

# Evolution in the Era of Component Based Software

University of Sannio  
3 May 2001

**Juan F Ramil**

**Meir M Lehman**

Department of Computing  
Imperial College  
180 Queen's Gate  
London SW7 2BZ  
tel. +44 (0) 20 7594 8216  
fax. +44 (0) 20 7594 8215

ramil@doc.ic.ac.uk  
<http://www.doc.ic.ac.uk/~ramil>

## Objectives

To briefly discuss issues related to **planning** and **management** of **components-intensive** software evolution

### Specific issues

- to show that **components** of **E-type software** will also **need to be evolved**
- to indicate some of the immediate **implications** of **laws** of software evolution **on component-intensive software** and their processes when components are provided from organisation-external sources
- to provide **recommendations** for **mitigating risks** associated with evolution of component-intensive software

## Trend to Component-intensive Software

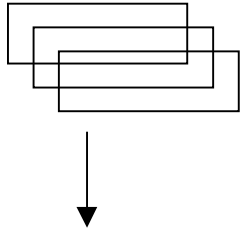
- **Software systems** are increasingly being assembled from **numerous self-contained elements**
  - each provides specified **functionality** to users or **fulfils a role** in system operation
  - each must also possess an **interface** for communicating with other elements
  - **examples** include GUIs, browsers, device drivers, network facilities, text editors, spreadsheets, e-mailers and so on
  - degrees of **integration** vary from **loosely** - such as integration via end user or files - to **highly coupled** systems - where each element implements a process in a concurrent system

## What is a Component?

- Generally accepted that a **component** is at most only to be **minimally modified**
- Assumed in present analysis that any **software element** obtained from an **outside supplier** should be treated in this way
  - **required** if one is to obtain **benefit** with respect to traditional development
- Related concepts
  - **Reuse**: raises some of the same issues
  - **COTS**: Commercial Off-the-Shelf Software
  - **GOTS**: Government Off-the-Shelf Software
  - **OS**: Open Source
  - **NDI**: Non Developmental Item - in some Government contracts the term is used to imply that item is already available

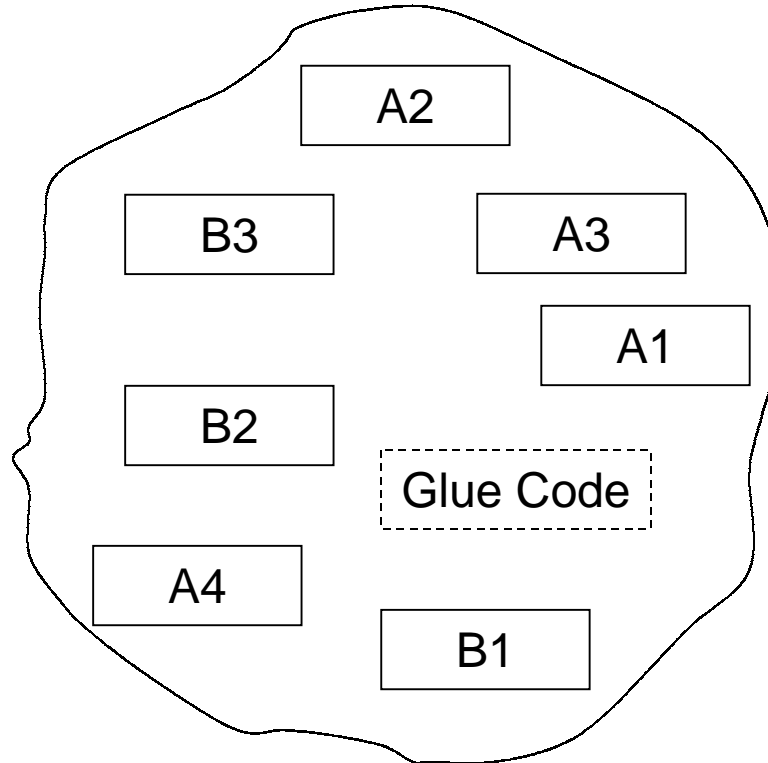
# Component Developers, Integrators and End Users

*Component Developer/Supplier 1*



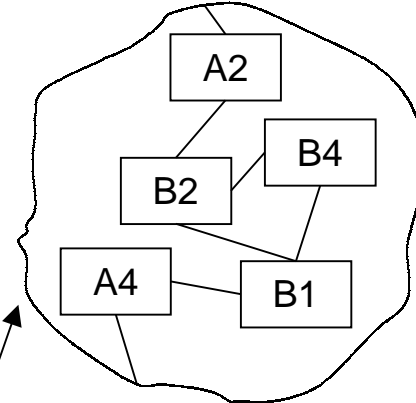
**Component A1,A2,A3...**

*System Architects/Builders (SA/B)*



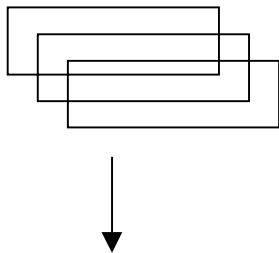
**Host System**

*End Users*

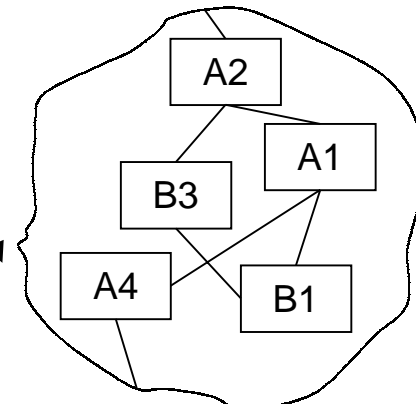


**Delivered & Configured System 1**

*Component Developer/Supplier 2*



**Components B1,B2...**



**Delivered & Configured System 2**

## Component-based Software Engineering (CBSE)

- Software development based on **combination** of a set of existing **components** was discussed in late 1960s - M D MacIlroy, *Mass Produced Software Components*, in Naur P and Randell B (eds.), *Software Engineering*, Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968, January 1969, 231 pps
- **Only recently** has practical application been widely explored
- From point of view of system architect & builder, **integration** tends to **dominate**
- Relatively **less effort** devoted to traditional activities, e.g., design, coding, testing
- Introduction based on
  - expectation of **faster** and **more effective preparation** of **first version** of **host system**
  - component development **costs shared** by many integrators and, ultimately, many end-users
  - need for **evolution** initially **addressed** by component **introduction** and **disposal**
- In 2002, according to Voas, **40 percent** of a typical corporation's IT portfolio will be COTS software - Voas J M, *Disposable Information Systems: The Future of Software Maintenance?*, Journal of Softw. Maint: Res. Pract. vol. 11, n. 2 143-150, 1999

## First and Sixth Laws - Continuing Change and Continuing Growth

- **Operational domain** undergoes **continuing change** - as explained in previous lectures
  - driven by technological advances, human desire for improvement, domain changes
  - installation and operation of system changes domain
- Software desired properties is **finite** set, properties of domain and application are potentially **infinite** - those properties include functional and non-functional attributes
  - thus, properties excluded by **bounding process** become source of limitation, error
  - embedding of **assumptions** in host system - and its components - inevitable
- Operational domain changes, invalidation of assumptions, source of **unending fixing, adaptation, enhancement**, that is, **evolution** - encompasses both development and maintenance

**Host system** and its **individual components** must be **evolved** if it is to remain satisfactory to its stakeholders

## Components of *E*-type Host Systems

- ***E*-type** components
  - **solve** a **problem** or address an application in a **real-world domain**
  - **stakeholders' satisfaction**, primary criterion of success
  - as for *E*-type systems, **continuing evolution**, an **inescapable fact of life**
- ***S*-type** components
  - **satisfaction** of an **specification**, primary criterion of success
  - constitute “**bricks**” from which systems are built
  - **acquire** *E*-type characteristics when **integrated** - e.g. act of integration will inevitably lead to assumptions about other components behaviour, about properties of wider domain within which host system will operate and so on
- Thus, **all** components of *E*-type host systems are subject to evolution
- **Pressures** for component evolution, **significant** if host system is to evolve mainly by introduction and disposal of components
- Achieving **evolution by “glue code” or “glueware”**, e.g., component wrapping, diminishes need for component evolution, but **reverts** to **traditional practice**

## First and Sixth Laws - Continuing Change and Continuing Growth (cont.)

### Scenario 1

- Component **upgrade by external organisation**, e.g. component developer/supplier
  - **delays** may occur between identification of need and its implementation
  - **potential conflict** between needs of several integrators and end users
  - integrator may have to implement **temporary patches** - a self defeating exercise

### Scenario 2

Component **upgrade by system architect and builder**

- **de facto reversion** into traditional evolution
- **denies original justification**, raises maintenance burden, increases cost
- requires **full access** to component **source code**, expertise, documentation, etc

### Scenario 3

Evolution by means of “glue code”, wrappers and, in general, bypassing components

- bypasses need for component modification, at least in the short term
- also implies **de facto reversion** into traditional evolution
- there exist limits to what can be possibly achieved by wrapping

All scenarios involve **risks** for the system architect/builder and end-users

Planning of **division of labour** between SAB and suppliers, essential

## Immediate Implications of First and Sixth Laws

- Unless periodically re-written - clearly uneconomical -, components must be **evolved**
  - suppliers and integrators are expected to be subject to laws of software evolution

How?

- **Component developers/suppliers** following traditional paradigm - the majority
  - the laws, their implications and rules and guidelines for evolution planning and management - discussed in a previous lecture - would apply to them as they are
- **System architects/builders** face some major issues
  - they are ultimately responsible for host system, must face end-users
  - however, **evolution** of host system is significantly **outside their control**
  - laws point towards some of the issues and how to address them
  - thus laws are useful to analyse **implications** to system architects/builders
  - await empirical validation - for the moment provide sound hypotheses

What are some of these implications?

## Second Law - Increasing Complexity

- As said, **introduction** of components may lead to **faster** and more effective preparation of **initial** version of host system
- Component suppliers may find it **increasingly difficult** to evolve components
  - effects of change upon change upon change - see for example, Parnas DL, *Software Aging*, Proc. 16th Int. Conf. on Softw. Engineering, May 16-21, 1994, Sorrento, Italy, pp 279-287
  - **orthogonality** of needs from diverse set of markets, integrators
  - complexity control brings long term benefits, but market and other pressures may enforce **short term view** - anti-regressive, complexity control activity, neglected
- These and other circumstances may force suppliers to **withdraw support** with consequent ripple effects
  - system architects/builders may have to replace components not longer supported
- This, so far, relates to complexity of individual components

What about complexity of host system as a whole and of its “glue code”?

## Second Law - Increasing Complexity

- With regards to host system, components are **primitives** of high level language
  - complexity constraints are difficult to avoid
  - **constraints** implied by 2nd law may **re-appear** at the higher level of abstraction
- Unless work is done to control complexity - both at host system and component levels - resultant increase will be reflected in **growing maintenance burden**
  - leading to a decline in maintenance productivity
  - consistent with observed trends of decreasing growth rate - as discussed in previous lectures

## Third and Fourth Laws - Self regulation and Conservation of Organisational Stability

- Laws refer to **global processes** - as discussed in previous lectures - of both supplier and architects/builders
- Processes of supplier and architects/builders **closely bound up** with local and inter-connecting feedback loops
  - bi-directional communication is essential
- **Bottle-necks** expected to occur in supplier's organisation
  - many customers, the integrators, with possibly dissimilar, orthogonal needs
  - market pressures, technological change
- Laws **imply constraints** in supplier's process performance
  - suggest existence of bounds to sustainable rate of evolution of individual components
  - interpreted as a consequence of feedback loops in processes

What about the system architects/builders?

## Third and Fourth Laws - Self regulation & Conservation of Organisational Stability

- **Integrator organisations** are likely to involve **smaller teams** than would have been required if traditional approach were followed
- Thus, **impact** of contributions of individual team members expected to be **larger than in traditional processes**
  - higher statistical variance in performance than in traditional processes
  - a hypothesis for empirical investigation
- Third and fourth laws suggest that **historical data** is **useful in planning** the future
  - lack of data on component-intensive evolution makes it difficult
  - unresolved measurement issues - e.g. how to measure size of host system?

## Fifth Laws - Conservation of Familiarity

- **Observations** suggest that an **above-the-historical-average** increase in growth rate tend to be followed by a decrease in growth rate, in some cases a decrease in size - this has been discussed in previous presentations
- Attempts to achieve **growth rate above historical average** may be **difficult**
- As name of law indicates, phenomenon has been interpreted as need for familiarisation of all those involved, when a large functional increment is introduced

Rate at which component supplier and system architects/builders can respond to domain changes is limited,  
excessive pressures for growth may result in product instability

## Seventh Law - Declining Quality

- Grounded on observation that **user needs** and **perceptions change** over time
  - a demand source for component and host system maintenance and evolution
- Quality may involve may different **dimensions**: usability, reliability, evolvability...
  - reflect fitness for purpose, match between system and the surrounding domains
  - Gilb's view: all attributes of interest must - and can - be measured, baselines established
  - Maibaum warns that “-ilities” are **dispositions**, not immediately observable
  - thus, one must be cautious with their operationalisation
- Component suppliers and system architects/builders must **share** maintenance and evolution **burden**

Fail to do so may lead to serious long-term quality problems

## Eighth Law - Feedback System

- Feedback nature of processes **difficult to deny**
- Some **challenges**
  - supply chain dynamics - one of Forrester's studies in his *Industrial Dynamics*, 1961
  - system architects/builders may have to accept releases with changes not required by them
  - delays in required changes due to work rate constraints, compatibility problems
  - ripple effects related to changes in technology, business environment, market
- **System dynamics** expected to play a role and must be taken into account
  - expected to apply to supplier and integrator's processes
  - localised, constrained effect of improvements to immediate technical process
  - process modelling, both black-box and white-box may be of help
- **Global Process** more **diffused**, **distributed** than in traditional development
  - less pressure to achieve high levels of process definition - see Morisio M et al
  - additional challenge in planning and management

## Some Recommendations

- **Long-term** component support
  - ensure that long-term support will be provided by supplier
  - negotiation essential
- Component are built with a **conceptual framework** in mind
  - be sure that it fits the host system and application domain
- Role of **assumptions**
  - capture, record, structure and periodically review them
  - give preference to components for which a precise specification exists
  - require assurances that as components are evolved, customers will be informed whenever assumptions
  - component documentation must be kept up to date as components are evolved
  - introduce process steps that address impact of embedded assumptions
- General evolution rules and guidelines may also be helpful

“An application system containing multiple COTS products requires a different kind of maintenance planning from that for a mostly custom system - Hyberston et al, 1997

## Some Recommendations (cont.)

- Adjust process models and activities to the realities of CBSE
  - in addition to activities addressing role of assumptions, consider activities such as component evaluation, selection, testing, identification of requirements embedded in components, etc
- Configuration management systems and CASE use a variety of COTS tools such as design, change control, testing and so on - an additional risk source

## Summary

- **Components** of real-world applications must be **evolved**
- Evolution of components, and hence, of host system **outside** control of system architects/builders
- Laws suggest that introduction of components, while **initially advantageous**, may result unreliable, expensive and risky in the long run, unless thoughtfully addressed
- **Laws** provide a basis and a **framework** to discuss planning and management of component-intensive evolution processes
  - suggest recommendations and guidelines to mitigate risks
- Despite risks, **CBSE** expected to be **increasingly popular** over coming years
  - strong short-term economic justification
  - long term evolutionary behaviour haven't been studied
  - provides material for further empirical studies of evolution
  - laws provide a source of hypothesis for such studies

Underlying message: *“Proceed with caution, forget the panacea”*

## Some Sources

### *Journal and Conference Papers*

- Carney D, Hissam, S A and Plakosh D, *Complex COTS-based Software Systems: Practical Steps for Their Maintenance*, J. of Softw. Maintenance : Res. and Pract., vol. 12, n. 6, 2000, pp. 357 - 376
- Hybertson D W, Ta A D, Thomas W M, *Maintenance of COTS-intensive Software Systems*, Journal of Software Maintenance: Research and Practice, vol. 9, n. 4, 1997, pp. 203 - 216
- Lehman MM and Ramil JF, *Software Evolution in the Age of Component Based Software Engineering*, IEE Proc. Softw., sp. issue on Component Based Software Eng., vol. 147, n. 6, Dec. 2000, pp. 249 - 255, earlier vers. as Tech. Rep. 98/8, Imp. Col., London, Jun. 1998
- Morisio M, et al, *Investigating and Improving a COTS-Based Software Development Process*, Proc. ICSE 22, 4-11 June 2000, Limerick, Ireland, pp. 32 - 41
- McKinney D, *Impact of Commercial off-the-shelf (COTS) Software on the Interface between Systems and Software Engineering*, Proc. of ICSE 99, May 1999, Los Angeles, CA, pp. 627 - 628
- Voas J M, *Disposable Information Systems: The Future of Software Maintenance?*, Journal of Softw. Maint: Res. Pract. vol. 11, n. 2 143-150, 1999
- Wu Y, Pan D and Chen MH, *Techniques of Maintaining Evolving Component-based Software*, Proc. ICSM 2000, 11-14 Oct., San Jose, CA, pp. 236 - 246