An empirical case study in software evolution:
When refactoring becomes necessary?

Naji Habra – Miguel Lopez - PRECISE Research Group
Agenda

• Empirical studies in Software Evolution
• A case study
• Lessons – Questions
Empirical studies in software evolution

- Empirical studies = major challenge in software evolution

  - ex. In IWPSE 2005
  - “empirical studies” identified as a challenge
  - underlies other challenges & needs:
    - analyzing huge data
    - need predictive models
    - benchmarking

  - ➔ the ultimate goal is to control (manage) the evolution process
Empirical studies in software evolution

- Existing works (empirical studies in Soft. Evol.)
  - Prospective - Predictive use
    - identify the parts of the software to be restructured (where)
    - identify the time restructuring becomes necessary (when)
    - predict effort needed for refactoring
  - Retrospective use
    - Where restructuring has been done
    - How much a given “quality” was improved
      ex. Impact of a given “refactoring” on the quality (Fowler..)
- Theory elaboration / validation
  - Towards a General theory for evolution
  - Validate/refine (some of) Lehman’s “Laws”
A case study - 1

• The approach = Goal-Questions-Metric
  • A top-down methodology for empirical study to select ad-hoc metrics on basis of higher level goals
    • Goal (Conceptual level)
    • Questions (Operational level)
    • Metrics (Quantitative level)
  • !!!!! GQM contributes but does not replace
    • Theory elaboration
    • Model construction

• The experimentation
  • SPIP (Open Source) – 9 releases – Php code -
A case study: Goal

1- The Goals

G1: Validate the hypotheses

- Hypothesis-1: software structure deteriorates over time & releases *
- Hypothesis-2: at a precise time $t$ refactoring becomes necessary

G2: (Find a hint to) determine the time $t$ (if any)

(*)

- Lehman II: increasing complexity (unless work is done…)
- Lehman VII: declining quality (unless work is done…)
A case study : Questions

2- The Questions

- what is well-structured software?
  - Candidates
    - Coupling ? (which coupling ?)
    - “good” use of O.O. mechanisms (inheritance / encapsulation..) ?
    - Use of some “patterns” ?
    - ..... 

- how this attributes evolve through the succession of releases?
  - linear ?
    - particular identified acceleration/stabilization point ?
  - where that attribute becomes too bad to require a refactoring?
    - Threshold ??
A case study: Metrics

• 3- The Metrics

• Model (representation) at the files level
  • Software structure as a graph
    • nodes are files
    • edges represent the “#include file” relationship

• Metrics
  • #nodes
  • #edges
  • indegree-outdegree of nodes
  • cyclomatic complexity ..
A case study: first results

<table>
<thead>
<tr>
<th></th>
<th>#nodes</th>
<th>#edges</th>
<th>density</th>
<th>CCN</th>
<th>#nodes fi = 0</th>
<th>%nodes fi = 0</th>
<th>%nodes fi &gt;0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIP-1.0.4</td>
<td>83</td>
<td>64</td>
<td>0.94%</td>
<td>146</td>
<td>49</td>
<td>59%</td>
<td>41%</td>
</tr>
<tr>
<td>SPIP-1.1 all</td>
<td>85</td>
<td>66</td>
<td>0.92%</td>
<td>150</td>
<td>51</td>
<td>60%</td>
<td>40%</td>
</tr>
<tr>
<td>SPIP-1.2 all</td>
<td>93</td>
<td>72</td>
<td>0.84%</td>
<td>164</td>
<td>56</td>
<td>60%</td>
<td>40%</td>
</tr>
<tr>
<td>SPIP-1.3 all</td>
<td>100</td>
<td>73</td>
<td>0.74%</td>
<td>172</td>
<td>64</td>
<td>64%</td>
<td>36%</td>
</tr>
<tr>
<td>SPIP-v1-4pr5 all</td>
<td>122</td>
<td>93</td>
<td>0.63%</td>
<td>214</td>
<td>77</td>
<td>63%</td>
<td>37%</td>
</tr>
<tr>
<td>SPIP-v1-5pr4</td>
<td>125</td>
<td>94</td>
<td>0.61%</td>
<td>218</td>
<td>74</td>
<td>59%</td>
<td>41%</td>
</tr>
<tr>
<td>SPIP-v1-6</td>
<td>138</td>
<td>94</td>
<td>0.50%</td>
<td>231</td>
<td>74</td>
<td>54%</td>
<td>46%</td>
</tr>
<tr>
<td>SPIP-v1-7</td>
<td>191</td>
<td>107</td>
<td>0.29%</td>
<td>297</td>
<td>88</td>
<td>46%</td>
<td>54%</td>
</tr>
<tr>
<td>SPIP-v1-8</td>
<td>279</td>
<td>110</td>
<td>0.14%</td>
<td>388</td>
<td>94</td>
<td>34%</td>
<td>66%</td>
</tr>
</tbody>
</table>
A case study: results

The evolution of complexity through versions

- #nodes
- #edges
- complexity
A case study: results

The evolution of IN+ & IN0 through versions

% vertics indegree = 0

% vertics indegree > 0

SPIP-1.0.4 SPIP-1.1 all SPIP-1.2 all SPIP-1.3 all SPIP-v1-4pr5 SPIP-v1-5pr4 SPIP-v1-6 SPIP-v1-7 SPIP-v1-8
First conclusion

- H1 : Software structure deteriorates over time & releases
  H2 : A precise time $t$ (release) refactoring becomes necessary

- Too early to conclude
  - more experimentation $\rightarrow$ statistical evidence
  - + more fundamental questions remain

- Case study as exemplary work
  - It involves several “questionable” choices
    $\Rightarrow$ several questions
    $\Rightarrow$ challenges of empirical works in this area
Question -1: which artifact

- The choice of the model (the representation) of the “software” as base to measure/assess
  - Here: Software = a graph of files with dependency, but
  - different levels of abstractions (classes / files / packages/…)
    *the need to “think” evolution at higher level
    *code analysis not sufficient to detect design drift
  - different products at the same level
    ex. class diagrams + seq diagrams (*co-evolution*)

- The choice of the product (the artifact being measured)
  - code / high level design (diagrams) /
    whole software as a compounded artifact
  - → The need of consensual view (standards)
Questions – 2 : which property(ies)

• The definition of the property : “well-structured”?
  • Here : structure = dependency between files
  • But other candidates
    • Coupling (which kind ?)
    • Complexity (which complexity ?)
    • Relationship “size” - “complexity”

• ➔ The need of consensual view (standards)
Questions-3 : model the evolution process

• What about the other factors ?
  • Factors under/not under experimentation control: structural factors/environment -
  • Influence of the size
    • Evolution = (add functionality + restructuring + …)
    • Hypothesis = functionality increase is regular ?
  • Influence of the software family
    • E-type or not
    • Open source or not
  • Influence: the development process.
  • ➔ The need of a complete quality model (theory)
  • ➔ satisfactory predictive models
Conclusion

Challenges

• 1) More empirical studies with reduced scope
   ( → !!! results to use only within the scope)
   • Ex: Open source + some characteristics (evolvability)
   • O.O. (Java) + “structural properties” + Gof patterns
   • ...........

• 2) Seek for consensual / standard definition (soft. model & properties)
   • medium term challenge “continuous effort”

• Establishing & validate laws (theory):
   • long term challenge
Thank you!

Any questions?

University of Namur
http://www.fundp.ac.be

Computer Science Institute
http://www.info.fundp.ac.be

PRECISE group
http://www.fundp.ac.be/precise