

Software Engineering State-of-the-art

An introduction to the not-so-innocent

Amnon H Eden
 Department of Computer Science, University of Essex
 Center For Inquiry, Amherst, NY
 9 Feb. 2005

Table of Contents

- Historical origins
 - Problems and strategies for solutions
- Software design
 - Strategic & tactical design
- The software lifecycle
 - Problems & solutions
- Future directions

2

Historical origins

The never-ending story of the “software crisis”

1968: “Software crisis”

- Definition: “*The application of a systematic, disciplined, quantifiable approach to the development ...and maintenance of software.*” [IEEE]
- NATO conference [Naur & Randell 69]
 - “The computer industry has a great deal of trouble in producing large and complex software systems.”
 - **Problems:** Late deployment, budget overflows, unreliable and unsatisfactory systems, systems never delivered...

4

1994: “Software’s chronic crisis”

- “Software engineering... a term of aspiration” [Gibbs 94]
- **Problems:** Late deployment, budget overflows, unreliable and unsatisfactory systems, systems never delivered...

The fate of US defense projects according to US government statistics. (Cammel & Shafer 1989)

5

2005: “Software hell”

- “...software debacles are routine. And the more ambitious the project, the higher the odds of disappointment.” [Carr 05]
- Only 34% of projects: timely & within budget.
- FBI
 - Since 2001: a database on suspected terrorists
 - Jan. 2005: \$M170, “not even close to having a working system”
- Ford Motors
 - Since 2000, project “Everest”: buying supplies, replacing legacy
 - Aug. 2004: \$M200 over budget, abandoned (β-version slower than legacy)
- McDonald’s
 - Since 1999, project “Innovate”, budget \$M1,000
 - Killed in 2002, writing off \$M170

6

Strategies

1. Mathematics
 - Objectives: Well-defined descriptions (specification, design), Analysis of properties, planning
 - Applications: Program generation, automated verification, software design theory, Metrics
2. Abstraction
 - Objectives: Improve understanding & planning
 - Applications: Conceptualization of abstract design components, e.g., design patterns, architectural styles
3. Standards
 - Objectives: Common language, reuse
 - Applications: Components, patterns & styles

Software engineering state-of-the-art 7

Software design

Strategic & tactical design

What is software design?

- Distinguish:
 1. Activity ("phase") in software lifecycle ("design before implementation")
 2. A current, abstract model of the implementation ("The design of this program")
- Trend: *design as activity*
 - 2nd interpretation: *Model, software modelling*

Software engineering state-of-the-art 9

Design as abstraction

- Objective: Develop manageable abstraction over programs
 - Motivation: Size and complexity
- Levels of abstraction:
 - Tactical design
 - Data structures, algorithms, design patterns, class diagrams
 - ...
 - Strategic design
 - Design principles and paradigms, architectural styles, component-based software engineering (CBSE) standards

Software engineering state-of-the-art 10

Algorithms & data structures

- The traditional focus in software design
- Origins: 1960s
 - Abstract & concrete data types
 - Sorting, searching, inserting/removing
- Focus: time/space efficiency
 - Other values: Elegant code

Software engineering state-of-the-art 11

Example: Stack

- Guiding principle: LIFO (Last In First Out)

```

public class Stack {
    public void push(Object x); // Add x to top
    public Object pop(); // Remove and return top
    public Object top(); // Return top
    public boolean Empty(); // Return true iff empty
    public int Size(); // Return size
}
    
```

Software engineering state-of-the-art 12

Stack II

- Concrete data types (implementations):
 - Linked list
 - Array
- Properties:
 - Operations are “cheap” – low complexity: $O(1)$
 - Standard, recognized, reusable

Software engineering state-of-the-art 13

Design patterns

- Origins: late 1987
- Means for capturing common solutions to common problems in software design
 - Organizing classes hierarchies & object collaborations
- Focus: Loose coupling
 - Other values: reuse, elegant structure

Software engineering state-of-the-art 14

Example: *Recursive Composite*

Source: [Gamma et. al 1995]

- **Intent:** “Compose objects into tree structures to represent part-whole hierarchies.
 - “treat individual objects and compositions of objects uniformly.”
- **Example:** Files and directories

Software engineering state-of-the-art 15

Recursive Composite II

- Structure:
- Participants:
 - Component
 - Leaf
 - Composite
 - Client
- Collaborations:
 - ...

Software engineering state-of-the-art 16

Software architecture

- Origin:
 - 1976 DeRemer & Kron: “Programming-in-the-large vs. Programming-in-the-small”
- The highest level of abstraction
 - “Beyond data structures and algorithms”
- Architectural styles: Abstraction of “families” of related systems
 - [Perry & Wolf 92]
 - [Garlan & Shaw 93]

Software engineering state-of-the-art 17

Example: *Layered Architecture*

- AKA: **Abstract Machine Model**
- Principles:
 - Each “entity” belongs to a *layer*

$$\forall e \exists !k \in \{N\} \bullet Layer(e) = k$$
 - Every entity may only depend on modules in same or lower layers

$$\forall x, y \bullet Depends(x, y) \Rightarrow Layer(x) \geq Layer(y)$$

Software engineering state-of-the-art 18

Layered Architecture II

Examples:

ISO Communication Protocols

UNIX Operating System

Software engineering state-of-the-art 19

Example: Client-Server

- A distributed system with --
 - **Server(s)**: Stand-alone, powerful machine(s)
 - **Clients**: Subsystem using the services
 - **Network**: Communicating clients with serves

Software engineering state-of-the-art 20

Design principles

- A design decision that affects all aspects of the implementation
 - **Modularization**: Breaking large systems to manageable modules
 - **Information hiding**: Distinguish “what” from “how” (interface vs. implementation)
 - **Design by contract**: Pre-/postconditions, assertions, invariants

Software engineering state-of-the-art 21

Design paradigms

- A.k.a. *programming paradigms*
- Combine several design principles
- Committed to a particular ontology
 - Object-oriented design
 - Abstractions: *Objects, messages*,
 - Programming languages: Java, C++, Smalltalk, Eiffel
 - Modular design
 - Abstractions: *data structure, module*
 - Programming languages: Modula, Pascal
 - Functional design
 - Abstractions: *Function, algorithm, recursion*
 - Programming languages: Lisp, Miranda, Haskell

Software engineering state-of-the-art 22

Example: Object-oriented design

- Modularization: Classes, members
- Computation: Objects, messages

Software engineering state-c 23

Component-Based Software Eng.

- Software reuse: The “holy grail”
 - CBSE: Most successful
- **Motivation**: A commercial market for standard components
 - Arithmetic operations, database management, grammar checking, mobile phone programming, graphic imaging, ...
- **Benefits**: competition, reduce risk by division of labour
- **Principal abstractions**: *component, interface*

↔

Software engineering state-of-the-art 24

CBSE standards

- Objective: Compatibility!
- Emerging standards:
 - Java Beans™ (Sun)
 - Enterprise JavaBeans™ (Sun)
 - CORBA (OMG)
 - COM/DCOM, .NET (Microsoft)
 - Visual Basic (Microsoft)
- Conventions:
 - Calling conventions ("language-independent")
 - Inter-process communications
 - Object communications
 - Foundation services

Software engineering state-of-the-art 25


The Software Lifecycle

Problems and approaches

Project management

"No scene from prehistory is quite so vivid as that of the mortal struggles of great beasts in the tar pits... In the mind's eye one sees dinosaurs, mammoths, and sabertoothed tigers struggling against the grip of the tar. The fiercer the struggle, the more entangling the tar, and no beast is so strong or so skillful but that he ultimately sinks.

Large-system programming has over the past decade been such a tar pit..." [Brooks 75]



Software engineering 27

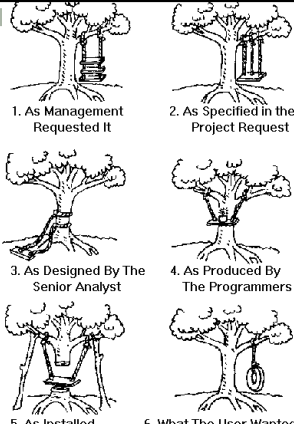
Project management II

- Brook's law: "Adding manpower to a late software project makes it later." [Brooks 75]
- Hofstadter's Law: "It always takes longer than you expect, even when you take Hofstadter's Law into account. [Hofstadter, "Gödel, Escher, Bach"]"

Software engineering state-of-the-art 28

Requirements engineering

- Principal difficulty: Well-defined, correct specifications
- Strategies: Mathematics, abstraction, standardization



Software engineering state-of-the-art 29

Software maintenance & evolution

- "Maintenance":
 - Bug correction
 - adjustment to new requirements
 - changes in operating environment
- **Problem:** Between 50%-90% of the cost of software goes to maintenance
- **Reasons?**
 - "Maintenance": Misconception.
 - Software *evolves* over time

Software engineering state-of-the-art 30

Maintenance or evolution?

Waterfall Process Model

- Leading assumption:
 - Maintenance (changes) is a phase*

Software engineering state-of-the-art 31

Maintenance or evolution?

“Evolutionary” model: Reflects reality better!

Software engineering state-of-the-art 32

Laws of software evolution

[Lehman 80, Lehman 96]

- I - Continuing Change
 - An E-type program that is used must be continually adapted else it becomes progressively less satisfactory.
- II - Increasing Complexity
 - As a program is evolved its complexity increases unless work is done to maintain or reduce it.
- VII - Declining Quality
 - E-type programs will be perceived as of declining quality unless rigorously maintained and adapted to a changing operational environment.

Software engineering state-of-the-art 33

Software modelling

- Objectives:
 - Well-defined map to large systems
 - Blueprints for systems under construction
- Examples
 - Flow charts: Algorithms
 - Data-flow diagrams: Data & control flow
 - 1995: “Unified Modelling language” – UML

Software engineering state-of-the-art 34

Example: UML

De-facto industry standard for software representation:

- Class & package diagrams
- Collaboration/Sequence diagrams
 - Origins: Booch, OMT
- State diagrams
 - Origins: Harel's statecharts
- Activity diagrams
- Component diagrams
- ...

Software engineering state-of-the-art

Unified? Modelling? Language?

- Pros:
 - Standard notation for planning/describing an O-O program
- Cons:
 - Language? No well-defined semantics
 - No automated verification
 - Modelling? Low level of abstraction
 - Inadequate means for abstraction & navigation
 - Unified? Little integration between sub-notations
 - No means for proving/refuting consistency

Software engineering state-of-the-art 36

UML: A tower of Babel



Pieter Bruegel, *Tower of Babel*, 1593. Boijmans Museum, Rotterdam

Software engineering state-of-the-art 37

“UML – The Positive Spin”

“UML is in fact as complex as a big and cryptic programming language, with generous use of “\$” and “#” and “-” and “” and “solid triangles with no tail” and rectangles and diamonds and solid lines and dotted lines and solid ellipses and dotted ellipses and arrows of all kinds and keywords such as “const” and “sorted” (not to be confused with “ordered”) and different semantics for a class depending on whether its name appears in roman or italics; ...*

...but at least a programming language, even the worst of languages, is executable! Here you have to learn all this monstrous complexity just to build diagrams of a possible future system.” [Meyer 97]

Software engineering state-of-the-art 38

Future directions

Towards a science of software

Formal methods

- *“It has long been my personal view that the separation of practical and theoretical work is artificial and injurious. Much of the practical work done in computing, both in software and in hardware design, is unsound and clumsy because the people who do it have not any clear understanding of the fundamental design principles of their work. Most of the abstract mathematical and theoretical work is sterile because it has no point of contact with real computing.”*

-- Christopher Strachey (1916-1975), Leader, Programming research group, Oxford U. computing laboratory

Software engineering state-of-the-art 40

Formal methods

- Mathematical techniques for describing software properties
- Applications:
 - Functional requirements specification
 - Non-functional requirements specification
- Languages and formalisms:
 - Communicating Sequential processes (CSP), Z, Object-Z, Abstract State Machines (ASM), VDM,

Software engineering state-of-the-art 41

Example: *Recursive Composite* in Z

Recursive Composite

Composite, Component : CLASS

Leaves : \mathbb{P} CLASS

Operation : SIG

$(Composite, Component) \in Inherit$

$\forall x \in Leaves \bullet$
 $(x, Component) \in Inherit$

$(Operation \otimes Composite,$
 $Operation \otimes Component) \in Invoke$

[CLASS]
[SIG]
[METHOD]

RefToMany, Inherit :
CLASS \leftrightarrow CLASS

Member :
CLASS \cup METHOD \leftrightarrow CLASS

Invoke :
METHOD \leftrightarrow METHOD

Signature :
METHOD \rightarrow SIG

\otimes :
SIG \times CLASS \rightarrow METHOD

Software engineering state-of-the-art 42

Example: Functional spec. in Z

```

StorageTank
  contents: ℕ
  capacity: ℕ
  light: {off, on}
  reading: ℕ
  dangerLevel: ℕ

  contents ≤ capacity
  redlight = on ⇔ reading > dangerLevel
  greenlight = on ⇔ redlight = off
  reading = contents
  capacity = 5000
  dangerLevel = 4500
    
```

Software engineering state-of-the-art 43

Example: *Client-Server* in Acme

```

System simple_cs = {
  Component client = { Port sendRequest }
  Component server = { Port receiveRequest }
  Connector rpc = { Roles {caller, callee} }
  Attachments : {
    client.sendRequest to rpc.caller ;
    server.receiveRequest to rpc.callee }
}
    
```

Software engineering 44

Example: *Layered architecture* in the predicate calculus

- Each "entity" belongs to a *layer*

$$\forall e \bullet \text{Entity}(e) \Rightarrow \exists !k \in \{\mathbb{N}\} \bullet \text{Layer}(e)=k$$
- Every entity may only depend on modules in same or lower layers

$$\forall x, y \bullet \text{Entity}(x) \wedge \text{Entity}(y) \wedge \text{Depends}(x, y) \Rightarrow \text{Layer}(x) \geq \text{Layer}(y)$$

Software engineering state-of-the-art 45

Towards a Science of Software

- Objective:** Precise, abstract, meaningful descriptions
- Strategies:** Mathematics, abstraction, standardization
- Benefits:** Understanding, analysis
- Examples:**
 - Metrics
 - Laws of software evolution (Lehman)
 - The Intension/Locality hypothesis

Software engineering state-of-the-art 46

Summary: Strategic, tactical, implementation

- A hierarchy of design abstractions

Strategic	Architectural styles CBSE standards Application frameworks Design principles
Tactical	Design patterns Refactorings Programming idioms
Implementation	Class diagrams Program documentation

Software engineering state-of-the-art 47

The Intension/Locality hypothesis

Intensional	Architectural styles CBSE standards Application frameworks Design principles	Non-local
The Intension criterion	Design patterns Refactorings Programming idioms	Local
The Intension criterion	Class diagrams Program documentation	Local
Extensional		

Software engineering state-of-the-art 48

Bibliography

Software crisis:

- P. Naur, B. Randell, (Eds.) *Software Engineering: Report of a conference sponsored by the NATO Science Committee* (7-11 Oct. 1968), Garmisch, Germany. Brussels, Scientific Affairs Division, NATO, 1969.
- W. W. Gibbs. "Software's Chronic Crisis". *Scientific American* (Sep. 1994), p. 86.
- N. G. Carr. "Does Not Compute". *The New York Times*, OP-ED contribution, 22 Jan. 2005.

Project management:

- F.P. Brooks. "The Mythical Man-Month". *Proc. Int'l Conf. Reliable Software*, Los Angeles, CA, 1975.

Software engineering state-of-the-art

49

Bibliography II

Design patterns:

- D. Schmidt, R. E. Johnson, M. Fayad (1996). *Software Patterns*. Guest editorial, *Communications of the ACM*, Special Issue on Patterns and Pattern Languages, Vol. 39, No. 10 (Oct. 1996).
- E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading: Addison-Wesley 1995.

Modelling:

- J. Rumbaugh, I. Jacobson, G. Booch. *The Unified Modeling Language Reference Manual*. Reading: Addison-Wesley, 2004.
- Bertrand Meyer. *UML — The Positive Spin*. *American Programmer*, Vol. 10, No. 3 (Mar. 1997).

Software engineering state-of-the-art

50

Bibliography III

Software architecture:

- D. E. Perry, A. L. Wolf (1992). "Foundation for the Study of Software Architecture". *ACM SIGSOFT Software Engineering Notes*, 17 (4), pp. 40–52.
- M. Shaw, D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

Software evolution

- M. Lehman. "Laws of Software Evolution Revisited". Lecture Notes in Computer Science 1149 (*Proc. 5th European Workshop on Software Process Technology*), pp. 108–124. Berlin: Springer-Verlag, 1996.

The Intension/Locality hypothesis:

- <http://www.eden-study.org/research/ilh.html>

Software engineering state-of-the-art

51