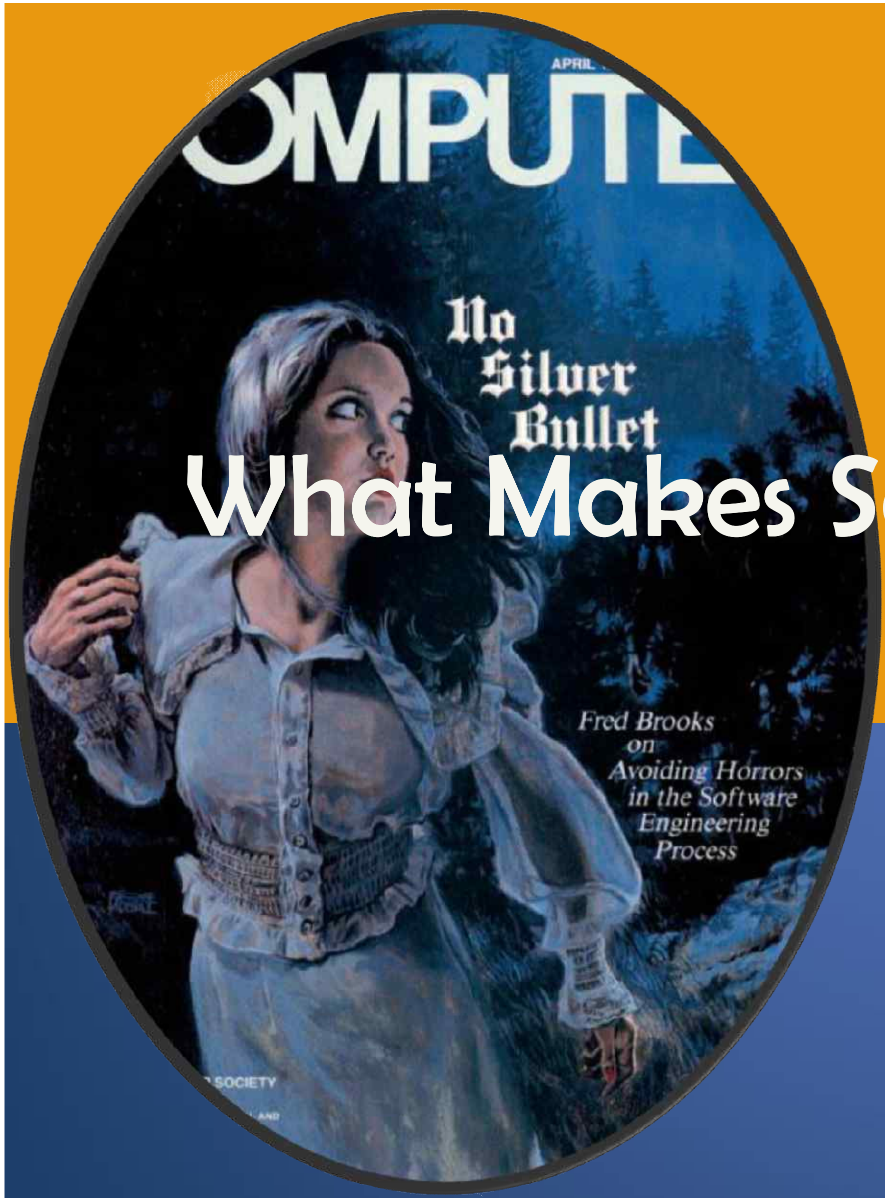
Competencies needed to Software Engineers in the Forthcoming IT Industries

Lee, Joon-Sang
LG Electronics Advanced Research Institute

Contents

- What makes software difficult?
- Future competencies

What Makes Software Difficult ?


APRIL
OMPUTE
No Silver Bullet
Fred Brooks on Avoiding Horrors in the Software Engineering Process
SOCIETY

Current Status of Software Engineers

- Negative phenomena at IT work places
  - No precise estimation of effort and schedule
  - Instant coding from unwritten requirements
  - Low reusability, and no trust on SW from others
  - Repeated & overtime work (6 code lines a year)
  - Frequent changes in requirements
  - Side work for hardware system ?
  - None asks about SW quality seriously (even end-users)
  - Always being dreadful about any little change
  - No time buffer to consider maintainability or robustness
  - Is learning programming language all ?
  - No objective & systematic qualification criteria on experts
- Is software so inherently easy and simple to be not worthy ?
- There's been no matured software engineering yet ?
- SW is the most creative, complex, and difficult work, but too easy to startup and partially demonstrate with no concept of quality!
  - HW engineers < SW developers < SW engineers

Aristotle's Metaphysics

- Theories about what exists and how we know that is exists
  - notice that software is invisible.
- Essential or accidental characteristics ?
  - this horse is white vs. this horse is a kind of brute fact
- How to define SW's beings & their behaviors
  - form & matter via state-based lifecycle model

Philosophical Approach

- What exist in the computational space
  - Ontology
    - process or object ? aggregate of them?
  - Epistemology
    - What do components know about one another ?
    - Scoping rule ? Or connectivity via naming server ?
  - Protocols
    - Dictations of how to interact among components ?
    - Synchronous or asynchronous ? with a type of IPC?
  - Lexicon
    - Vocabulary of component interactions

Essences and Accidents

- Invisibility
  - No geometric abstraction
    - e.g. land (map), silicon chips (diagrams), computers (connectivity schema), building (floor plan)
  - Just superimposed directed graphs on upon another
    - control/data flow, data dependency, time sequence, name-space relationships, module structure, etc.
    - Even planar so inherently hierarchical

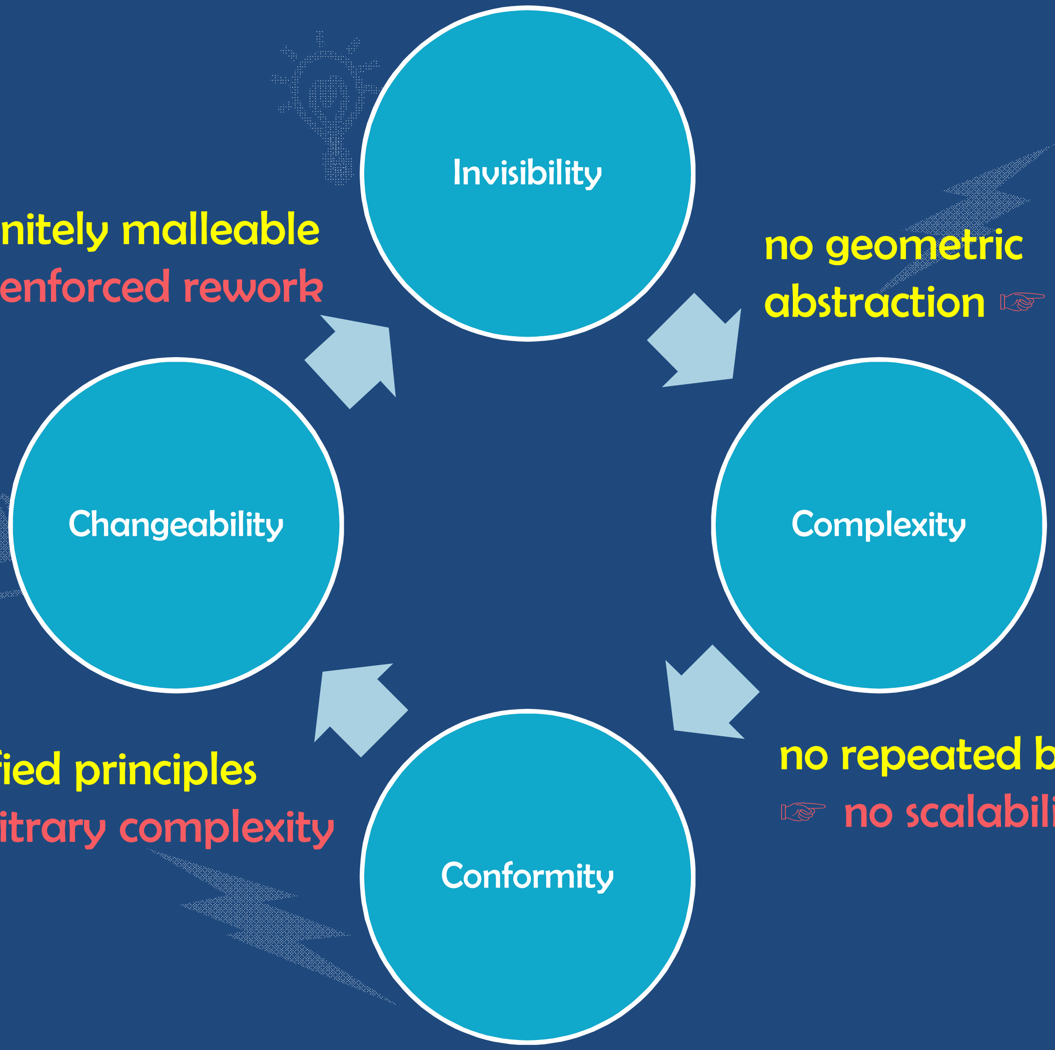
Essences and Accidents

- Complexity
  - No repeated elements are abound
    - Scaling-up does not merely mean a repetition of the same elements
  - Order-of-magnitude more states than digital computers
    - Flow-like architecture vs. invocation-like architecture
    - No black-box abstraction; low reusability & optimizability
  - The most complex entities than any other human construct, for its size
    - Inherently hierarchical structure
    - Non-linear increase with size
  - Not only technical problem, management problems come

Essences and Accidents

- Conformity
  - No unifying principles as with Physics
    - “there must be simplified explanations of nature, because God is not capricious or arbitrary” said Einstein
  - Arbitrary complexity caused by various people
    - last arrival on the scene & most conformable
- Changeability
  - Infinitely malleable and easy for change
    - SW is hero or zero?
  - Embedded in a cultural matrix of applications
    - Various stakeholders with different interests and knowledge

Essences and Accidents



Invisibility

no geometric abstraction ☞ order or magnitude

Complexity

no repeated building blocks ☞ no scalability

Conformity

no unified principles ☞ arbitrary complexity

Changeability

Infinitely malleable ☞ enforced rework

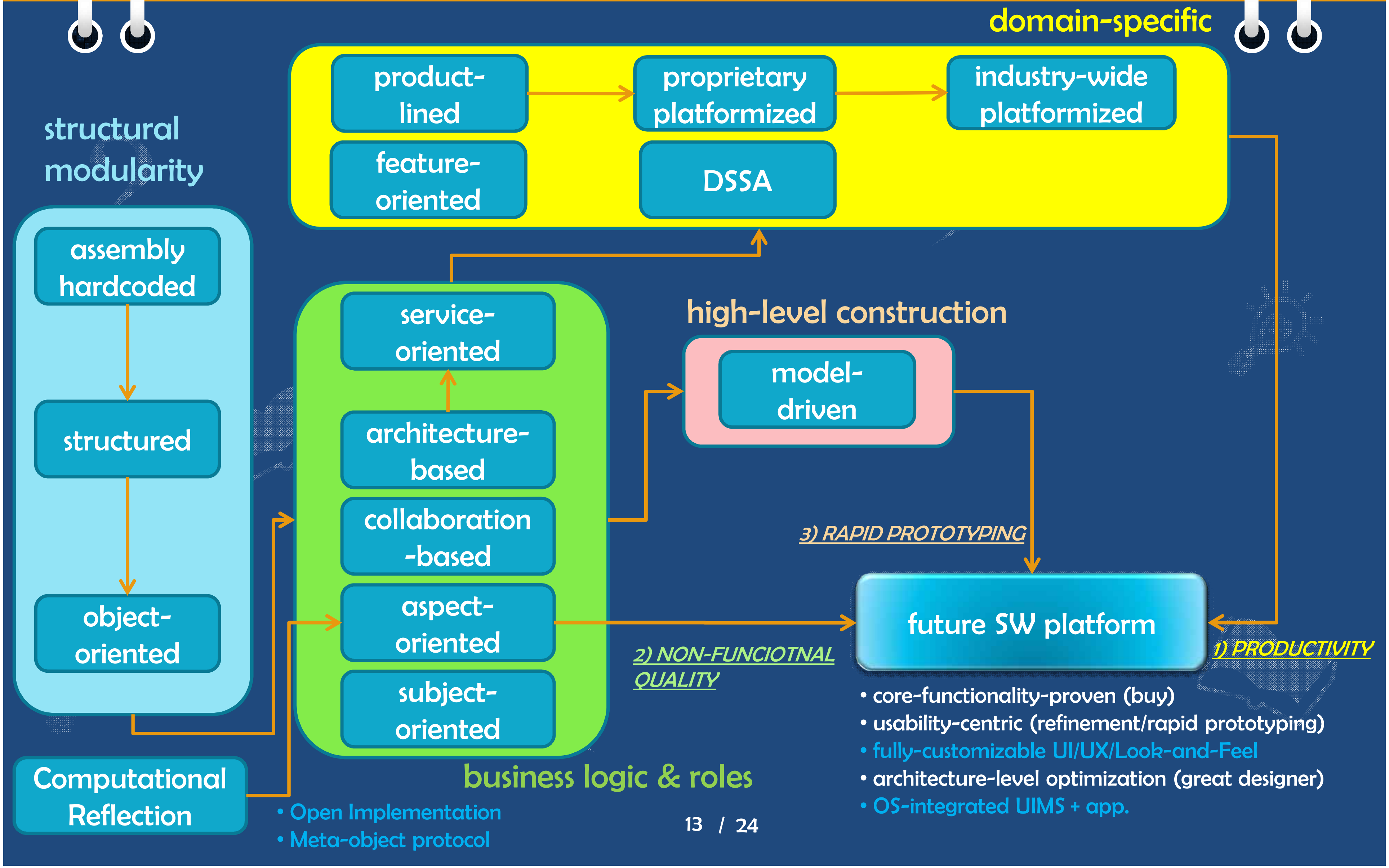
Essences and Accidents

- Breakthroughs against accidents
  - High-level languages
    - conceptual constructs: operations, data types, sequences, and communication
  - Unified programming environments
    - Integrated libraries, file format, tool benches, testing & debugging,
- Hopes for the silver
  - Object-Oriented Programming
    - ADT & hierarchical types
  - Automatic programming (since 1985)
  - Graphical programming
  - Program verification

No Silver Bullet

- Revolutionary or incremental advances towards "essences" and "accidents"?
- Productivity equation
  - $time\_of\_task = \sum_{n \in i} (frequency)_i \times (time)_i$
- Promising attacks on Conceptual Essence
  - Buy vs. build
    - Firstly mentioned in the NATO Software Engineering Conferences, 1968
  - Requirements refinement and rapid prototyping
  - Great designers

Evolution of SW Development Methods



domain-specific

product-lined

feature-oriented

proprietary platformized

DSSA

industry-wide platformized

structural modularity

assembly hardcoded

structured

object-oriented

service-oriented

architecture-based

collaboration-based

aspect-oriented

subject-oriented

high-level construction

model-driven

3) RAPID PROTOTYPING

future SW platform

1) PRODUCTIVITY

2) NON-FUNCIOTNAL QUALITY

- core-functionality-proven (buy)
- usability-centric (refinement/rapid prototyping)
- fully-customizable UI/UX/Look-and-Feel
- architecture-level optimization (great designer)
- OS-integrated UIMS + app.

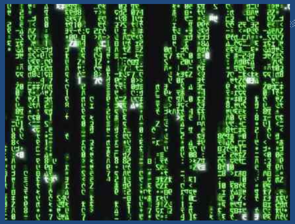
Computational Reflection

business logic & roles

- Open Implementation
- Meta-object protocol

Computational Reflection

- Definition
  - “a computational process that is able to reason about itself” by Brian Smith (1982)
  - “self-referential behaviors” in computational process

- Analogies
  - program expression ⇔ program data
  - metaphor ⇔ object
  - control program ⇔ robot arm
  -  ⇔ the Matrix

Computational Reflection

In view of Instruction Set Architecture (ISA)

```
L1    ...
      load      r10,      [pc+6]
      i-code    r11,      "add"
      cmp       b0,       r10,      r11
      br        b0,       L2
      i-code    r10,      "br  L1"
      store     [pc+1],   r10
      add       r01,      r02
L2    ...
```

In high-level programming

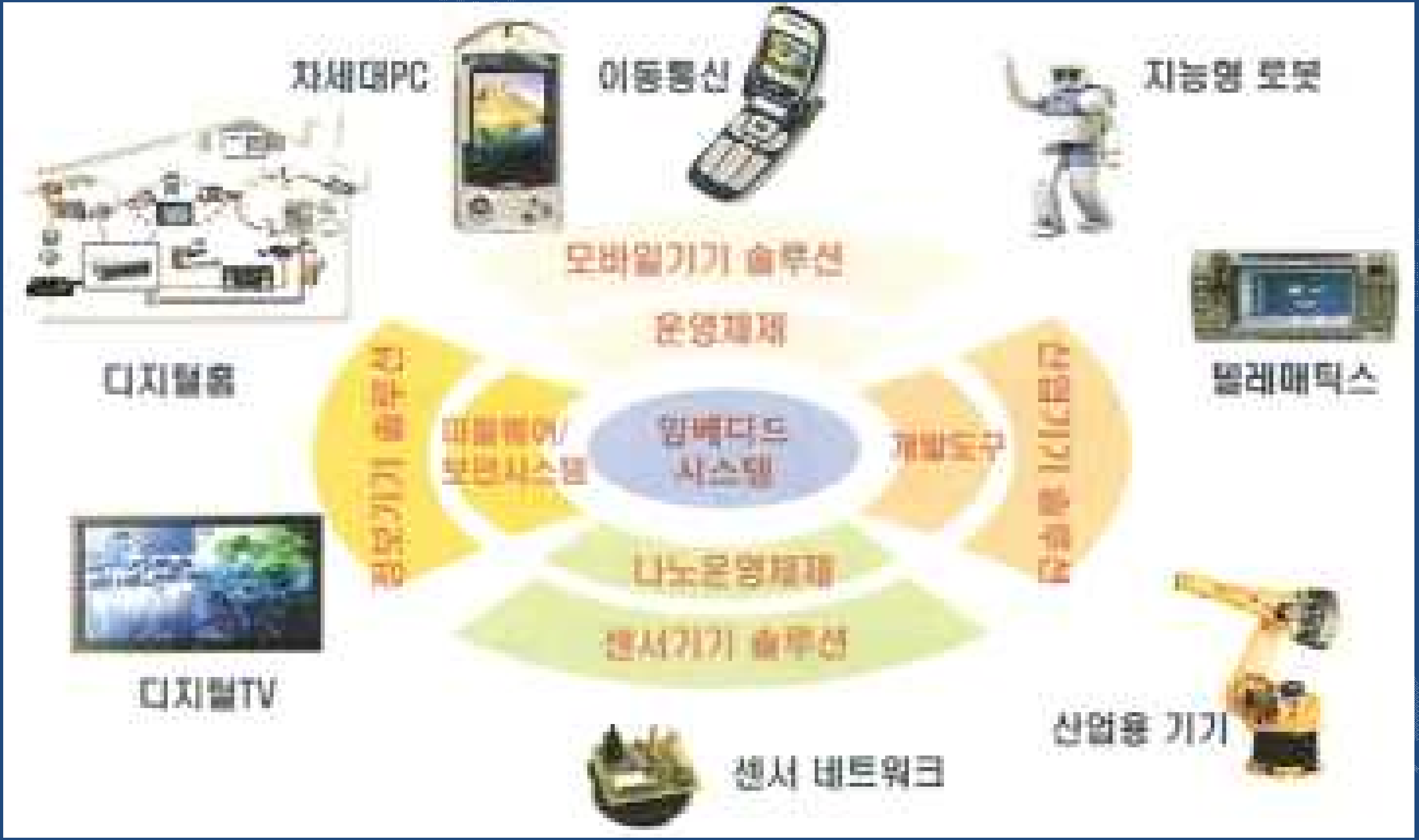
```
Class cls = Class.forName("Foo");
Object foo = cls.newInstance();
Method method = new Method("hello() { System.out("hello"); }");
cls.add(method);
cls.invoke(foo, "hello", null);  // ok?
```

Computational Reflection

- For further evolvability
  - Dynamically evolvable
  - Autonomously adaptable
  - Context-aware
- For higher modularity & reusability
  - Separation of cross-cutting concerns
  - Late binding to non-functional requirements
  - Building blocks with black-box abstraction

Future Competencies

What's Major Volume in SW industries



Classic Embedded Software

- Resource-constrained development and usage environments
  - e.g. an objective function of cost, memory, performance, and physical dimension
- Targeted at single or restricted tasks
  - shorter obsolescence cycle and no general scheme for SW/HW optimization (e.g. router software)
- HW replacement for flexibility or cost
- Mostly small-sized but manually optimized
- Embedded to infrastructures, utilities, or automotive mechanics
- Mostly, quality can not be compromised for cost

Modern Embedded Software

- Smart products
  - available resources as in desktop application
    - e.g. TI's OMAP3430 (ARM v7, 800MHz)
  - general-purposed and open platform
    - e.g. Windows mobile, Android, LiMo, Symbian
  - major part of system in both function and size
  - major volume of market: mobile, home & work
  - Needs for seamless cooperation (IT convergence)

- Change in priority precedence
  - time-to-market >> cost > quality
    - getting generous about system shut-down (Microsoft)
    - contributed to a fast growth in the market ?

Future's Embedded Software

- Life-care products
  - embedded in all types of living spaces
    - brains, skins, bones, internal organs, artificial muscles, clothes, glasses, personal vehicle, healthcare or medical assistant, etc.
    - endow-able, or printable software system ?
  - commoditized and standardized platform of IT convergence
- Further change in priority precedence
  - quality >> time-to-market > cost
- Liability to show certifications in quality

Needed Roles & Activities

- Requirements engineer
  - modeling & analysis
- Usability engineer
  - system modeling in a usability view
  - usability evaluation
- Software architect
  - architectural design & analysis
  - trade-off optimization
- Software system tester
  - integration/system testing
  - formal verification of protocol
  - non-functional quality analysis
- Software developer
  - communication-enabling technology
  - system or infra software (e.g. OS or platform)

Closing Remarks

- Software is still high for its age, difficulty, and importance
- The point is to prepare software competencies demanded in future

References

1. Edward A. Lee, “What’s Ahead for Embedded Software?,” IEEE Computer, pp. 18-26, Sept. 2000.
2. "No Silver Bullet - Essence and Accidents of Software Engine ering", Brooks, F. P., IEEE Computer, vol. 20, no. 4, pp. 10-19, April 1987.
3. Robert L. Glass, "Software: Hero or Zero?," IEEE Software, vol. 25, no. 3, pp. 96, 95, May/June 2008
4. Kiczales, G, “Beyond the black box: open implementation,” IEEE Software, vol. 13, no. 1, Jan 1996.