Evolution Metrics

Tom Mens
Programming Technology Lab
Vrije Universiteit Brussel
Belgium

tom.mens@vub.ac.be

Serge Demeyer
Lab on Re-Engineering
Universiteit Antwerpen
Belgium

serge.demeyer@ua.ac.be

Goals

- Study existing research literature on evolution metrics to

  ① classify existing approaches/techniques based on their *purpose*

  ② compare existing empirical studies to identify open problems and future research trends

Empirical approaches

- have been used to
  - estimate maintenance cost, effort and productivity
  - predict/estimate maintainability, reliability, ...
  - predict improvements/degradations in quality/structure
  - identify trends and change points in evolutionary behaviour
  - identify (un)stable parts of the software
  - identify the need for reengineering/restructuring
  - understand the nature of software evolution processes

Purpose of evolution metrics

- ***Predictive*** versus ***retrospective*** *analysis* of the software
  - ***Predictive*** (*before* evolution)

    use previous and current releases to predict changes in future releases (what, where, how, how much, how big, ...)
  - ***Retrospective*** (*after* evolution)

    compare different releases to
    - understand the evolution (which kind, why)
    - detect improvements/degradations of quality
- ***How*** versus ***what and why*** (cf. Lehman and Ramil)
  - ***What and why***: study nature and properties of evolution
  - ***How***: improve software evolution process

Purpose of evolution metrics

- Classify based on parts of software that are affected
  - ***Evolution-critical** parts* need to be evolved due to
    - poor quality, incomplete code, bad structure, unused code, duplicated code, overly complex code
  - ***Evolution-prone** parts* are likely to evolve
    - correspond to highly volatile software requirements

    (detect by examining release histories)
  - ***Evolution-sensitive** parts* have high estimated change impact
    - e.g., highly-coupled parts

Goals

- Study existing research literature on evolution metrics to

  ① classify existing approaches/techniques based on their *purpose*

  ② compare existing empirical studies to identify open problems and future research trends

Long-term evolution studies

	Ramil&al00	Burd&Munro	Gall&al98	Graves&al00	Perry&al98	Godfrey&al00
software size	**large**	**large** 300 KLOC	very **large** 10MLOC	very **large** 1.5 MLOC	very **large**	very **large** 2.2 MLOC
kind of data	change history	C source code	release database	change history	change history	C source code
granularity	**coarse**: subsystem, module	fine: function, function call	**coarse**: system, subsystem, module	**coarse**: module, feature, modif. req.	**coarse**: feature, modif. req., file change	fine: LOC
availability	**proprietary**	public domain	**proprietary**	**proprietary**	**proprietary**	open source
time scale	10 years	long	21 months	several years	12 years	6 years
# releases	?	30	8 major, 12 minor	> 1 / year	12 US, 15 international	34 stable, 62 development
# cases	1	1	1	1	1	1
purpose	predictive	predictive	retrospective	predictive	retrospective	retrospective
analysis	statistical	human interpretation	human interpretation	statistical	statistical, visual	visual, human interpretation
representative	?	probably for C code (GNU)	?	?	for highly-reliable embedded real-time systems	?

Short-term evolution studies

	Mattsson&al99	Demeyer&al99	Demeyer&al00	Antoniol&al99	Basili&al96
software size	**medium** 70<NOC<600	**medium** 500<NOC<800	**medium** 100<NOC<1000	**medium** 120<NOC<135	**medium** NOC=180 5 < KLOC < 14
kind of data	C++ source code	Smalltalk source code	Smalltalk source code	C++ source code	C++ source code
granularity	medium: classes	**fine**: classes, methods, …	**fine**: classes, methods, …	**fine**: classes, methods, LOC	**fine**: classes, methods, …
availability	1 proprietary & 2 commercial	commercial	1 commercial, 2 public domain	1 public domain	academic
time scale	short	< 4 years	short	unknown	very short
# releases	between 2 and 5	3	between 2 and 4	7 major, 24 minor	1
# cases	3	1	3	1	8
purpose	predictive	retrospective	retrospective	predictive	predictive
analysis	human interpretation	human interpretation	human interpretation	statistical	statistical
representative	?	for OO frameworks	for refactored OO frameworks	probably for C++ code (GNU)	No. Too small, too academic

Comparison of approaches

	Short-term evolution	Long-term evolution
software size	medium	(very) large
kind of data	OO source code	change management database
availability	commercial, public domain	proprietary
granularity of metrics	fine or medium	coarse
time scale	< 5 years	> 2 years
# releases	< 10	> 10
# case studies	between 1 and 3	1
analysis of results	human, visual	statistical, visual
representativeness	limited	often

Need more work on...

- Evolution metrics
  - must be precise and unambiguous
  - must be empirically validated (e.g., what constitute good coupling and cohesion metrics)
  - are preferrably *language independent*
  - at which level of granularity?
- Scalability
  - metrics require enormous amount of data about software
  - becomes even worse when studying a release history
  - *visualisation* and *statistical* techniques may help

Need more work on...

- Empirical validation
  - validate on sufficiently large set of realistic cases
  - take care with human interpretation
  - ensure replicability
  - common *benchmark* of cases to compare experimental results
- Compare evolutive nature of software based on
  - Development process (open source vs traditional)
  - Application domain (telecom, e-commerce, ...)
  - Problem domain (GUI, embedded, distributed, real-time, ...)
  - Solution domain (framework, program, library, ...)
- Process issues
  - How can we predict/estimate productivity, cost, effort, time, ...

Need more work on...

- Measuring software quality
  - How can we detect decreases/increases in quality?
  - How can we express quality in terms of software metrics?
- Understanding evolution
  - Can we detect the kind of evolution?
  - Can we reconstruct the *motivation* behind a certain evolution?
- Data gathering
  - Often, limited amount of data is available from previous releases
  - Use change-based instead of state-based CM tools
  - Document as much decisions/assumptions/... as possible