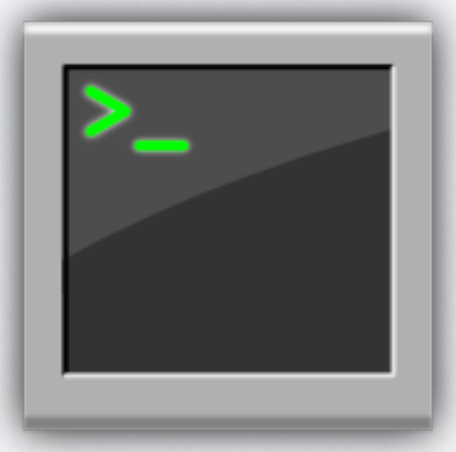
SOFTWARE MAINTENANCE

Dr. Vadim Zaytsev
Universiteit van Amsterdam
27 January 2014



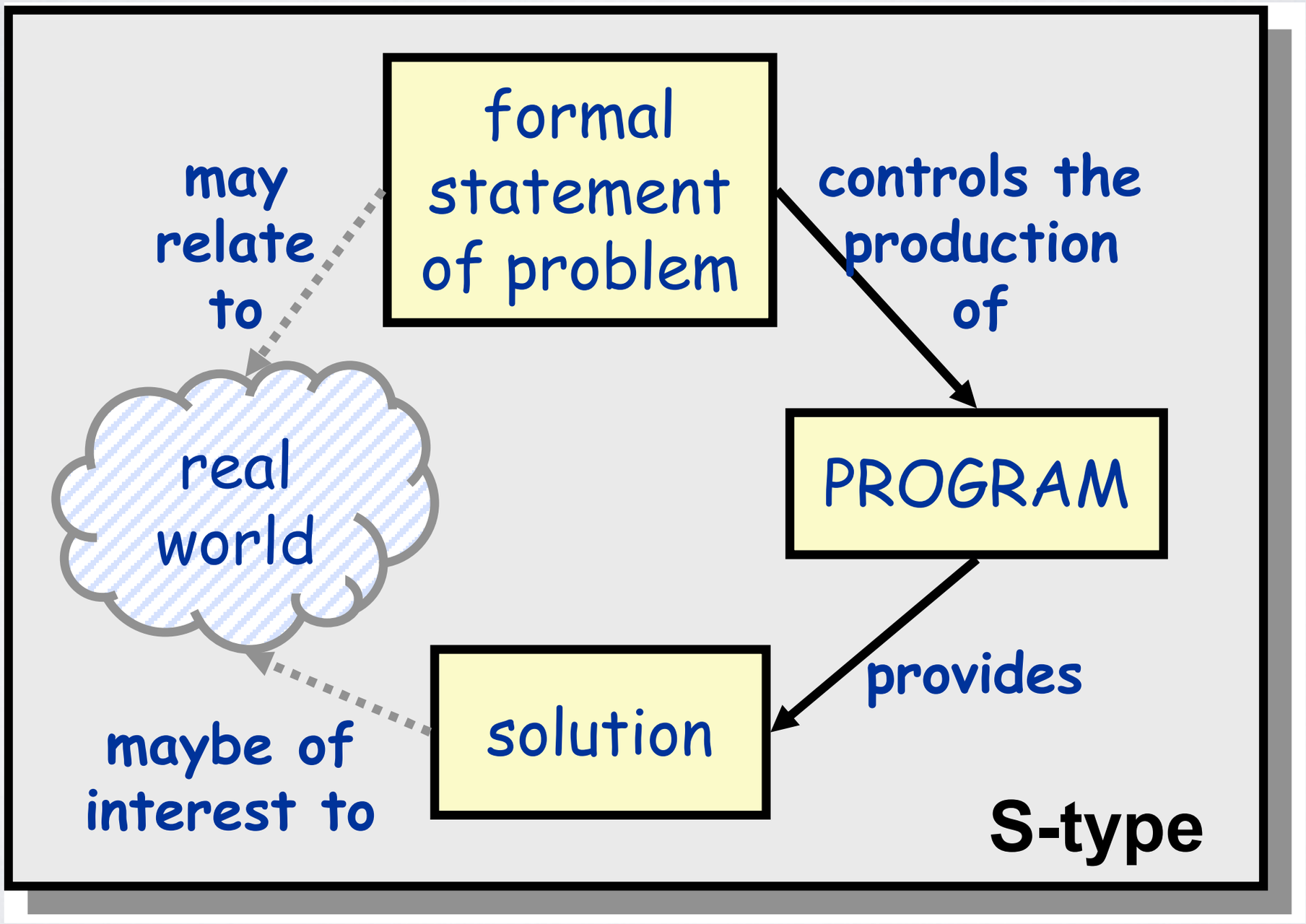
SOFTWARE TYPES

Program Types: S

- S-type programs
  - "specifiable"
  - problem can be formally defined by a spec
  - automated acceptance possible
  - such software **does not evolve**

M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Program Types: S



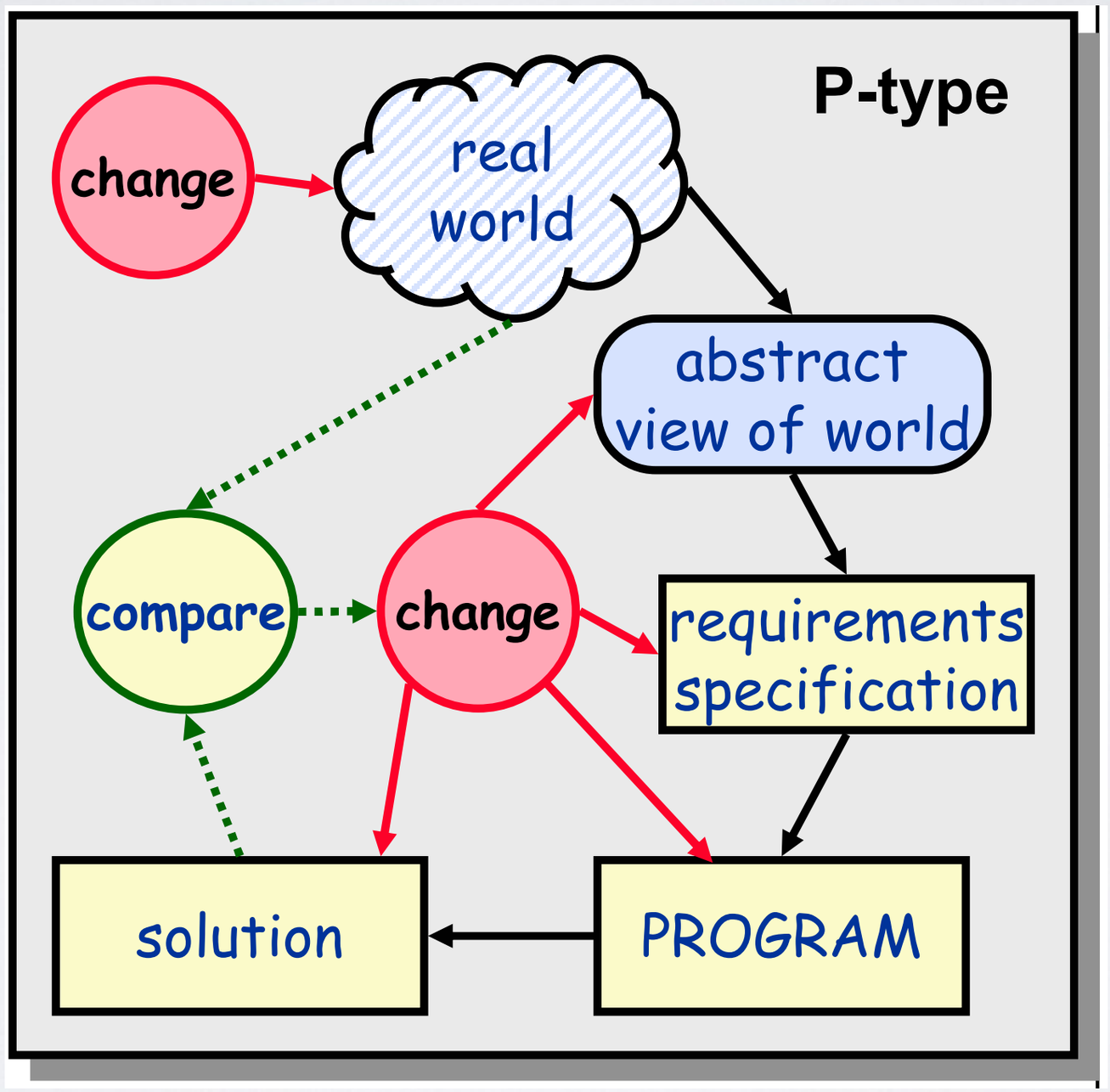
Steve Easterbrook, http://www.cs.toronto.edu/~sme/CSC444F/slides/L20-SoftwareMaintenance.pdf

Program Types: P

- P-type programs
  - "problem-solving"
  - problem imperfectly models a real-world task
  - qualitative acceptance
  - such software probably evolves continuously

M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Program Types: P



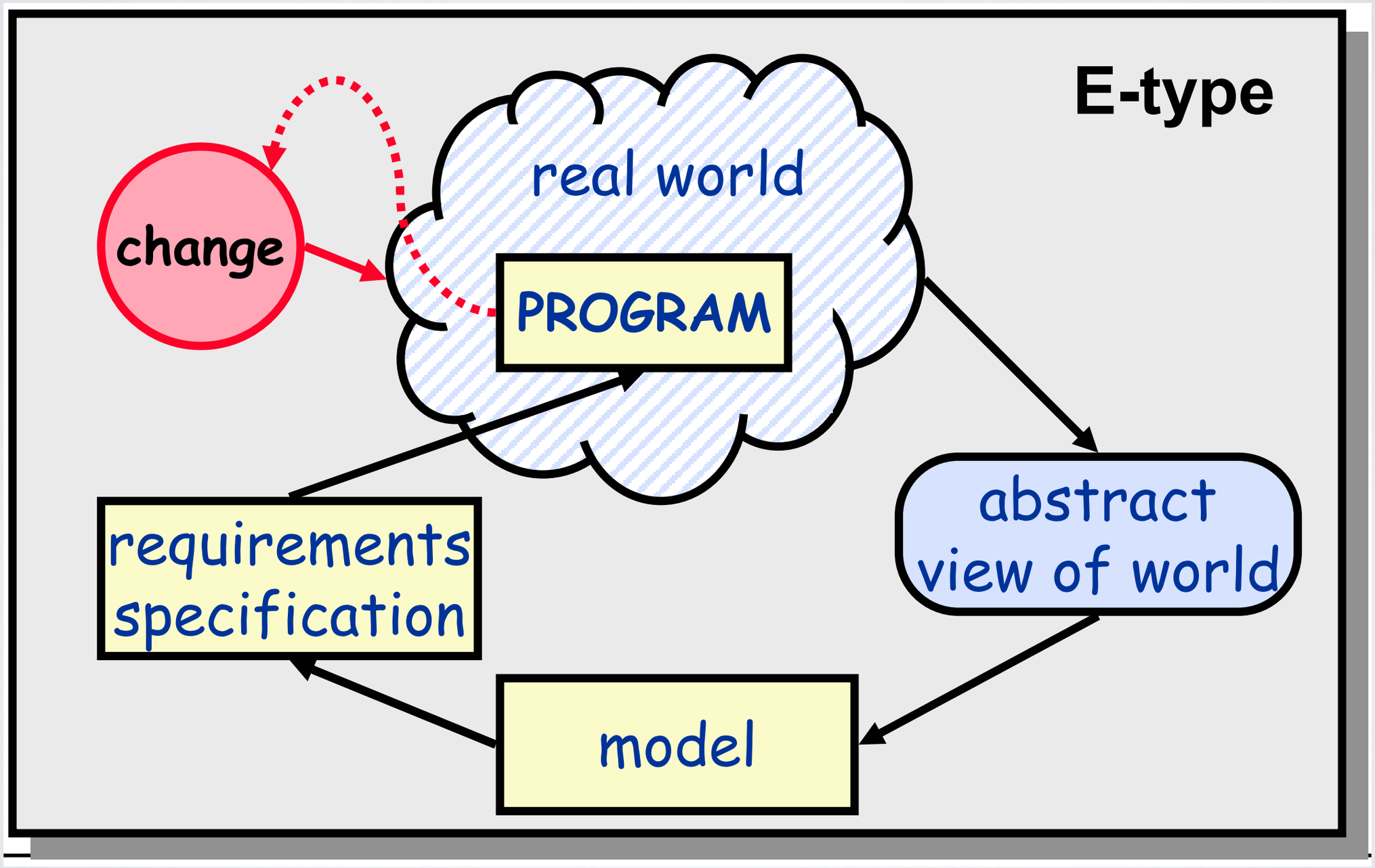
Steve Easterbrook, http://www.cs.toronto.edu/~sme/CSC444F/slides/L20-SoftwareMaintenance.pdf

Program Types: E

- E-type programs
  - "embedded"
- solution becomes a part of the world
- acceptance is subjective
- such software is inherently evolutionary

M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Program Types: E



Steve Easterbrook, http://www.cs.toronto.edu/~sme/CSC444F/slides/L20-SoftwareMaintenance.pdf

LEHMAN'S LAWS OF SOFTWARE EVOLUTION

Lehman's Laws (1/8)

- Continuing Change
  - E-system rots unless adapted
  - the process never stops
  - (true for P-systems as well)

M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Lehman's Laws (2/8)

- Increasing Complexity
  - E-system becomes more complex
  - evolving means complicating
  - (unless we do something)

M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Lehman's Laws (3/8)

- Self-regulation
  - E-system evolution is SRP
  - obeys certain statistical laws
  - (distribution close to normal)

M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Lehman's Laws (4/8)

- Conservation of Organisational Stability
- E-system dev activity is invariant
- throughout its lifetime
- (does not depend on resources)

M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Lehman's Laws (5/8)

- Conservation of Familiarity
  - E-system changes per release: invariant
  - throughout its lifetime
  - (too little: bored; too much: overwhelmed)

M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Lehman's Laws (6/8)

- Continuing Growth
  - E-system must add features over time
  - to keep users satisfied
  - (expectations creep)

M.M.Lehman, J.F.Ramil, P.D.Wernick, D.E.Perry, W.M.Turski,
Metrics and Laws of Software Evolution — The Nineties View, METRICS, 1997.

Lehman's Laws (7/8)

- Declining Quality
  - E-system perceived quality declines
  - internal as well as external
  - (unless constantly maintained)

M.M.Lehman, J.F.Ramil, P.D.Wernick, D.E.Perry, W.M.Turski,
Metrics and Laws of Software Evolution — The Nineties View, METRICS, 1997.

Lehman's Laws (8/8)

- Feedback System
  - E-system evolution is a feedback system
    - multi-level
    - multi-loop
    - multi-agent

M.M.Lehman, J.F.Ramil, P.D.Wernick, D.E.Perry, W.M.Turski,
Metrics and Laws of Software Evolution — The Nineties View, METRICS, 1997.

Simonyi's Law Wish

- Costs proportional to
  - SE: entities * aspects (impl)
  - DSL: entities + aspects (impl)
  - Goal: entities + aspects (domain)

Charles Simonyi, The Magic of Software, MoDELS 2013 keynote.

MAINTENANCE

Software engineering

Year	New projects	Enhancements	Repairs	Total
1950	90	3	7	100
1960	8,500	500	1,000	10,000
1970	65,000	15,000	20,000	100,000
1980	1,200,000	600,000	200,000	2,000,000
1990	3,000,000	3,000,000	1,000,000	7,000,000
2000	4,000,000	4,500,000	1,500,000	10,000,000
2010	5,000,000	7,000,000	2,000,000	14,000,000
2020	7,000,000	11,000,000	3,000,000	21,000,000

Jurgen Vinju, http://homepages.cwi.nl/~jurgenv/teaching/evolution1314/slides/intro-evol.pdf

Definition of maintenance

- Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment

IEEE 1219, 1993

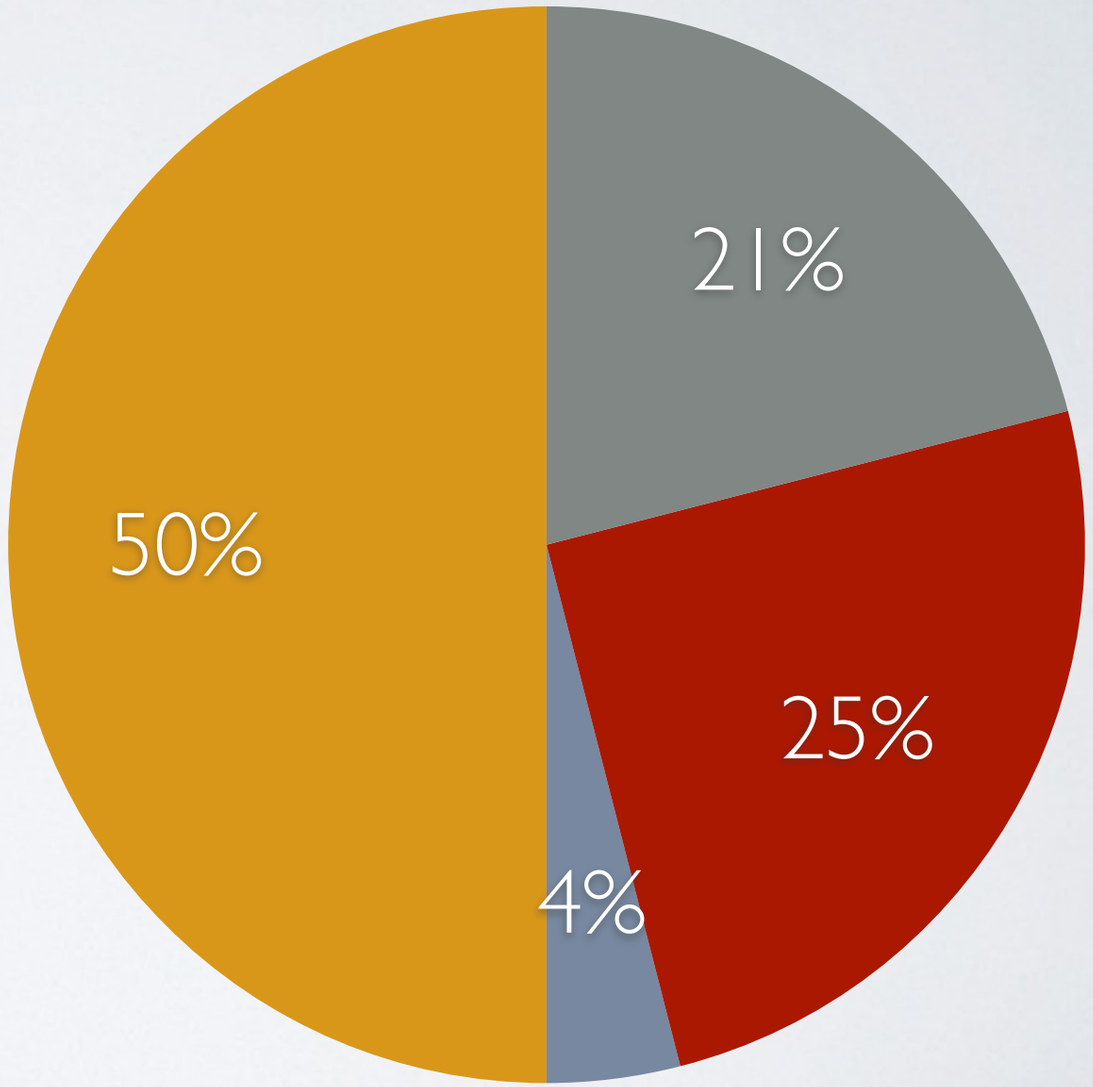
Maintenance phases

- Introductory
  - user support!
- Growth
  - correcting faults!
- Maturity
  - enhancements!
- Decline
  - technology replacement!

Hans van Vliet, *Software Engineering: Principles and Practice*. Jon Wiley & Sons, 2009.

Types of maintenance

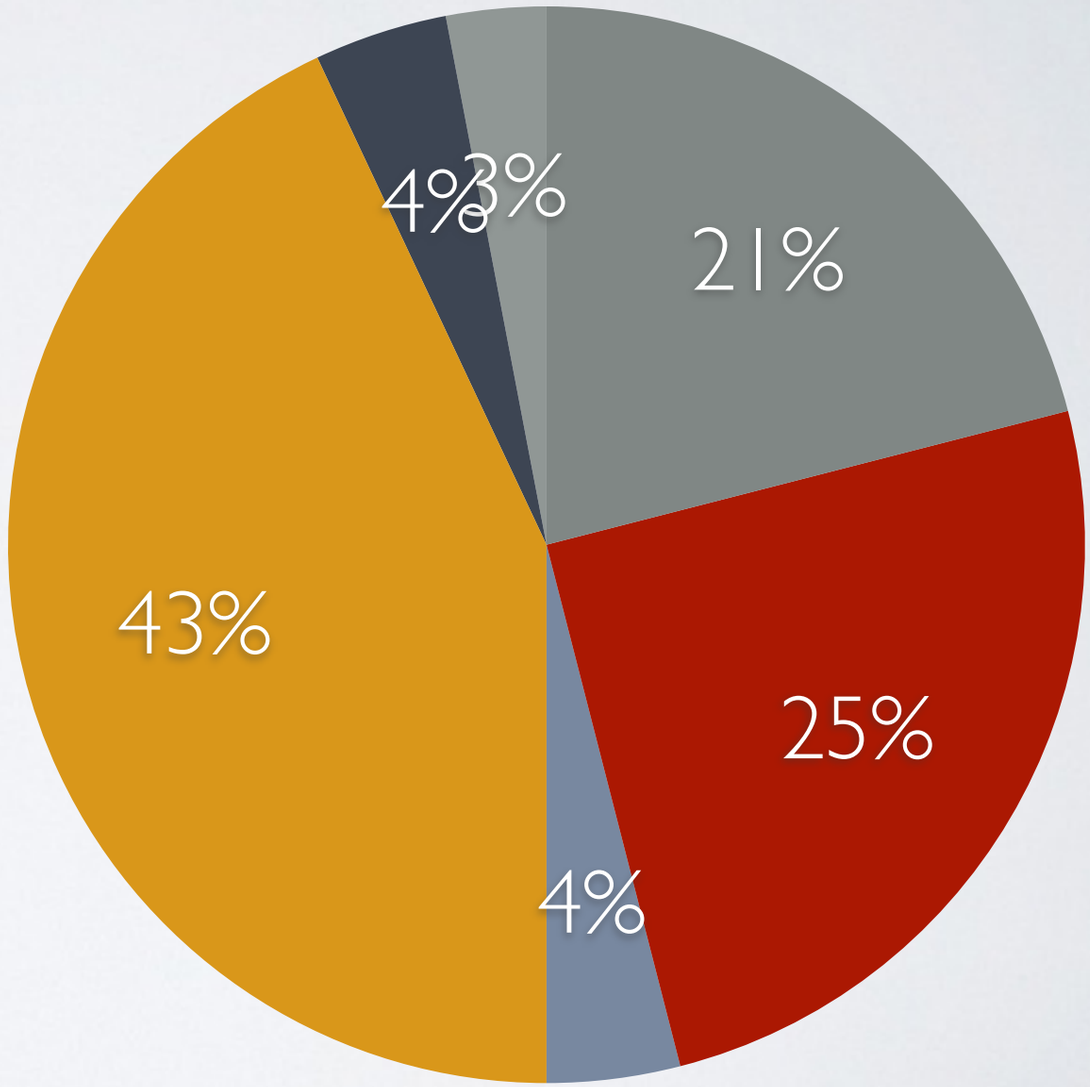
- Corrective
- Adaptive
- Perfective
- Preventive



B.P.Lientz, E.B.Swanson, *Software Maintenance Management, A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*, 1980.

Types of maintenance

- Corrective
- Adaptive
- Perfective
  - user enhancements
  - efficiency
  - other
- Preventive



B.P.Lientz, E.B.Swanson, *Software Maintenance Management, A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*, 1980.

Top 5 problems

- Quality of documentation
- User demand for enhancements
- Competing demands for maintainers' time
- Meeting scheduled commitments
- Turnover in user organisations

S.L.Pfleeger, *Software Engineering: Theory and Practice*, Prentice Hall, 1998.

Is it hopeless?

- Higher quality
  - less (c) maintenance
- Anticipating changes
  - less (a&p) maintenance
- Better tuning to user needs
  - less (p) maintenance
- Less code
  - less (*) maintenance

Maurice ter Beek, http://www.liacs.nl/~mtbeek/se-ma.pdf

Legacy systems

- Size and complexity
  - out of control
- Knowledge
  - "in the basement"

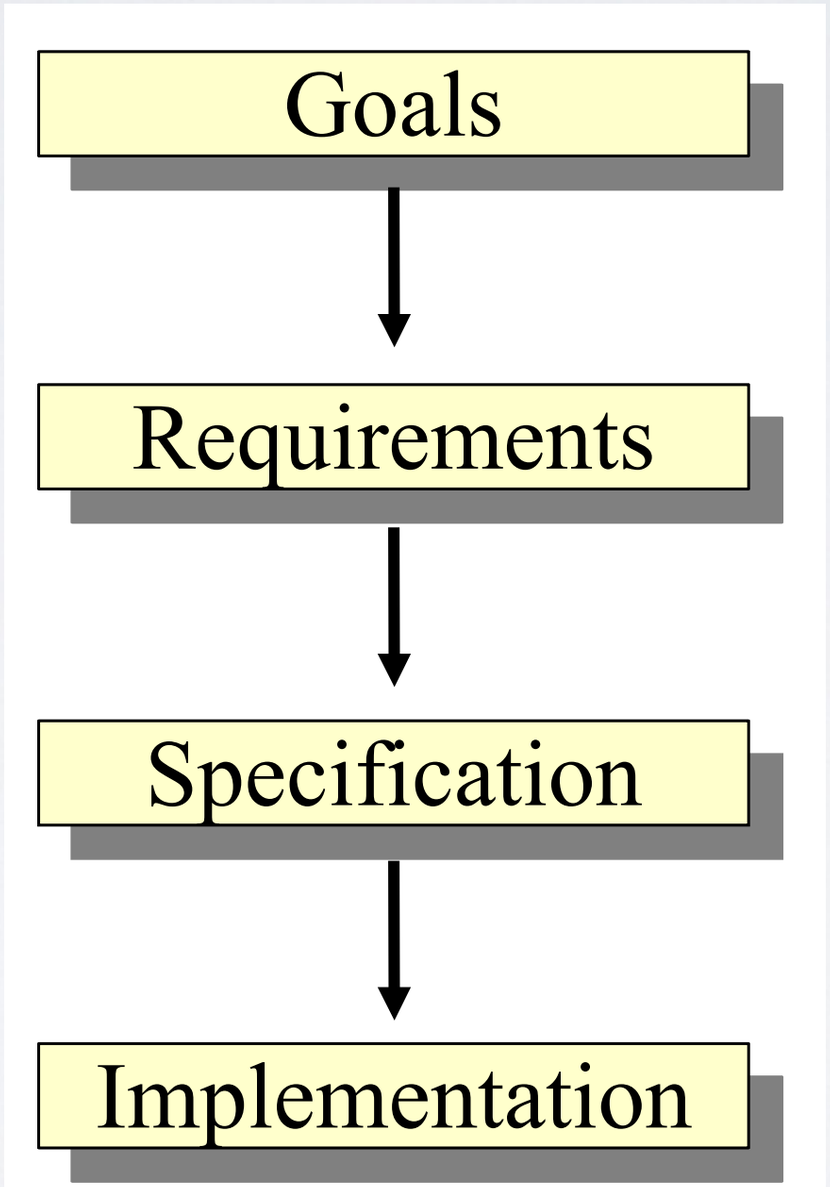
Typical software system

~1-100 MLOC

- Implemented with a language cocktail
- Glue languages: JCL, Make, ...
- Some parts never run/accessed
- Some source code lost/incomplete
- Documentation is incomplete or obsolete

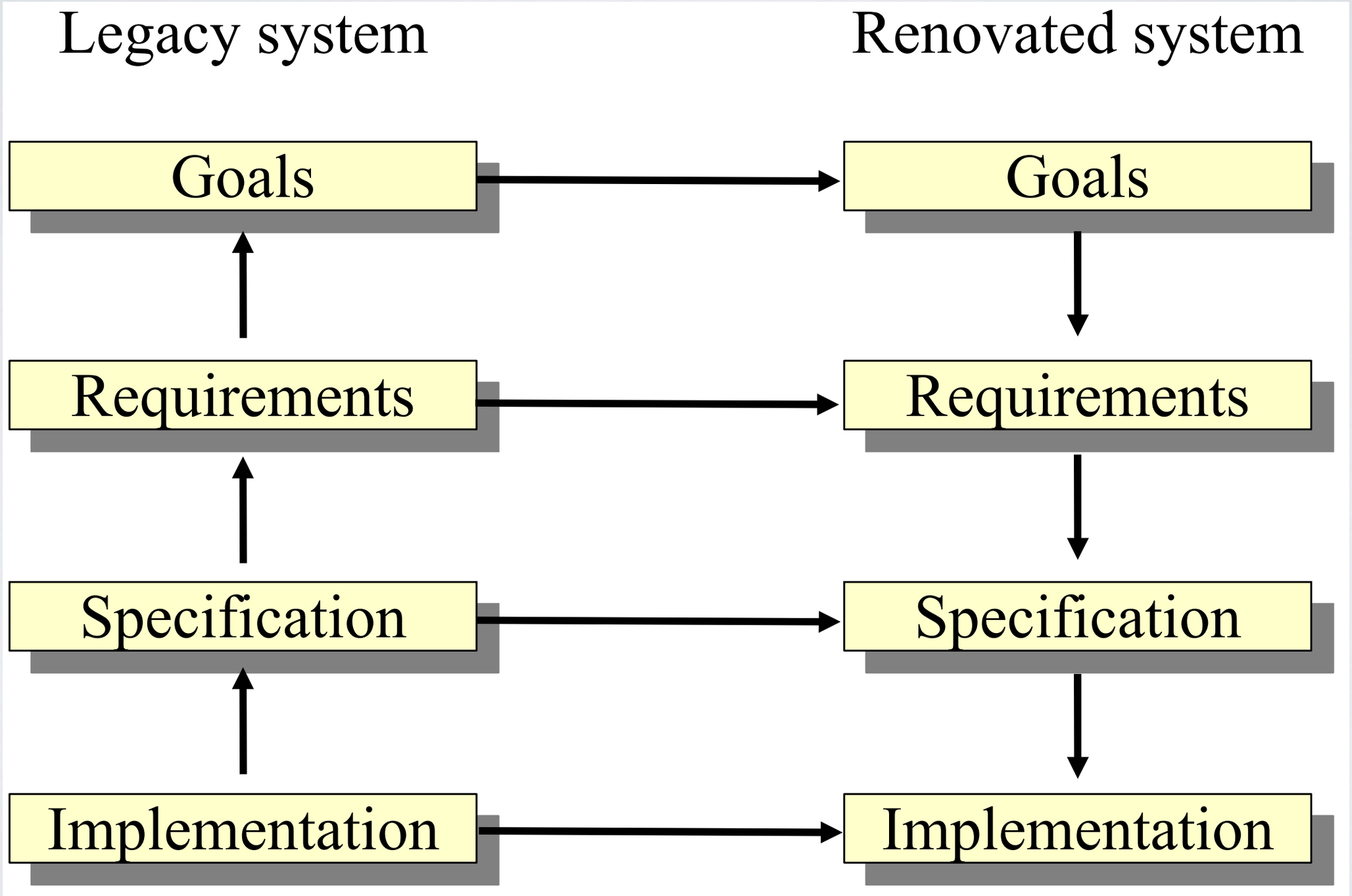
Jurgen Vinju, http://homepages.cwi.nl/~jurgenv/teaching/evolution1314/slides/intro-evol.pdf

Forward Engineering



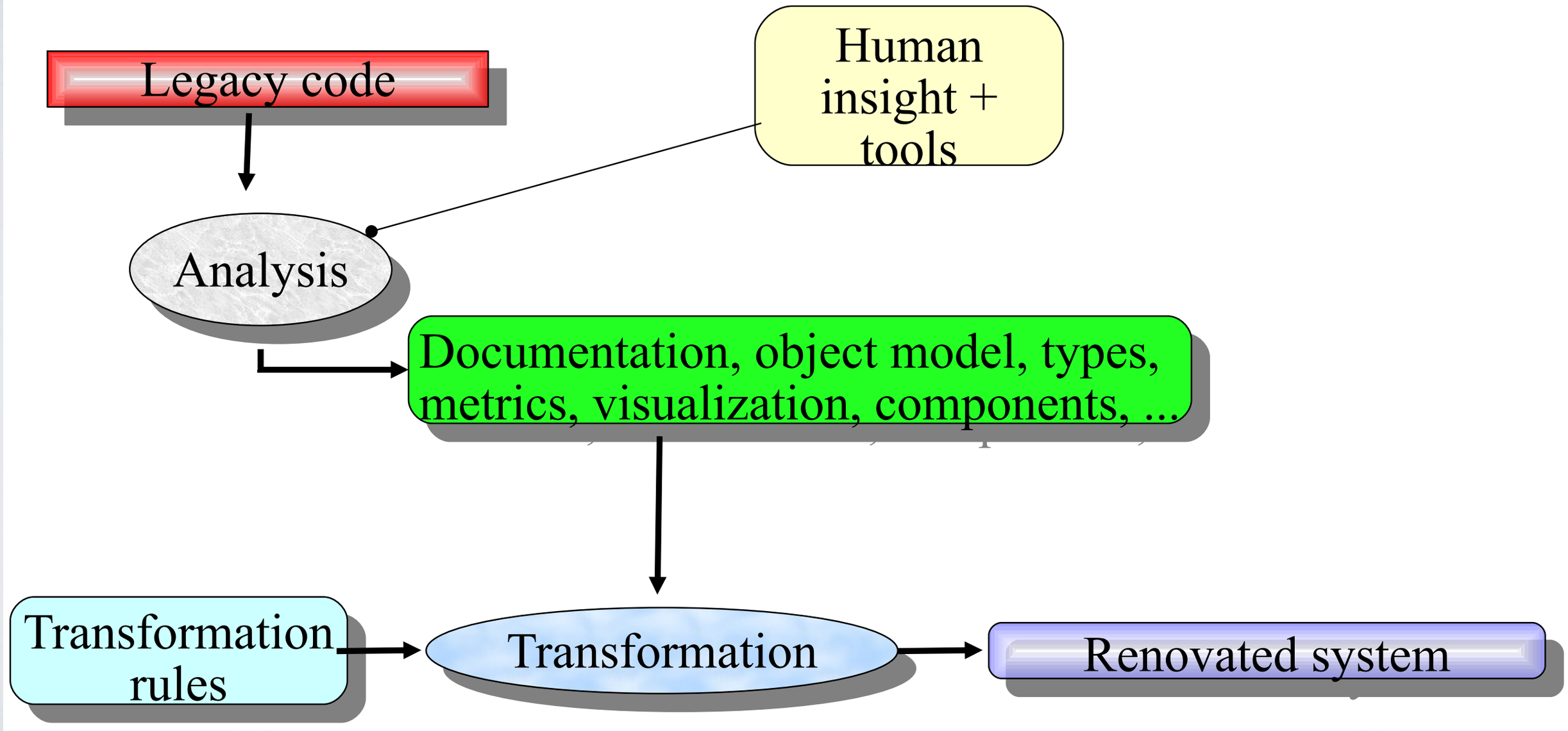
Jurgen Vinju, http://homepages.cwi.nl/~jurgenv/teaching/evolution1314/slides/intro-evol.pdf

Reverse Engineering



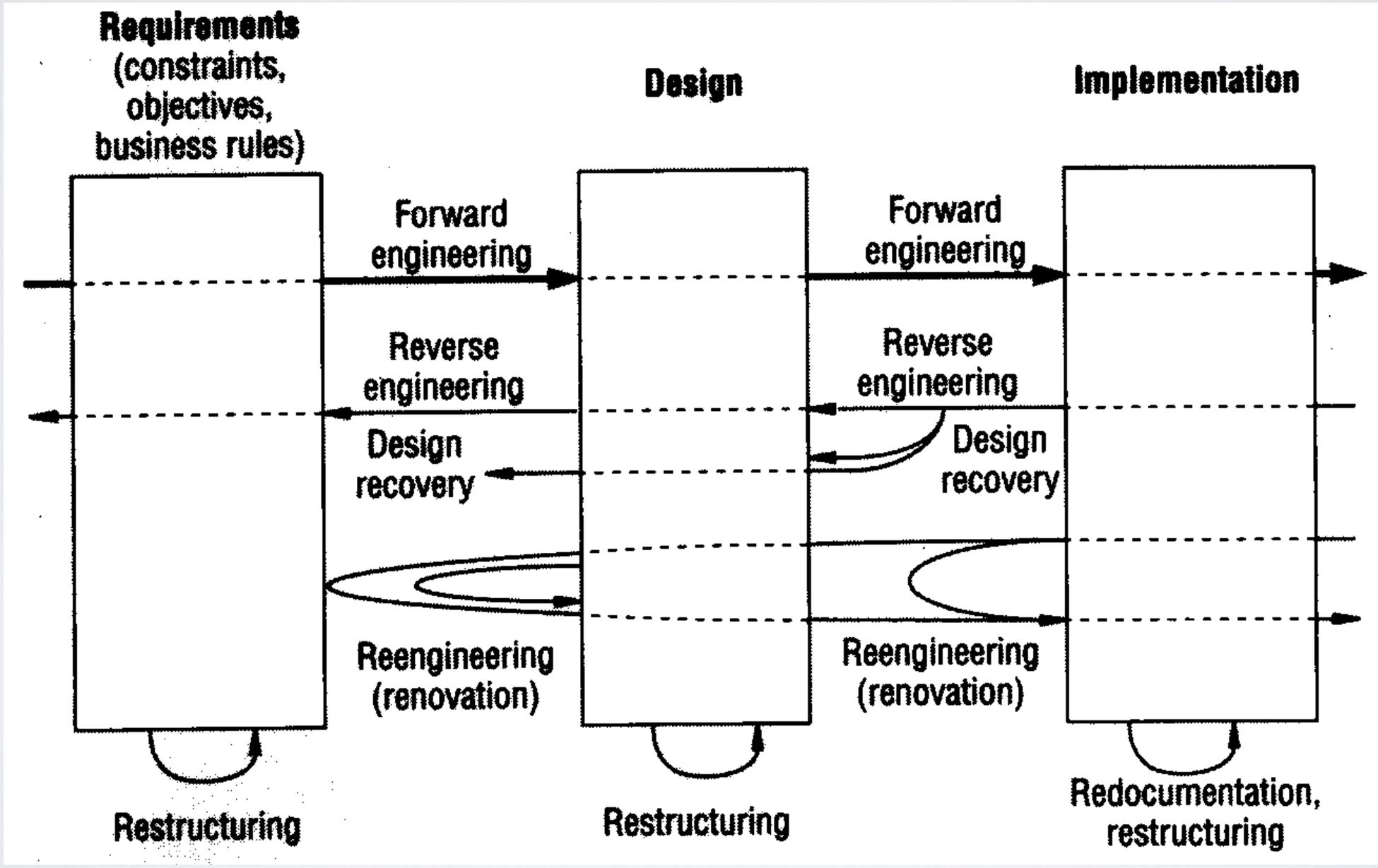
Jurgen Vinju, http://homepages.cwi.nl/~jurgenv/teaching/evolution1314/slides/intro-evol.pdf

Software renovation



Jurgen Vinju, http://homepages.cwi.nl/~jurgenv/teaching/evolution1314/slides/intro-evol.pdf

Reverse Engineering and Design Recovery



Elliot J. Chikofsky, James H. Cross II: *Reverse Engineering and Design Recovery: A Taxonomy.* IEEE Software 7(1), 1990.

Refactoring

- Rewriting code without changing behaviour
- Built-in in many IDEs
- Remove code smells
  - long method, ugly name, many arguments,
  - middle man, feature envy, shotgun surgery

M.Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.

CONCLUSION

Summary

- Software evolves
- Software evolution obeys certain laws
- Software rots in time
- 70% of software engineers do maintenance
- Many software systems are legacy
- Forward, reverse and re-engineering

- Sources of information/inspiration
  - given on the bottom of each slide
- Slides?
  - http://grammarware.net/slides/2014/maintenance.pdf
- Fonts?
  - Avdira — George Douros, Unicode Fonts for Ancient Scripts, 2009.
  - Finger Paint — Ralph Oliver du Carrois, 2013.
  - Wild Honey — Denis Sherbak, 2013
- Questions? Ask or email or tweet.