Towards a Software Engineering Meta-Environment

Arndt von Staa
Departamento de Informática
PUC-Rio
September 2010
What is the problem?

We want to develop software that is dependable from the user’s point of view. And do this economically, i.e. with little amount of useless rework.
What is the problem?

- Quality definition fallacy
  - Typical definition: “A system is correct if it implements exactly what has been specified.”
  - What if the specification does not correspond to what the user needs, wants or wishes?
- \( \approx 80\% \) of software cost is due to maintenance
  - Evolution of information systems (new features) \( 65\% \)
  - Adapting to new rules, laws, platforms, etc. \( 18\% \)
  - Correcting \( 17\% \)
- Maintenance like actions are performed also while developing
  - especially when performing incremental development
  - (old statistic) about 70\% of the defects removed are due to wrong specifications

Nosek, J.T.; Palvia, P.; “Software Maintenance Management: changes in the last decade”; Software Maintenance: Research and Practice 2(3); 1990; pp 157-174
What is the problem?

- Software intensive systems **must be dependable**
  - some requirements
    - **adequacy**: always does what the user needs, wants, or wishes
    - **reliability**: always does it correctly
    - **robustness**: observes malfunction or incorrect use at execution time
    - **maintainability**: the collection of artifacts can be **co-evolved** while maintaining at an acceptable cost
    - **integrateability**: the collection of artifacts can be correctly integrated whenever needed
    - **verifiability**: the collection of artifacts can be verified, validated and approved whenever desired and at an acceptable cost
    - **reusability**: libraries, components, frameworks, models, services, ... as long as they are dependable
    - ...

Thomas, D.; “The Deplorable State of Class Libraries”; *Journal of Object Technology* 1(1); Zürich, CH: ETH Zürich; 2002; pages 21-27
Modeling development and maintenance

- Software development and maintenance can be viewed as a knowledge acquisition process
  - knowledge about the problem to be solved (requirements)
  - knowledge about the solution (technology)
- Knowledge is kept in representations
  - documents, specifications, models, code, test scripts, ...
- The amount of knowledge tends to be huge
  - divide and conquer leads to establishing levels of abstraction
  - the variety of knowledge categories leads to partitioning according to views: structural, functional, dynamic, aspectual, ...
- The collection of representations forms a system
  - a change to one representation may induce the need to change adjacent ones

Armour, P.G.; *The Laws of Software Process: A New Model for the Production and Management of Software;* Taylor & Francis; 2003;
Modeling development and maintenance

• But representations are too coarse grained
  – each representation is composed of (small) fragments that may be shared by several representations

• The knowledge about software can be viewed as a graph consisting of a large number of strongly connected (interdependent) representation fragments
Representations are strongly interdependent

The collection of representations corresponds to a system of representations.
Abstract actions performed on representations

- **Create** representation
- **Transform** a representation: predecessor to successor
  - from a higher level to a lower level of abstraction
  - from a view (language) to another view
- **Reflect** changes: successor to predecessor (reverse engineering)
- **Refactor** a collection of representations (reengineering)
- **Add** information (details) to a representation
- **Modify** a representation
- **Extract** information
- **Control quality** of individual representations and of collections
- **Consolidate** – assure that the collection is consistent
- **Navigate** from one to another representation
Representations form a hyper-document
Co-evolution – a crude procedure

1. Change a representation
2. Verify the representation, if not OK
   • correct the representation and repeat
   • or create an issue to be handled later
3. Validate the representation with respect to all interdependent ones, if not OK
   • for all representations that became inconsistent repeat from 1
   • or refactor the collection and for all affected repeat from 1
   • or create a issues to be handled later
4. Occasionally approve a collection of representations, if not OK, or if change requests or issues are to be handled
   • search the starting representations that must be changed to accommodate the change requests or issues and for all of them repeat from 1

changes ripple through the collection of representations until a new stable state is reached

What is the cost of doing this systematically
Navigating a hiper-document

- Select target representation
- With focus on source object display representation using rules

- Rep I
  - Object A
- Rep J
  - Object A
- Rep K
  - Object A
- Rep L
  - Object A

Dictionary

Object A

Software base
Diagrama Fluxo de Dados

Nome: xxx
Instância de: processo
Processo: 123
Posição: 30 100

Base de software

Processo: 123
Nome: "Obter dados"
Aliás 3: 2.1
Relação Agentes: João
Maria
José

Base de definição

Carimbo de: processo
Moldura:

- Campo 1: Aliás 3
  moldura _________
- Campo 2: Nome
- Campo 3: Relação Agentes
  moldura _________
Dimensões: 12 x 8

Vídeo (Folha)

área ocupada

João
Interrelated meta-editors
3 level meta-specification

Diagram

Software base schema
- Label
- Adornment
- Instance
- Link

Parameter base schema
- Software base static schema
- Definition base static schema

Definition base schema
- Instance classes
- Link rules
- Link classes
- Label classes
- Label rules
- Adornment classes
- Adornment rules

belongs to

obeys
Research agenda – meta-environment

Specifier & Reviewer

External specification

Specification mark up

Design interface

Develop component models

Test case selection criterion

Generate test cases

Manual test cases

Interact with other components

Perform static analysis

Develop component

Component

Test log & findings

Interface sketch

Other data dictionaries

Data dictionary / onthology

SWB

Manual test cases

Test component

Automated test scripts

Tool
State of progress
Epilogue

• Research agenda
  – Develop a software engineering meta-environment
    • distributed specification, design, development and quality control capabilities
    • will act as an experiment factory
  – Develop and perform experiments with a variety of representation languages
    • transformations, reflections (reverse engineering)
    • generation
    • co-evolution
    • domain specific language integration with environment
    • assess while developing real systems
    • develop itself
      – export from Talisman 4.4 to Talisman 5.0
      – continue developing with 5.0, 5.1, ... 5.N, 6.0
      – learn and enhance while using it
Towards a Software Engineering Meta-Environment

Arndt von Staa
 arndt em inf.puc-rio.br

Departamento de Informática
 PUC-Rio
 September 2010