring: Differences and Commonalities

Antonia Bertolino
ISTI-CNR, Pisa, Italy
antonia.bertolino@isti.cnr.it

Keynote at Jornadas Sistedes 2014 - Cádiz, 18 September 2014

In the third millennium society software is increasingly pervasive
Unfortunately software may be faulty ... 


Errare humanum est...

Awareness that introducing faults in software is somehow unavoidable has been acquired very soon ...
... perseverare autem diabolicum

... this is why, as with any other engineering artifact, along the production of software we apply approaches and tools for assuring and evaluating its quality (fitness for purpose)

Potential threats to software dependability and means to deal with them are intensely studied

Many approaches to sw verification

Mixing different approaches

• Much research effort since the early days of SE has been spent to assess and compare the effectiveness of V&V approaches
Comparing the Effectiveness of Software Testing Strategies

VICTOR R. BASILI, SENIOR MEMBER, IEEE, AND RICHARD W. SELBY, MEMBER, IEEE

Abstract—This study applies an experimentation methodology to compare three state-of-the-practice software testing techniques: code reading by stepwise abstraction, b) functional testing using equivalence partitioning and boundary value analysis, and c) structural testing using 100 percent statement coverage criteria. The study compares the strategies in three aspects of software testing: fault detection effectiveness, fault detection cost, and class of faults detected. Thirty-two professional programmers and 42 advanced students applied the three techniques to four unit-sized programs in a fractional factorial experimental design. The major results of this study are the following: 1) With the professional programmers, code reading detected more software faults and had a higher fault detection rate than did functional or structural testing, while functional testing detected more faults than did structural testing, but functional and structural testing were not different in fault detection rate. 2) In one advanced student subject group, code reading and functional testing were not different in faults found, but were both superior to structural testing, while in the other advanced student subject group there was no difference among the techniques. 3) With the advanced student subjects, the three techniques were not different in fault detection rate. 4) Number of faults observed, fault detection rate, and total effort in detection depended on the type of software tested. 5) Code reading detected more interface faults than did the other methods. 6) Functional testing detected more control faults than did the other methods. 7) When asked to estimate the percentage of faults detected, code readers gave the most accurate estimates while functional testers gave the least accurate estimates.

Copyright © 2014 ACM 089791-7644/14/0009...

Proceedings Paper - A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections

Glenford J. Myers
IBM Systems Research Institute

This paper describes an experiment in program testing, employing 59 highly experienced data processing professionals using seven methods to test a small PL/I program. The results show that the popular code walkthrough/inspection method was as effective as other computer-based methods in finding errors and that the most effective methods (in terms of errors found and cost) employed pairs of subjects who tested the program independently and then pooled their findings. The study also shows that there is a tremendous amount of variability among subjects and that the ability to detect certain types of errors varies from method to method.

Key Words and Phrases: software reliability, program verification, debugging, testing, code walkthroughs, code inspections, personnel selection

Commun. ACM 21, 9 (September 1978)
Mixing different approaches

- Much research effort since the early days of SE has been spent to assess and compare the effectiveness of V&V approaches

- It is now widely acknowledged that the different V&V approaches yield differing effectiveness and target different types of faults, and to get the most benefits they should be used in combination
Mixing different approaches

• Much research effort since the early days of SE has been spent to assess and compare the effectiveness of V&V approaches

• It is now widely acknowledged that the different V&V approaches yield differing types of faults and target different types of faults, and to get the most benefits they should be used in combination

The traditional software life cycle

In early SE a neat separation between product development and usage was considered.
Breaking the separation between development (lab) and operation (real world)

e.g., see Luciano Baresi and Carlo Ghezzi. 2010. The disappearing boundary between development-time and run-time. In Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER ’10)

Emerging trends

Software intensive systems have become the foundation of modern technology

- Software content in virtually every product and service will continue to grow—in some cases dramatically
- Software must be demonstrably safe, secure, and reliable
- Requirements continuously emerge as systems dynamically evolve

[R. Pressman, "Emerging trends in software engineering", JASST09]
Emerging trends

- Unprecedented pace of change:
  - Requirements change continuously
  - Blurring boundaries between the system and the external world
  - Both components and the overall structure can change
  - No single organization is in control of the whole system
  - Self-healing and self-adaptation

“We must shift from the prevailing synchronous approach to distributed programming to a fundamentally more delay-tolerant and failure resilient asynchronous programming approach. Global behaviors emerge by asynchronous combinations of individual behaviors, and bindings and compositions change dynamically.”


QoS contracts

- More and more systems are formed by the cooperation among autonomous software services, and their global behaviour emerges as a result.

- For such kind of systems we need to verify that each involved entity complies with its “contract”, which is impossible prior to execution.
Challenges

• System assembled dynamically
• Reference specification of expected/correct operation not a-priori available
• Specifications are learnt/inferred, thus they can be incomplete, unstable, uncertain
• Special emphasis on run-time assessment
• Assessment activities must accommodate change (and must be adaptable themselves)

What about V&V?

• It is our thesis that with the blurring boundary between production and use (pre and post deployment) we should try to get the most mutual benefits from combining pre and post deployment V&V
Validation @ runtime

• All such trends go towards the direction of moving V&V to the field after deployment

• Relies on sensing what is happening and on timely collecting relevant information

→ To do this, we need to monitor systems behaviour

Two generic approaches to sw verification
Two generic approaches to sw verification

Research and solutions for testing (in the lab) and monitoring (in real usage) have taken quite separate tracks

Goal of this talk
Surveying and discussing potential ways in which testing and monitoring can benefit each other
Why testing and monitoring?

A word cloud from titles of most recent papers of mine
Agenda

• In the rest of the talk I will:
  i. Make a shot recap of main concepts and research challenges in software testing and software monitoring
  ii. Overview a few proposals in the literature for combining methods or mutually benefit from each other, while...
  iii. ... attempting a taxonomy of testing+monitoring research

Software Testing

Software testing is a major activity within a product Verification and Validation. Testing is a complex process including more than “just” running test items, it also requires several preparation and follow-up activities.

Dynamic Test Processes

All-inclusive definition

Software testing consists of

- the *dynamic* verification of the behavior of a program
- on a *finite* set of test cases
- *suitably selected* from the (in practice infinite) input domain
- against the *expected behavior*


Software testing conceptual flow

Ref → Test Inputs → SUT → Ouputs and/or Traces → Expected O/Tr → Pass/ Fail

© 2014 A. Bertolino: Sw testing and/or sw monitoring
Software Monitoring

• the process of dynamic collection, interpretation, and presentation of information concerning objects or software processes under scrutiny


In other words...

• A monitor is a system that observes the spontaneous behavior of a system and, given a specification of desired properties, it checks that such properties hold for the given execution
Software Monitor conceptual flow

An over-loaded term

- Large (but fractioned) body of research, carried out over decades
- Different authors use the term “monitoring" to indicate different things/aspects
- A monitoring system is in fact an assembly of different pieces dealing with different concerns
Components of a monitoring system

A useful reference

A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools
Nelly Delgado, Student Member, IEEE, Ann Quiroz Gates, Member, IEEE Computer Society, and Steve Roach, Member, IEEE Computer Society

Abstract—A goal of runtime software-fault monitoring is to observe software behavior to determine whether it complies with its intended behavior. Monitoring allows one to analyze and recover from detected faults, providing additional defense against catastrophic failure. Although runtime monitoring has been in use for over 30 years, there is renewed interest in its application to fault detection and recovery, largely because of the increasing complexity and ubiquitous nature of software systems. This paper presents a taxonomy that developers and researchers can use to analyze and differentiate recent developments in runtime software fault-monitoring approaches. The taxonomy categorizes the various runtime monitoring research by classifying the elements that are considered essential for building a monitoring system, i.e., the specification language used to define properties, the monitoring mechanism that oversees the program’s execution, and the event handler that captures and communicates monitoring results. After describing the taxonomy, the paper presents the classification of the software-fault monitoring systems described in the literature.

Index Terms—Assertion checkers, runtime monitors, specification, specification language, survey, software/program verification.
Testing vs. Monitoring

In a nutshell, both activities observe some executions in order to detect issues or assess sw quality (wrt some specified property). However, major differences/complementarities can be noticed concerning:

- What executions are observed
- What properties are checked
- The context, i.e., when/where/how the observation is performed

Testing vs. Monitoring: what executions?

Active vs. Passive

- In ST, “generally” the test cases are selected based on various criteria, including strategy, testing stage, fault model, available tools and resources
- In SM, system executions are observed ideally without interference: the aim is to keep under control the system spontaneous behaviour
### Pros and Cons of

**Actively selecting test cases**
- + can detect possible issues in advance
  - - but the identification of “effective” test cases can be quite complex and costly
  - + negative tests
  - - for some test approaches, determining the test oracle can be very difficult

**Passively observing what happens**
- - a problem is found only after it happened
  - + no cost involved in selection
  - - no robustness or stress check
  - + oracle somewhat implicit in the monitored property

---

### Passive testing

- In contrast to more common “active testing”, passive testing consists of passively observing a system in operation without interfering with it in order to detect faults
- The approach has been mostly investigated in the field of networks for conformance testing of network protocols
Active tester vs. passive tester

Ref → Test Inputs → SUT → Outputs and/or Traces → Expected O/Tr → Pass/ Fail

SUT → Outputs and/or Traces → Ref → Pass/ Fail

This is in fact a monitor

Ref → Test Inputs → SUT → Outputs and/or Traces → Expected O/Tr → Pass/ Fail

SUT → Outputs and/or Traces → Ref → Pass/ Fail
Active tester, passive tester, monitor

- Emphasis of research in passive testing is in formal approaches to express the properties to be checked and in making the monitor more effective in detecting failures

Testing vs. Monitoring: what properties?

- In both ST and SM functional and non-functional properties can be checked
- In ST functional somehow prevails, while research in testing of non-functional properties stays behind
- In SM non-functional is more common domain
  - This descends naturally from the difficulty of assessing performance, reliability, and security properties in a context different from the actual one
Complex event processing

The concept of event processing is older than computing itself: reacting to events is a natural part of our society and economy.

Event processing has recently emerged as a substantial new field of software engineering.

Complex event processors (CEP) combine information from many data points to generate new insights, and use algorithms and rules to process event data received from one or more sources during some time period. Complex events are produced as summary level facts that are put in context to identify threats and opportunity situations.

Ref: The event processing manifesto
Dagstuhl Seminar on Event Processing 2010

Testing vs. Monitoring: the context
Reproduction vs. genuine, off-line vs. “in-process”

• In ST, the test execution requires a proper test environment, i.e. all facilities, hardware, software, firmware, procedures, and documentation intended for or used to perform the tests
  – Setting up the test environment may be costly and complex
Testing vs. Monitoring: the context
Reproduction vs. genuine, off-line vs. “in-process”

• In ST, the test execution requires a proper test environment, i.e. all facilities, hardware, software, firmware, procedures, and documentation intended for or used to perform the tests
  – Setting up the test environment may be costly and complex

• On the other hand, lab tests are safe, i.e. no effect in real world

Testing vs. Monitoring: the context
Reproduction vs. genuine, off-line vs. “in-process”

• In ST, the test execution requires a proper test environment, i.e. all facilities, hardware, software, firmware, procedures, and documentation intended for or used to perform the tests
  – Setting up the test environment may be costly and complex

• On the other hand, lab tests are safe, i.e. no effect in real world

• In SM, the system is verified in operation: no need to reproduce environment, however costs are still implied for instrumenting or annotating the system so to allow observation

• Probe effect: The monitor could interfere with system behaviour
How can ST and SM benefit each other?

• Traditionally separate research domains, but several recent proposals for positively contaminating the two activities

Let’s see a sample from literature

Collecting and mining field data for improving ST process

• An obvious synergy between ST and SM is collecting data about the runtime behavior of deployed programs while they are in use, and use such data for improving the ST process, including:
  – Bug statistics
  – Fault localization and repair
  – Augmenting a test suite
  – Reproducing real failures in the lab

Collecting and mining field data for improving ST process


Moving testing to the field

Testing in the field

• For dynamic systems whose behaviour, changes and user profile are hard to predict and simulate
• Differently from monitoring, online testing is proactive:
  – Big challenges in terms of costs, acceptance, side effects...

Context for testing in the field

And vice versa, moving monitoring in the lab

- Runtime verification is a type of monitor, with an emphasis of the research on the specification of properties to be monitored.
- Runtime verification monitors programs for testing purposes, whereby the emphasis is on collecting the traces rather than on the actual usage context.
- For the purpose of this talk RV can be seen as monitoring performed in the lab.

Runtime verification

- A relatively new discipline (RV series started in 2001)
- Introduced as a lightweight verification approach, to complement heavyweight ones such as theorem proving and model checking.
Runtime verification

- Runtime verification is the discipline of computer science that deals with the study, development, and application of those verification techniques that allow checking whether a run of a system under scrutiny satisfies or violates a given correctness property.


Monitoring as seen by RV

- A monitor is a device that reads a finite trace and yields a certain verdict

- For a monitor to be ideally suited for RV, it should adhere to the two maxims impartiality and anticipation.
  - **Impartiality** requires that a finite trace is not evaluated to true or, respectively false, if there still exists an (infinite) continuation leading to another verdict.
  - **Anticipation** requires that once every (infinite) continuation of a finite trace leads to the same verdict, then the finite trace evaluates to this very same verdict.
Test process automation

- By exploiting the instrumenter and the trace analyzer of a monitor a test session could be fully automated

Example of a complete framework for MBT and RV of model simulator and/or code

Li Tan, Jesung Kim, Insup Lee, Testing and Monitoring Model-based Generated Program, ENTCS, 89 (2), 2003, 128-148
Monitoring can be exploited as a test oracle

Using test coverage criteria for improving monitor observation capability

Even using different approaches, they all share one limitation: they can only passively observe and elaborate what happens.

Sometime it can be important also to know what does NOT happen, e.g. ...

- some service has not been invoked for long time, or
- some usage profile is different from expectations ...

Wisdom is knowing how little we know

[Socrates, Athens, 469–399 BC]
Using test coverage criteria for improving monitor observation capability

Empower the monitor with the knowledge to decide if what is being observed is adequate

We introduce the notion of a monitoring coverage criterion

Monitoring coverage

✧ Similarly to test coverage criteria, we define a set of entities that a monitor should cover

✧ Differently from test coverage:
  – we need to set how long to wait for observing an entity
    ⇒ Observation window $\delta$
  – monitor adequacy can increase or decrease along time
Adequate monitor architecture

Rules Generator: generates the rules from the entities, depending on the coverage criterion and the observation windows
Adequate monitor architecture

**WS_Monitor**: receives event messages from the services involved in the composition and forwards them to the Inferential Engine.

**Inferential Engine** (impl. Drools): matches the rules and executes the associated actions.

**Coverage Analyzer**: visualizes collected info, elaborates stats, raises alerts, etc..
provides cumulative coverage
At each $t$, provides individual coverage.
Summarizing:

• There are techniques from testing discipline that could be used in operation to make monitoring more powerful, e.g.
  ❖ proactive launch of test cases in the deployed system (how to deal with effects)
  ❖ measuring coverage within defined observation windows
Summarizing:

- There are techniques from monitoring discipline that could be used in laboratory testing to improve automation and reduce costs, e.g.
  - resolve non determinism (oracle)
  - test selection

Summarizing:

- Each activity could mutually benefit from each other in a perpetual verification approach
Not only system test and monitoring

• E.g., adequate monitor can be used to assess learners performance

Learn PAd Project
Model-Based Social Learning for Public Administrations
http://www.learnpad.eu
Learn PAd Motivation

Training and knowledge sharing within PA are characterized by (state of the practice):
- Informative and off-line approach
- Difficulties in providing “on-the-job” support
- Difficulties in knowledge sharing
- Difficulties in identifying knowledge sources and competences
- Users perspective is often ignored

...
Learn PAd innovation

- Model based collaborative learning
- BP driven content production
- Automated verification of local/global process coherence
- Automated evaluation of natural language defined contents
- Automated model-based derivation of self-assessment tasks
- Model based simulation for learning purpose
- Ontology based and OSS inspired KPIs

© 2014 A. Bertolino: Swe testing and/or see monitoring