Towards Understanding the Importance of Variables in Dependable Software

Matthew Leeke and Arshad Jhumka
Department of Computer Science
University of Warwick
{matt, arshad}@dcs.warwick.ac.uk
Outline

๏ Software Dependability
๏ Mechanisms for Error Detection and Correction
๏ Related Work
๏ The Importance Metric
๏ A Case Study: FlightGear
๏ Discussion and Limitations
๏ Future Work
Computer systems have become pervasive

Functionality increasingly defined by software

- Software dependability is critical across all facets of modern society

- A strong motivator for much current work in software dependability

How do we actually design for dependability?

- A firm conceptual basis
A dependable software system contains two important types of artefact [Arora et al., 1998]:

- Error Detection Mechanisms (e.g. executable assertions)
- Error Recovery Mechanisms (e.g. exception handlers)

In order to contain errors and avoid system failures both artefacts must be effective:

- Coverage
- Latency
Error Detection Mechanisms (EDMs)

- An EDM seeks to detect whether the system state at a given point can threaten the proper functioning of a software system.

- Effectiveness depends on location and predicate:
  - Implemented predicate is essentially a boolean expression defined over a set of program variables.
  - This predicate is non-trivial [Jhumka et. al., 2006]
  - Properties of non-triviality are accuracy and completeness.
  - Accuracy and completeness are sensitive to the set of variables.
Error Recovery Mechanisms (ERMs)

- An ERM seeks to restore a suitable safe state for a software system in order for it to continue execution.
  - Values of some corrupted variables must be overwritten.
- If correct and timely operation is to be maintained, some set of variables need to hold suitable values in order to make the system state safe for further execution.
- If this set of critical variables is known, it is easier to develop predicates and determine appropriate locations for EDMs and ERMs.

Knowledge of critical variables eases the design and placement of EDMs and ERMs.
Our Approach

- We develop a variable centric approach to understand and capture the importance of variables in an effort to contain error propagation.

- We present:
  - Two metrics which contribute to a definition of Importance
  - The Importance Metric
  - An experimental approach for estimating Importance
  - A case study to demonstrate the application of these metrics
Related Work

๏ Experimentally evaluating the coverage and latency of EDMs and ERMs [Arlat et. al.,1989] [Vinter et. al,2001]

- Established that EDMs exhibiting high coverage and low latency serve to reduce error propagation

๏ ERM/EDM Location

- Subsequent research on locating EDMs [Jhumka et. al.,2005]

- Guidelines and heuristics for the location and some aspects of design [Hiller et. al.,2000] [Rabejac et. al.,1996]

- Metrics for identifying modules which do not propagate errors [Khoshgoftaar et. al,1999]
- Quantifying error propagation / influence between interacting modules [Jhumka et. al., 2001] [Suri et. al., 1998]

- Framework for identifying vulnerabilities in software based upon error permeability [Hiller et. al., 2004]
  - Captures how likely errors are to propagate from a module input to its output, allowing the identification of permeable modules
  - Requires accurate data flow information

- **ERM/EDM Design**
  - Specification used to derive programatic tests which capture some aspects of functional correctness [Richardson et. al., 1992]
Related Work

- Static analysis to detect vulnerabilities [Wilken et. al., 1990]
  - Completeness
  - False positives

- Finite state programs considered in the automated design of EDMs
  [Arora et. al., 1998] [Jhumka et. al., 2006]
  - Applicability of analysis on finite state systems

- Little work relating specifically to the design of predicates

- In some sense the described approach is asking “What variables should be captured by the implemented predicate?”
Models

๏ System Model

- We consider a software to be a set of interconnected components, though we do not know anything about the interconnections.

- We adopt a grey-box view => Access to the source code is allowed, but knowledge of specific functionality or structure is not available.

๏ Fault Model

- Transient value fault model

- Model hardware faults as bit flips which cause instantaneous changes to values held in memory.
The Importance Metric

- Essentially related to measuring the impact that corruption in a particular variable can have in two domains:
  - Spatial
  - Temporal

- Intuitively, to minimise the likelihood of software failure, each aspect needs to be appropriately handled:
  - The number of corrupted variables, hence modules, and the duration of the corruption need to be minimised.
The spatial impact of variable $v$ of component $C$ in a run $r$ is the number of components that get corrupted in $r$.

$$\sigma_{v,C}^r$$

The spatial impact of variable $v$ of component $C$ is the maximum of all runs.

$$\sigma_{v,C} = \max\{\sigma_{v,C}^r\}, \forall r$$
The temporal impact of variable \( v \) of component \( C \) in a run \( r \) is the number of time units over which at least one component remains corrupted in \( r \):

\[
\tau_{v,C}^r
\]

The spatial impact of variable \( v \) of component \( C \) is the maximum of all runs:

\[
\tau_{v,C} = \max \{ \tau_{v,C}^r \}, \forall r
\]
Considerations:

1. A variable may have a high spatial impact, which means that by the time the error disappears, several other components have been corrupted, thus limiting the effectiveness of a recovery.

2. A variable may have a high temporal impact but a low spatial impact, which means that, even though very few components are affected, a recovery may not be effective.
Calculating Importance

- A general form for a metric which accounts for the described factors in expressing the importance of a variable \( v \) in a component \( C \), using arbitrary function \( G \), \( K \) and \( L \), can be taken to be:

\[
I_{v, C} = G[K(\sigma_{v,C}), L(\tau_{v,C})]
\]
Calculating Importance

- Any instantiation should be as rich as possible with respect to the goals of error containment and failure avoidance.
- Account for factors which influence importance, but are not directly captured by the defined impact metrics.
  - Incorporating failure rate helps to realise these goals.
Calculating Importance

\[ I_{v,C} = \frac{1}{(1 - f)^n} \left( \frac{\sigma_{v,C}}{\sigma_{max}} + \frac{T_{v,C}}{T_{max}} \right)^m \]

- Normalisation of the impact metrics is performed to ensure that their addition does not mask or enhance either impact metric.

- Applicable where emphasis is to be placed upon the need to detect errors or to recover from them.

  - The values of \( n \) and \( m \) dictate whether emphasis is placed upon the need to avoid failures or the need to prevent widespread system corruption.
A Case Study - FlightGear

- **Target System**
  - Open-source flight simulator
  - Over 220,000 lines of C/C++

- **Test Cases**
  - Takeoff procedure of 2700 simulation loop iterations
  - 500 iteration initialisation and
  - 2200 iteration pre- and post-injection periods
  - 9 test cases; 3 aircraft masses and 3 sets of environmental conditions
A Case Study - Fault Injection

- **Instrumentation**
  - Instrumented modules randomly selected
  - Exhaustive analysis with respect to bit representation and code location

- **Fault Injection and Logging**
  - Modified PROPANE used to perform fault injection [Hiller et. al., 2002]
  - Modifications allowed almost full automation and analysis of 3,773,736 experiments

- **Characterising Erroneous State and Failure**
  - Deviation from a golden run characterised error states
  - Failure specification based upon expert knowledge
### A Case Study - Results

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Module</th>
<th>Fail Rate</th>
<th>Spatial Impact</th>
<th>Temporal Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>fgFlightTime</td>
<td>G</td>
<td>0.003472</td>
<td>4</td>
<td>2000</td>
</tr>
<tr>
<td>delta_time_sec</td>
<td>G</td>
<td>0.006944</td>
<td>3</td>
<td>2000</td>
</tr>
<tr>
<td>dump</td>
<td>C</td>
<td>0.013889</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>EmptyWeight</td>
<td>D</td>
<td>0.011905</td>
<td>2</td>
<td>2000</td>
</tr>
<tr>
<td>initialisedEngines</td>
<td>C</td>
<td>0.001389</td>
<td>5</td>
<td>77</td>
</tr>
<tr>
<td>sBound</td>
<td>C</td>
<td>0.004167</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Mass</td>
<td>D</td>
<td>0.011905</td>
<td>12</td>
<td>1432</td>
</tr>
</tbody>
</table>

**Table 1 - Components of the Importance metric**
<table>
<thead>
<tr>
<th>Identifier</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentThrust</td>
<td>1.047348</td>
</tr>
<tr>
<td>HasInitialisedEngines</td>
<td>1.016663</td>
</tr>
<tr>
<td>numTanks</td>
<td>1.012560</td>
</tr>
<tr>
<td>TotalQuantityFuel</td>
<td>1.011618</td>
</tr>
<tr>
<td>firsttime</td>
<td>1.009914</td>
</tr>
<tr>
<td>dt</td>
<td>0.506376</td>
</tr>
<tr>
<td>Tw</td>
<td>0.090196</td>
</tr>
<tr>
<td>numEngines</td>
<td>0.068625</td>
</tr>
<tr>
<td>_run_3</td>
<td>0.058308</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>1.048938</td>
</tr>
<tr>
<td>EmptyWeight</td>
<td>1.030808</td>
</tr>
<tr>
<td>bixx</td>
<td>1.008410</td>
</tr>
<tr>
<td>bixy</td>
<td>1.008410</td>
</tr>
<tr>
<td>bixz</td>
<td>1.008410</td>
</tr>
<tr>
<td>bizz</td>
<td>1.008160</td>
</tr>
<tr>
<td>biyz</td>
<td>1.008160</td>
</tr>
<tr>
<td>biyy</td>
<td>1.007966</td>
</tr>
<tr>
<td>Mass</td>
<td>0.772751</td>
</tr>
</tbody>
</table>

*Table 2/3 - Importance metric values for Module C/D*
Table 4 - Overall Importance Ranking For All Modules

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Module</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>D</td>
<td>1.048938</td>
</tr>
<tr>
<td>currentThrust</td>
<td>C</td>
<td>1.047348</td>
</tr>
<tr>
<td>EmptyWeight</td>
<td>D</td>
<td>1.030808</td>
</tr>
<tr>
<td>delta_time_sec</td>
<td>G</td>
<td>1.023784</td>
</tr>
<tr>
<td>fgFlightTime</td>
<td>G</td>
<td>1.019890</td>
</tr>
<tr>
<td>HasInitialisedEngines</td>
<td>C</td>
<td>1.016663</td>
</tr>
<tr>
<td>numTanks</td>
<td>C</td>
<td>1.012560</td>
</tr>
<tr>
<td>TotalQuantityFuel</td>
<td>C</td>
<td>1.011618</td>
</tr>
<tr>
<td>firsttime</td>
<td>C</td>
<td>1.009914</td>
</tr>
</tbody>
</table>

A Case Study - Results
We might envision the relative Importance ranking being used to inform the development and maintenance of software:

- A software release can not address all known issues
- Address vulnerabilities in order of severity

Importance does not require knowledge of software structure or communication paths:

- Analysis can be performed post-implementation by an engineer with no prior system knowledge
- Consistent with modern software development methods (unfortunately)
Lessons Learnt

๏ A variable centric approach can be employed in guiding the design and placement of EDMs and ERMs

๏ The proposed metric can identify critical variables which are not immediately evident, even to those with prior system knowledge

  - More on this to come

๏ The Importance metric and its constituent components can be evaluated in an automated fashion, thus facilitating a low-cost dependability analysis

  - Automation is a key facilitator for this form of analysis
Limitations

- The relative ranking generated is sensitive to the set of variables under consideration
  - Ideally all variables in a software would be analysed
  - Value in analysing on a module-by-module basis

- Importance metric does not attempt to estimate / quantify the real-world importance of variables

- Inherent limitations of the fault injection process
  - Intrusiveness, test case dependance and potential for unknown sources of variability
Future Work

- Validating identified critical variables
  - Blind test... interesting!
  - Correlating with predicates derived by mining FI data
- Approaches to EDM and ERM design based upon identified critical variables
  - Critical variables only?
- Relating access restrictions to variable importance
We develop a variable centric approach to understand and capture the importance of variables in an effort to contain error propagation.

We present:
- Two metrics which contribute to a definition of Importance
- The Importance metric
- An experimental approach for estimating Importance
- A case study to showcase these metrics
References


References


References


Questions?