Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Hovac: A Fault Injection Framework for Benchmarking the Dependability of C/C++ Applications

Lena Herscheid, Daniel Richter, and Andreas Polze

Operating Systems & Middleware Group
Hasso Plattner Institute at University of Potsdam, Germany

Summary

Hovac

- configurable tool for dependability benchmarking
- uses DLL API hooking to inject faults into third party library calls
- fault classes: based on the Common Weakness Enumeration (CWE) database
- detailed and systematic approach to benchmark the dependability of C/C++ applications

Motivation

reuse of commercial off-the-shelf (COTS) software components:

- enhances modularity & developer productivity
- new kind of risk: rely on robustness & correctness of third-party code

- failure of third-party libraries is much harder to predict, debug, & fix than in-house code
- portability to unreliable (e.g. distributed) environments

Dependability Benchmarking

- Repeatable, uniform, comparable ways of quantifying dependability aspects of software systems are needed, built similarly to performance benchmarks. Benchmarking is an "experimental approach to measure well-defined features of a system or component according to an agreed (i.e., accepted as standard) set of methods and procedures"

H. Madeira and P. Koopman, "Dependability Benchmarking: Making choices in an n-dimensional problem space," in *First Workshop on Evaluating and Architecting System Dependability (EASY)*, 2001.

Dependability Benchmarking

- compare a well-defined set of dependability attributes in a systematic and re-usable manner
  - provide fine-grained and precise control over the injected faultload
- aim:
  - **portable** for a well-defined set of platforms and applications
  - **reproducible** in a controlled and observable experiment setting
  - **representative** of “faultloads in real life”

Dependability Benchmarking

Hovac:

- primarily in the fault injection domain
- contrary to software testing strategies we do not aim at maximizing input or control flow path coverage
- try to activate and trigger interesting faults which allow developers to fix problems in their fault tolerance or error handling code, thus increasing robustness.

Hovac 101

- focus: multithreaded single-node applications written in C++
- fault model: erroneous behavior of third-party libraries; all sub-classes of computation faults*
- system under test: application depending on potentially unreliable third-party libraries
- approach: orchestrated fault-injection experiments based on dynamic-link library (DLL) hooking; injected at interface calls into foreign libraries

*) according to: F. Cristian, "Understanding fault-tolerant distributed systems," *Communications of the ACM*, vol. 34, no. 2, pp. 56–78, 1991.

Related Work

Hovac: A Fault Injection Framework for Benchmarking the Dependability of C/C++ Applications

Related Work

- automated software testing
  - generation of robustness test cases for black box software interfaces

S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. McMinn et al., "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software*, vol. 86, no. 8, pp. 1978–2001, 2013

- fault injection

R. Natella, D. Cotroneo, J. A. Duraes, and H. S. Madeira, "On fault representativeness of software fault injection," *Software Engineering, IEEE Transactions on*, vol. 39, no. 1, pp. 80–96, 2013.

H. Ziade, R. A. Ayoubi, R. Velazco et al., "A survey on fault injection techniques," *Int. Arab J. Inf. Technol.*, vol. 1, no. 2, pp. 171–186, 2004.

Dependability Benchmarking

DBench

- five prototype benchmarks for different application domains
  - operating systems, automotive, on-board space control, online transaction processing (OLTP)
- focus: effectiveness of several corruption techniques for faultload implementation

K. Kanoun, J. Arlat, D. Costa, M. Dal Cin, P. Gil, J.-C. Laprie, H. Madeira, and N. Suri, "DBench: dependability benchmarking," in *Supplement of the 2001 Int. Conf. on Dependable Systems and Networks (DSN-2001)*, 2001, pp. 12–15.

Dependability Benchmarking

Koopman et al.

- compare the dependability of operating systems using an exhaustive set of system calls
- key insight: interpretation of which failure class is most severe depends on the application area
- even at the operating system level, the reliability of external code should not be taken for granted

P. Koopman, J. Sung, C. Dingman, D. Siewiorek, and T. Marz, "Comparing operating systems using robustness benchmarks," in Reliable Distributed Systems, 1997. *Proceedings., The Sixteenth Symposium on. IEEE*, 1997, pp. 72–79.

Dependability Benchmarking

EDFI

- combination of dynamic and static source code instrumentation, which is used to insert a dynamic fault model
- key insights: strong faultload coverage and reproducibility are hard to guarantee during fault injection.
  - makes injection campaigns more controllable and precise by introducing a controller component

C. Giuffrida, A. Kuijsten, and A. Tanenbaum, "EDFI: A dependable fault injection tool for dependability benchmarking experiments," in Dependable Computing (PRDC), *2013 IEEE 19th Pacific Rim International Symposium on*, Dec 2013, pp. 31–40.

Fault Injection

CrashMe

- robustness of operating systems
- execute random byte sequences
- key insight: uncovered various flaws in different operating systems
- strong motivation for our work, as they demonstrate need for more robustness testing

G. J. Carrette, "Crashme: Random input testing," http://people.delphiforums.com/gjc/crashme.html, 1996

Fault Injection

FlakyIO

- software exception injection; test exception handling code

M. W. Bigrigg, “Framework for exercising i/o exception handling code,” Int. J. Inf. Commun. Techol., vol. 1, no. 3/4, pp. 244–258, Mar. 2008. [Online].

LFI

- inject faults at boundaries between application and its dynamically linked libraries; focus: detect incomplete/erroneous handling of return values

P. D. Marinescu and G. Candea, “LFI: A practical and general library-level fault injector,” in Dependable Systems & Networks, 2009. DSN’09. IEEE/IFIP International Conference on. IEEE, 2009, pp. 379–388.

Interface Robustness Testing

Microsoft TestApi https://testapi.codeplex.com/

- CLR profiling API
- fault rules (method signature, fault condition, fault type)

Ballista

- automated software testing vs. fault injection
- test input selection based on coverage criteria
- modified argument values in POSIX API calls

N. P. Kropp, P. J. Koopman, and D. P. Siewiorek, "Automated robustness testing of off-the-shelf software components," in *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty Eighth Annual International Symposium on*. IEEE, 1998, pp. 230–239.

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Our Approach

Hovac: A Fault Injection Framework for Benchmarking the Dependability of C/C++ Applications

Library Load Time Fault Injection

- portability, reproducibility, and representativeness can be traded off for each other

Library Load Time Fault Injection

- categorization according to

C. Giuffrida, A. Kuijsten, and A. Tanenbaum, "EDFI: A dependable fault injection tool for dependability benchmarking experiments," in Dependable Computing (PRDC), 2013 IEEE 19th Pacific Rim International Symposium on, Dec 2013, pp. 31–40.

Library Load Time Fault Injection

Trigger Mechanism

- where and when, during the program's execution, faults should be injected, and what feature of the program causes such an injection.
  - Time-based: inject faults at periodic time intervals,
  - Location-based: inject faults into predetermined memory locations,
  - **Execution-driven: inject faults based on the control flow during runtime**.
- Hovac: function call boundaries; synchronous as it deterministically injects faulty code before and after a function call.

Library Load Time Fault Injection

Injection Time

- when does the injection take place?
  - Pre-runtime, usually by adding code mutations,
  - During runtime, by using available hardware or software traps to interrupt the execution and inject faults,
  - **At library load time**.
- our approach: linking against a custom error state generating library before the third party library is loaded

Library Load Time Fault Injection

Level of Injection

- What representation of the program is modified?
  - **Binary**,
  - Intermediate code representation,
  - Source code.
- Hovac: hooking to function calls at link time, making no source code modifications necessary

Library Load Time Fault Injection

Purpose of Fault Injection

- What system characteristics are targeted?
  - **To compare the dependability of systems,**
  - **To evaluate fault tolerance mechanisms,**
  - To increase test coverage.
- Hovac: ordering with regard to certain dependability characteristics

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Faultload

Hovac: A Fault Injection Framework for Benchmarking the Dependability of C/C++ Applications

Faultload

- „upsetload“/“faultload“
- Common Weaknesses Enumeration Database (CWE)
  - classify all kinds of software weaknesses
    - i.a. programming language, severity, kinds of error states
  - created & managed by MITRE research corporation
  - not only security issues but all kinds of weaknesses
  - publicly available source of failure date from industry

Hovac: classification by types of error states & activation

Fault Classes

Computation

- variables, in particular computation results of primitive data types, contain a value different from what was expected.
  - Off by One (CWE ID 193)
  - Signed to Unsigned Conversion (CWE ID 195)

Timing

- certain part of the code takes more than the expected time to execute
  - Hovac: call to a library function returns too late

Fault Classes

Control Flow

- input triggers an incorrect execution path through the application
  - unhandled exceptions

Environment

- interaction between the program and its environment is other than expected; unforeseen states in the execution environment or the operating system; programmer's assumptions regarding the environment are violated
  - Signal Errors (CWE ID 387)

Fault Classes

Race Condition

- accesses to shared memory are not properly synchronized
  - switch statements (CWE ID 365)
  - data shared between multiple threads (CWE ID 366)
  - signal handlers (CWE ID 364)

Memory

- state of the memory is corrupted.
  - specifically Hovac/C/C++: corruption or leaking of heap & stack memory due to programming mistakes
  - Heap-/Stack-based Buffer Overflow (CWE IDs 121-122)

Hovac Fault Classes

- scope is limited to faults which entail observable error states as listed before
- focus less on hard faults
  - lead to a failure regardless of the conditions under which the program is executed
  - are detectable by traditional testing approaches
- do not deal with architecture-specific and portability problems
  - e.g. data structure alignment or compiler misbehavior.

Hovac Fault Classes

- implemented CWE entries applicable to C/C++implemented in a pluggable fashion
- generic AbstractFault interface can be extended to incorporate further faults
- library loader automatically loads the faults and include them in fault injection experiments
- future faults, or a set of faults to be used for a benchmark, can be encapsulated in a dynamic link library, as long as they correspond to the AbstractFault interface.

HPI Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Fault Injection Implementation

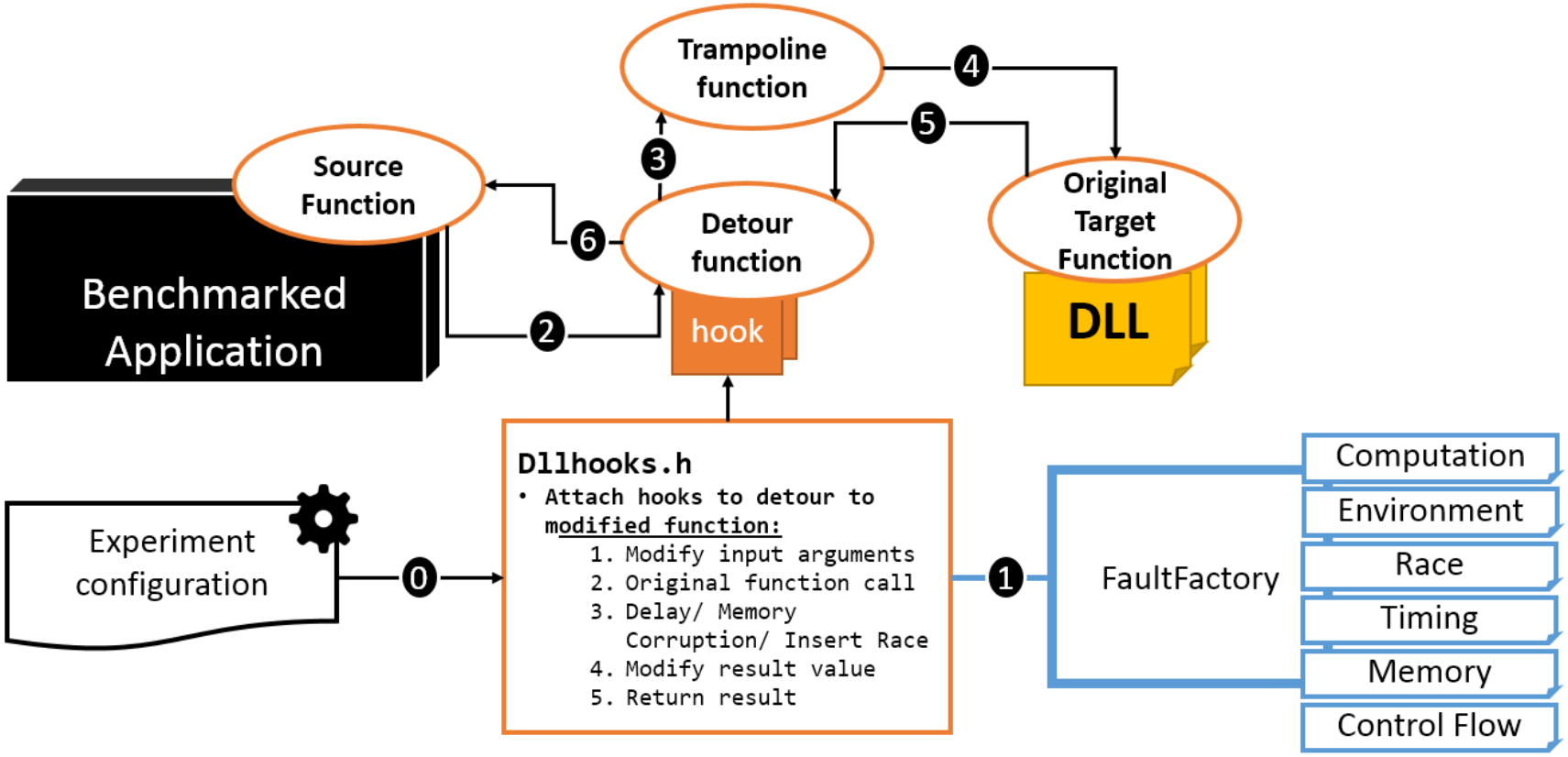
Hovac: A Fault Injection Framework for Benchmarking the Dependability of C/C++ Applications

Basic Idea

- implemented in C++11
- using *Detours* library for hooking DLL calls
  G. Hunt and D. Brubacher, "Detours: Binary interception of win32 functions," in Third USENIX Windows NT Symposium. USENIX, July 1999, p. 8.
- other hooking tools (e.g. Microsoft Hotpatching facilities)could be included with little effort
  J. Passing, A. Schmidt, M. von Löwis, and A. Polze, "NTrace: Function boundary tracing for Windows on IA-32," in Reverse Engineering, 2009. WCRE'09. 16th Working Conference on. IEEE, 2009, pp. 43–52.

1. configure relevant functions calls
2. generate C++ code for Detours hooking
3. link to error state generating DLL

Basic Idea


Trampoline function
4
5
3
Source Function
Original Target Function
6
Detour function
Benchmarked Application
2
hook
DLL
Dllhooks.h
• Attach hooks to detour to modified function:
1. Modify input arguments
2. Original function call
3. Delay/ Memory Corruption/ Insert Race
4. Modify result value
5. Return result
Experiment configuration
0
1
FaultFactory
Computation
Environment
Race
Timing
Memory
Control Flow

Fault Classes Implementation

Computation

- arguments are simply manipulated before the original function call, or the return value before it is returned to the application
- implemented as template classes, instantiated with many supported data types, as to allow for a broad range of arguments

Fault Classes Implementation

Environment

- environment variables are modified or a random signal is emitted to test how well signal handling is implemented
- need to be configured by the user

Timing

- delays are artificially created by waiting for a certain time before or after the original function call

Fault Classes Implementation

Race Condition

- reference to a variable or memory area which will be affected by the race condition needs to be passed to the fault implementation
- variable will be modified concurrently to the execution of the original function call (i.e., in a separate thread)

Fault Classes Implementation

Memory

- memory which is to be corrupted needs to be passed to the fault implementation
- too application-dependent to be implemented generically
  - Uncontrolled Memory Allocation allocates an additional amount of memory whenever it is activated (representative for all kinds of memory leak bugs)

Fault Classes Implementation

Control Flow

- C++ allows developers to throw anything)not just exception type objects)
- arbitrary throwing of an unexpected exception, or other object
- will lead to an unforeseen deviation from the intended control flow

Forwarding Arguments

- computation faults (e.g. OffByOne) need to work for different datatypes (int, unsigned int, …)
- implemented as template classes
- using C++11 features and the *boost* template libraries
  http://theboostcpplibraries.com/boost.any

Example: Fractal Computation

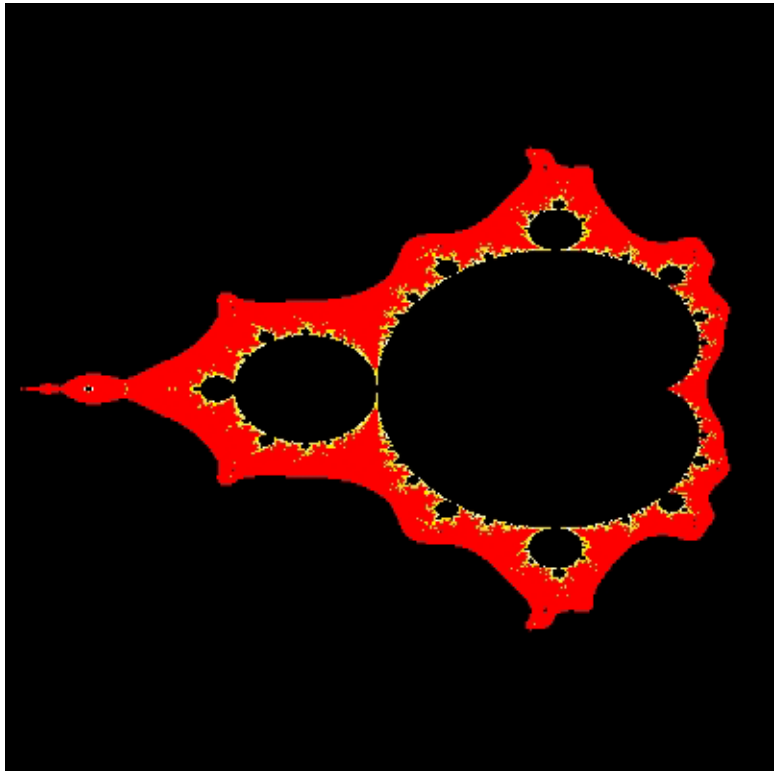
- calls a library to render a fractal into a bitmap image (fractal.dll)
- focus on performance and not robustness, may contain faults
- Hovac: benchmark how well the application can handle errors originating from the fractal library
  - inject errors into all interactions with fractal library

```
<faulttarget>
  <headerfile>
    fractal.h
  </headerfile>
  <signature>
  unsigned int
  calculateMandelbrotIterations
    (std::complex c, unsigned int
          maxiter);
  </signature>
  <callingconvention>__cdecl</
        callingconvention>
  <timing>true</timing>
  <computation>true</computation>
</faulttarget>
```

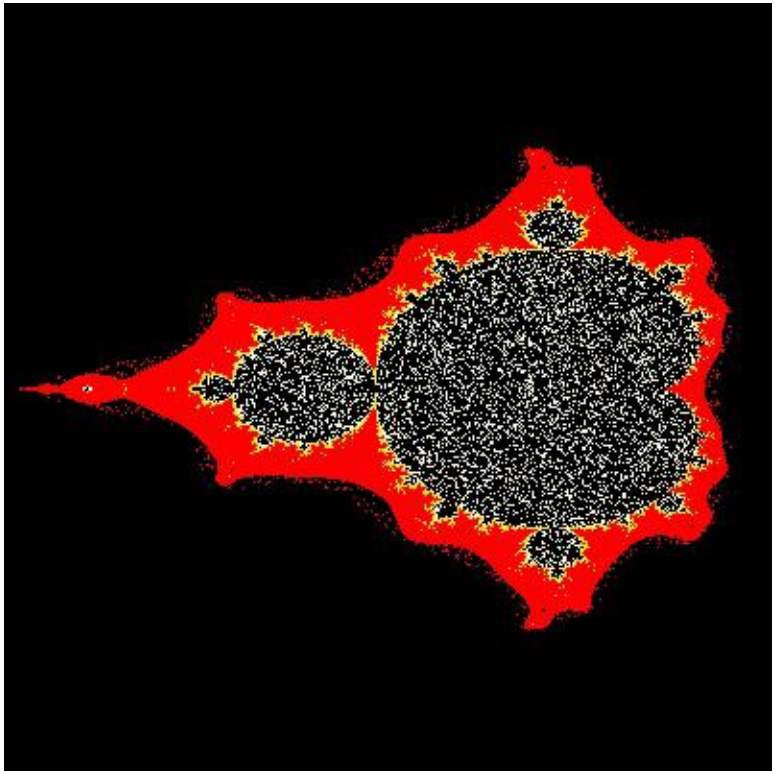
Example: Fractal Computation

HPI Hasso Plattner Institut

correct



under faultload



Example: Fractal Computation


```
class AbstractFault {
public:
  AbstractFault(const std::string cweId = "");
  virtual ~AbstractFault() { tidyUpState(); };

  /*** Custom implementation of fault activation ***/
  virtual Any activate(AnyList arguments = AnyList(), LambdaPtr =
        nullptr) const = 0;

  const ErrorType& getErrorType() const;
  const std::string& getCweID() const;

protected:
  void tidyUpState() = 0;

  Severity m_severity;

  std::string m_cweID;
  ErrorType m_errorType;

  bool m_bCumulative;
};
```

Example: Fractal Computation

- detouring of a function call in dllhooks
- possibilities for fault injection:
  1. modifying the arguments before passing them to the original function call
  2. Injecting a race or timing fault, e.g. delaying the execution by executing the function call lambda late or modifying critical environment variables beforehand
  3. Modifying the return value before passing it back to the application.

HPI Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Dependability Benchmarking Workflow

Hovac: A Fault Injection Framework for Benchmarking the Dependability of C/C++ Applications

Motivation

- correctness vs. robustness
  - correctness: careful coding and an extensive test suite
  - dependability: strongly influenced by robustness.
- systematic workflow to judge the dependability of an application
- tested at the example of the Gravit simulator
  - open source gravity simulation too
  - not only safety-critical but also need to be robust against computational errors
  - makes use of third party libraries
    - Simple DirectMedia Layer (SDL)
    - DirectX

Steps

1. Finding **Locations**
2. Identifying **Fault Classes**
3. **Fault Frequency** and **Severity**
4. Observing **Behaviour** under Faultload

Step 1: Finding Locations

- find dependability "hotspots"
  - application profiling
  - documentation and source code analysis
  - identify some essential functions manually
  - usage of error codes and exception throwing behavior of the functions
- rank by the following criteria:
  1. frequency of function calls,
  2. error-proneness (predicted by code analysis or other sources such as documentation)
  3. criticality of their impact on the benchmarked application

Step 2: Identifying Fault Classes

- specify fault classes per location & distribution
- Hovac: XML-based configuration format
  - if not specified, all fault classes are included for each function call
  - environment faults: separate configuration filewith relevant signals and environment variables
  - function calls with const arguments or no return values, naturally, the computation faults will not have strong effects.
  - default: random fault triggering of error class

Step 3: Fault Frequency & Severity

- how often faults should be injected?
- modify all functions call or only a fraction?
  - how to distribute faults over time?

- Hovac default:
  - inject at least one fault at each function call

Step 4: Observing Behaviour w/ Faultload

- observable failures within a certain time frame?
  - missing coverage of the fault tolerance mechanisms in the application
- uniform way for evaluation & comparison needed e.g. “CRASH”-scale

  P. Koopman, J. Sung, C. Dingman, D. Siewiorek, and T. Marz, “Comparing operating systems using robustness benchmarks,” in Reliable Distributed Systems, 1997. *Proceedings., The Sixteenth Symposium on. IEEE*, 1997, pp. 72–79.
- fault-tolerance vs. recovery-orientation

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Conclusions

Hovac: A Fault Injection Framework for Benchmarking the Dependability of C/C++ Applications

Fault Injection

- injects faults at library call interfaces
- supports different fault classes, categorized by their resulting error states
- realistic faultload: error state-oriented classification of fault implementations based on the CWE database
  - knowledge and experience of both researchers and industrial practitioners
- abstract fault interface can comfortably be extended to accommodate more diverse types of faults

Dependability Benchmarking Workflow

- Steps:
  1. Finding Locations
  2. Identifying Fault Classes
  3. Fault Frequency and Severity
  4. Observing Behaviour under Faultload
- not fully automated, but a step towards convenient, portable, and representative dependability benchmarking

Future Work

- automatically determining a realistic fault model
- more extensive studies using the tool applied to a broader range of software products
- possibilities of building reliability tools based on the CWE database should be explored further

Summary

Hovac

- configurable tool for dependability benchmarking
- DLL API hooking is portable and universally applicable for testing software reliability when using third-party libraries
- fault classes: based on the Common Weakness Enumeration database
- detailed and systematic workflow to benchmark the dependability of (C/C++) applications

- open source: **https://github.com/laena/hovac**

Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Hovac: A Fault Injection Framework for Benchmarking the Dependability of C/C++ Applications

Lena Herscheid, Daniel Richter, and Andreas Polze

Operating Systems & Middleware Group
Hasso Plattner Institute at University of Potsdam, Germany