Impact Analysis of Software Change for Mission Critical Systems

Damian Kennedy, Dr Sergey Nesterov

Centre of Excellence for Defence and Industry Systems Capability
Defence and Systems Institute
University of South Australia
Mawson Lakes Campus
Mawson Lakes 5095
Outline

- Introduction
- Identified Methods to Ease Impact Analysis
  - Behavior Trees
  - Model Based Integration of Embedded Systems
  - Model Based Verification
  - Mission Orientated Architectural Evolution
  - Architecture Analysis and Design Language
  - Diagram Approach
- Suitability of Methods to the current AP-3C Software Maintenance Process
- Conclusion
Introduction (1)

- Earlier a defect is found the cheaper it is to correct.

- In general, defects found during testing or after release can cost between 10 to 100 times more to fix (McConnell 1996, Pressman 2001, Tian 2005, Sommerville 2007)

- Impact analysis is an effective way to establish the extent of a change, and mitigate the risk of defects during testing

Change request process. Adapted from (Sommerville 2007)
Introduction (2)

- Impact Analysis typically includes:
  - New requirements analysis and validation
  - List of source code files affected
  - Estimated scale of documentation updates
  - Estimated impact on operation procedures and operator’s workload
  - Estimated scale of testing required
  - Estimation of resources required for change implementation
  - Analysis of cost benefits for the requested change from the maintainers and customers perspective
  - Work Breakdown Structure and estimated schedule
Introduction (3)

- Outsourced maintenance and Impact Analysis complicated by:
  - Documentation is incomplete or missing
  - Staff are not familiar with the application domain
  - Few maintenance professionals with experience due to difficulty in retaining staff in the environment
  - Insufficient plans for transfer of knowledge from developers to maintainers
  - Lack of strategic corporate plans for software maintenance
Introduction (4)

- Embedded systems further complicated
  - Subtle dependencies rarely documented
  - Latent errors only discovered during integration

- Traditionally Impact Analysis
  - Done through “best efforts” of experienced SW engineers after cursory examination of code and documentation
Behavior Trees (1)

- Developed by SQI, Griffith University
- Aims to develop a design from the functional requirements
- Provides traceability between requirements and the design
- Constructs a BT for each requirement by:
  - Identifying components and states
  - Capturing behavior expressed
- Individual BT’s integrated together using the following axiom’s (Dromey, R.G., 2003):
  - Precondition Axiom
  - Interaction Axiom
Behavior Trees (2)

- Modified version has been used with a system with changing requirements (Lian and Dromey, 2004)
MOBIES (1)

- U.S. Defence Advanced Research Projects Agency program
- Aimed towards developing a complete end-to-end tool chain for a model based approach to embedded system development (Zonghua et al., 2003)
- Focus on analysis of non-functional aspects (i.e. timing and scheduling)
- Key areas (Schulte, 2005):
  - Multi-view modelling
  - Model-based analysis
  - Code generation
MoBIES end-to-end tool chain (Zonghua et al, 2003)
Model Based Verification (1)

- Developed by the Software Engineering Institute at Carnegie Mellon University

- Set of processes to help reduce number of software errors and increase quality of upgrades to mission critical systems (Gluch and Weinstock, 1998)

- Combination of modelling, formal methods, and model checking

- Emphasis on detecting presence of errors as opposed to providing formal proofs of the system

- Driving concept to “… create simplified models of the critical (important and risky) parts of a system rather than detailed models of the complete system” (Gluch and Brockway, 1999)
Model Based Verification (2)

- Variety of formal modelling and analysis techniques used
  - Developer selects those best suited to the system
- Focus on applying “... emerging model-based techniques and related support tools to improve the quality of software upgrades” (Gluch and Weinstock, 1998)
MORALE (1)

- Mission Orientated Architectural Legacy Evolution
- Targeted towards legacy system evolution
- Designed to address the following concerns (Rugaber, 1999)
  - Making sure the new requirements are consistent with those of the existing version
  - Maintaining control of the system architecture
  - Understanding code of the current version
  - Suggesting how enhancement may be made without compromising the integrity of the design
MORALE (2)

- Treats system evolution as a process:

**INPUTS**
- existing system
- design documents
- new requirements

**OUTPUTS**
- new system
- updated requirements
- updated design documents

**Can be broken into three activities (Abowd et al, 1997):**
- Requirements Analysis
- Architectural Extraction
- Change Impact Analysis

**Supported by Espirit de Corp Suite (EDCS) of tools**
MORALE (3)

- SAAM – Requirements Analysis
  - Predict system level quality attributes through architecture analysis
  - Uses scenarios from different perspectives (users, maintainers)
  - Process revolves around group discussion

```
Scenario Development \(\xrightarrow{\text{iterate}}\) Architecture Description
and
Individual Scenario Evaluation \(\xrightarrow{\text{or}}\) Overall Evaluation
Assess Scenario Interaction
```

SETE 2007 Sydney
AADL (1)

- Architecture Analysis and Design Language
- SAE Standard (SAE AS 5506)
- Architectural modelling language
- Developed for real-time fault-tolerant embedded systems
- Describes structure of system in terms of
  - Hardware components
  - Software components
- Supports functional and non-functional aspects of components and interactions
AADL (2)

- **AADL Error Model Annex**
  - Developed to support analysis (qualitative and quantitative) of dependability attributes
  - Main difficulties in building dependability models is component interdependencies
  - GSPNs can be developed from AADL model
Diagram Approach (1)

- Uses standard diagrams to understand change impact

- Assumption:
  - Understanding impact of previous change requests will assist in understanding impact of new change requests

- Uses 3 main types of diagrams:
  - Structural Diagrams
  - Behaviour Diagrams
  - Interaction Diagrams
Diagram Approach (2)

- Approach trialed
  - Entire system modeled using each diagram type
  - Changes to each diagram caused by CR marked
  - Marked-upped diagrams analysed

- Results compared to impact analysis estimates by experienced software engineers

- Compared to SE estimates, Diagram Approach had higher accuracy

- Reasons:
  - SE based estimates on partial view of system (class diagrams)
  - Estimates limited by partial view
Analysis (1)

- Difficult to determine specific judgment criteria
- All methods have their merits, and could be useful
- Main gap in capability: impact at requirements level
Analysis (2)

- Behavior Trees:
  - Aimed at functional requirements
  - Only suited for changes that affect requirements

- AIRES:
  - Part of a larger chain of methods
  - Determines extent of impact at SW component level
  - Makes use of dependency graphs

- Diagram Approach:
  - Able to determine scope of change
  - Possibility to determine extent of regression testing required

- AADL:
  - Focus on describing HW and SW components and their interactions
  - Can describe functional and non-functional system aspects
  - Models can be converted to Generalised Stochastic Petri Nets and Markov Chain models for further analysis
Conclusion

■ All approaches described would be suitable for different aspects of the change analysis process
■ Each would provide improvement in performance when compared to the current practice
■ Most applicable determined to be Behavior Trees:
  ■ Are able to focus on impact on the functional requirements
  ■ Assess the completeness of functional requirements
  ■ Possibility to extend its uses as work is being done to introduce non-functional requirements
Questions
References


Dromey, RG 2003, 'From requirements to design: formalizing the key steps', paper presented at the First International Conference on Software Engineering and Formal Methods.


Sommerville, I 2007, 'Software Engineering'.


Glossary / Acronyms

- AADL – Architectural Analysis Design Language
- BT – Behavior Tree
- CR – Change Request
- EDCS – Espirit de Corp Suite
- GSPN – Generalised Stochastic Petri Net
- HW – Hardware
- MBV – Model Based Verification
- MOBIES – Model-Based Integration of Embedded Software
- MORALE – Mission Orientated Architectural Evolution
- SAAM – Structured Architectural Analysis Method
- SAE – Society for Automotive Engineers
- SE – Software Engineer
- SW – Software
- SQI – Software Quality Institute
- TAD – Tenix Aerospace and Defence
- U.S. – United States
Applicability (2)

- **Behavior Trees:**
  - Aimed at functional requirements
  - Only suited for changes that affect requirements

- **AIRES:**
  - Part of a larger chain of methods
  - Determines extent of impact at SW component level
  - Makes use of dependency graphs

- **SAAM:**
  - Only useful when used as part of the MORALE tool chain

- **MBV:**
  - Appears fairly generic: models required not specified
  - Provides framework to tie different system models together
  - Little scope for a performance improvement over current process

- **Diagram Approach:**
  - Able to determine scope of change
  - Possibility to determine extent of regression testing required

- **AADL:**
  - Focus on describing HW and SW components and their interactions
  - Can describe functional and non-functional system aspects
  - Models can be converted to Generalised Stochastic Petri Nets and Markov Chain models for further analysis