

---

# FEAST/2 - Some Results, Current View, Issues

FEAST/2 Internal Workshop  
Dept. of Computing, Imperial College, London  
19 December 2000

Juan F Ramil  
(with contributions by M M Lehman, G Kahen and S F Lim)

Dept. of Computing  
Imperial College  
180 Queen's Gate  
London SW7 2BZ  
ramil@doc.ic.ac.uk  
<http://www.doc.ic.ac.uk/~ramil>

---

# Introduction

---

- Evolution, including maintenance, involves from 50 to 80 percent of all resources allocated to software organisations
- Evolution is a **fact of life** of computer application and of software
- It involves world-wide large amounts of resources - 50 to 80 percent of the software business estimated in £200 billion in sales (1996) according to a report from Scottish Enterprise & Scottish Software Federation
- Within this context we have been pursuing the question
  - are there **similarities** in **evolutionary behaviour** of different software systems?
- In affirmative cases, the following questions have become relevant
  - what are these **similarities**
  - are there **behavioural invariants**?
  - what **common phenomena** do they reflect?
  - are there **implications**?
- Ultimate goals of the study include improving long term evolution: e.g., **maximise benefit** to stakeholders, **mitigate risks**

- Systems studied:
  - **ICL** VME Operating System Kernel, **Lucent** Real Time System, **Logica** FW, **BT** CSS Operations Support System
  - **Matra-BAe** Defence System, an example of development as evolution
  - behaviour compared to that of **IBM OS/360**, first system studied in late sixties
- Study based on analysis and interpretation of metric data
  - **start with** a small set of **attributes**, **extract** metrics, **build**, **calibrate**, **interpret**, **validate** models, achieve **top-down** refinement through iteration
- Outputs
  - **patterns** across different systems and domains have been identified
    - encapsulated for example in laws of software evolution
  - **empirical generalisations** have emerged - inputs for a theory of software evolution
  - **decision support** models and tools proposed - system dynamics models, planning rules, guidelines, suggestions for tools, models for resource estimation

# A possible set of 'evolution size' indicators

<b>Indicator Based on</b>	<b>Abbreviation (Full name)</b>	<b>Description</b>
		Symbol '#' to be read as 'count over interval $t$ to $t+I$ '
Modules	<i>ModifHandlings(t)</i> (Modification Handlings)	# of changes to modules as reflected by number of change log modification entries, referred to as <i>handlings</i>
Modules	<i>ModulesChanged(t)</i> (Modules Changed)	# of modules modified
Modules	<i>ModulesCreated(t)</i> (Modules Created)	# of modules added to the system
Modules	<i>TotalHandlings(t)</i> (Total Number of Handlings)	# of total change log entries, that is, including both creation entries and modification entries
Modules	<i>ModulesHandled(t)</i> (Modules Handled)	# of modules, either added to the system or modified, or both (if both, module is counted once)
Subsystems and Modules	<i>SubsysChanged(t)</i> (Subsystems Changed)	# of subsystems that underwent modifications to their modules
Subsystems and Modules	<i>SubsysHandled(t)</i> (Subsystems with Modules Handled)	# of subsystems that underwent either additions or changes to their modules, or both (if both, <i>subsys.</i> is counted once)
Subsystems and Modules	<i>SubsysInclCreations(t)</i> (Subsystems that include Modules Created)	# of subsystems which underwent module additions

In general can be extracted from change-log records and other sources

# Example of data source: change-log in program header

---

```
; Change log:-  
;  
;  
; 20NOV85                HJ  
; Modified to properly support reprint of alarms  
;  
;  
; 22NOV85                DV  
; Modified to use LOGINT module to interpret alarms, as alarms are  
; just logs in VAX TMAN, and virtual address space is not limited!  
;  
;  
; 01DEC85                DV  
; Modified to use printer sequence number proof accumulator  
;  
;  
; 04DEC85                PC  
; Modified to use VMS port names instead of RSX port numbers  
; 31Jul87                HC  
; Made size of lines area to be driven off the symbol  
; sok.APPFIL_size from FW$:OPTIONS.MAR.  
;  
;  
; 20th November 1988    HJ  
; Changed ln:w.flags to ln:l.flags  
;  
;  
; 20th November 1988    JC  
; (implemented by HJ)  
; If the output sequence number gets out of sync with the proof so that  
; the proof is larger of the two, then the alarms printer will print  
; all the alarms from the time the system was cold started. If this  
; situation occurs then the proof will be reset to the output sequence  
; number.
```

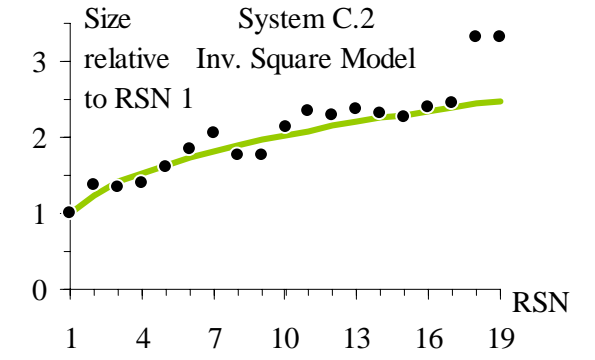
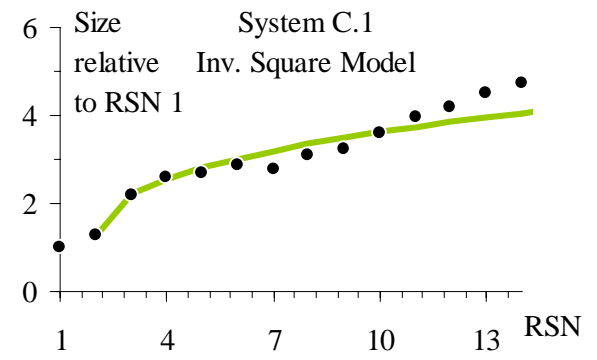
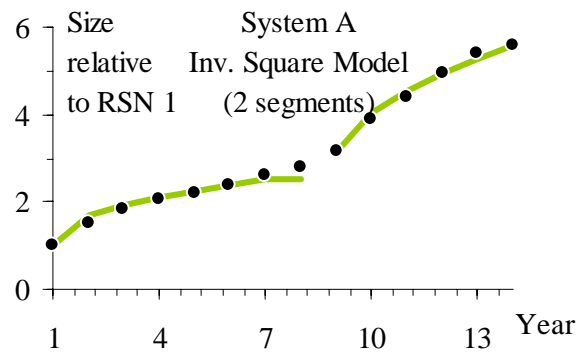
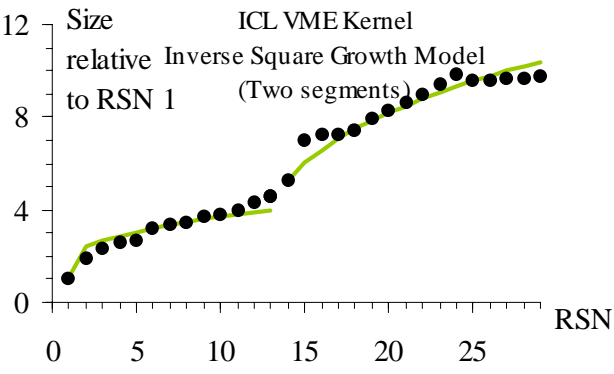
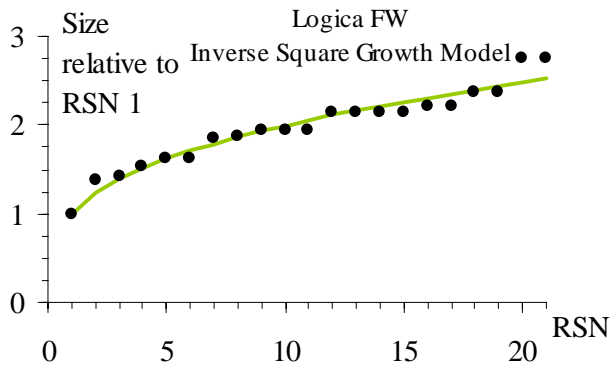
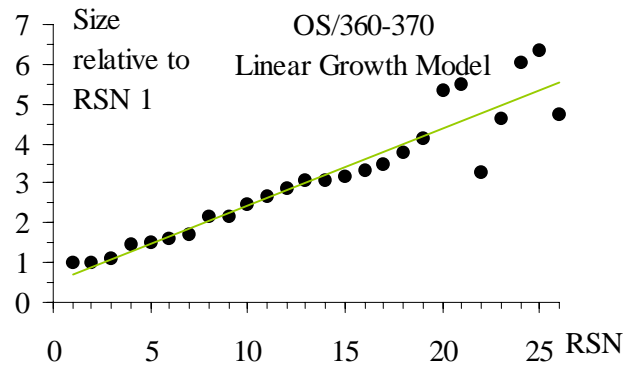
## **Observation 1: decaying growth with regeneration points**

---

- **Decaying growth rate**, over one or more segments, has been observed in all available data - with exception of OS/360
- Following Turski, **inverse square models** have been fitted to all of relevant data - model discussed in FEAST literature at <http://www.doc.ic.ac.uk/~mml/feast>
- Interpreted, in general, as an **indicator of growing complexity** - other interpretations not to be excluded
- **Regeneration** can be achieved by, for example, **process change**, **restructuring** of the software or **consolidation** of parallel versions
- **Regeneration points** are apparent in three of the systems growth trends
  - consistent with Bennett and Rajlich's view of staged software evolution

# Observation 1: decaying growth with regeneration points

- Size indicated here as the total number of modules



Note superimposed *ripple*

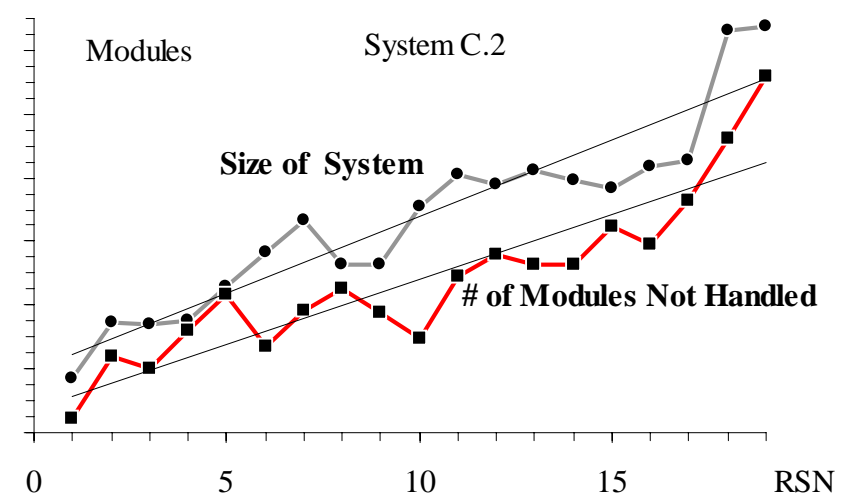
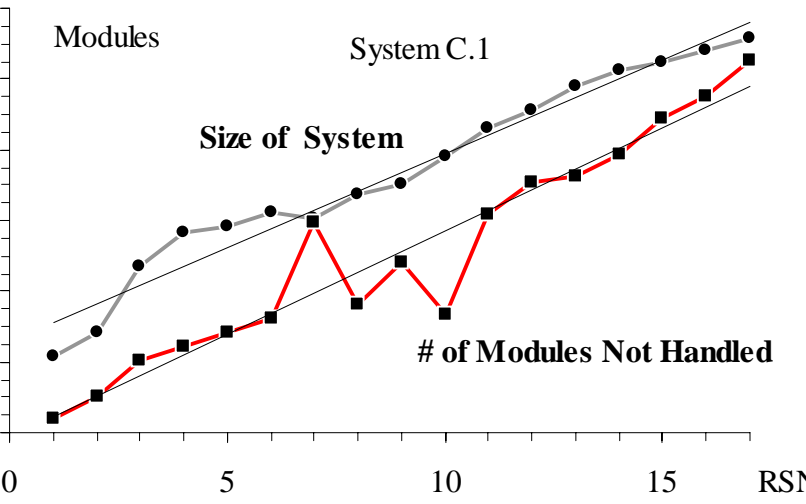
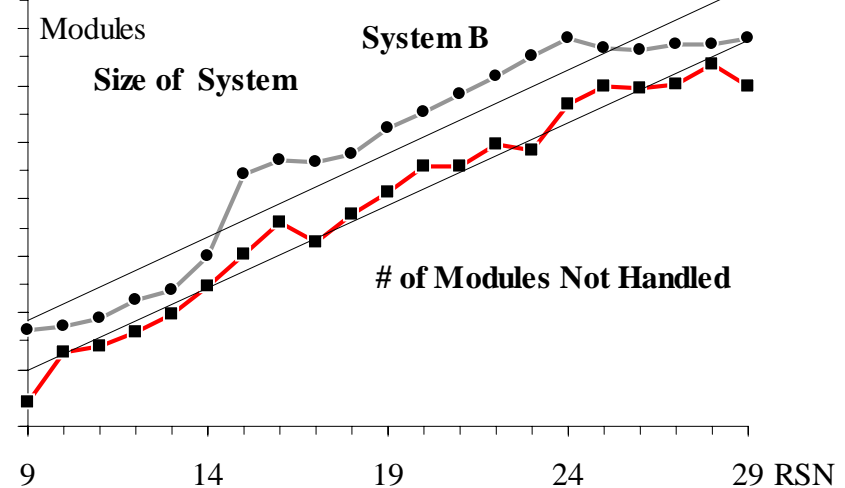
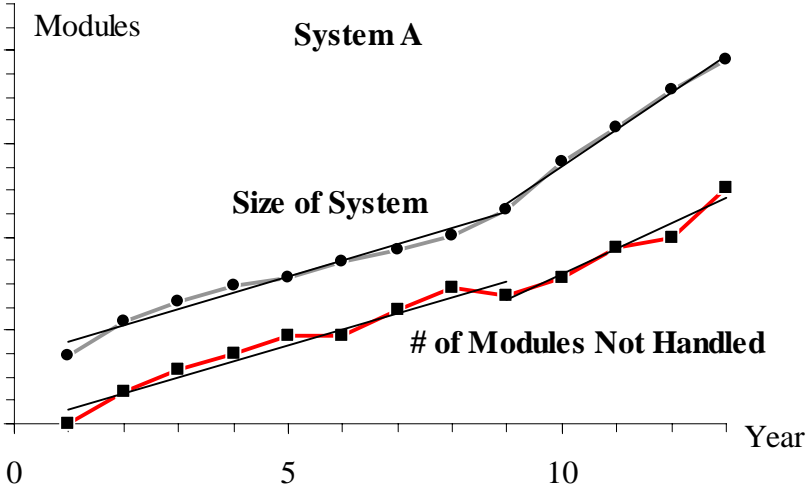
## Observation 2: *ripple*

---

- Imposed on growth trends, a *ripple* is in general visible
- Observation in OS/360 (1972) triggered feedback hypothesis
- Interpreted as system dynamics stabilisation
- Observations
  - ripple **not** resulting from **direct conscious** human control
  - process and product attributes conscious controlled by managers relate for example to resources usage defect reports and rates, etc. not directly to size
- Ripple can be interpreted as consequence of 2 types of feedback forces
  - those that tend to **increase** evolution rate, driven by need to increase profit or market-share
  - pressures to **decrease** them such as those that emerge from communication losses in large teams, increasing complexity and orthogonality of system with the application, need for clean up after major growth
- Aggregated consequence suggests process *self-stabilisation*

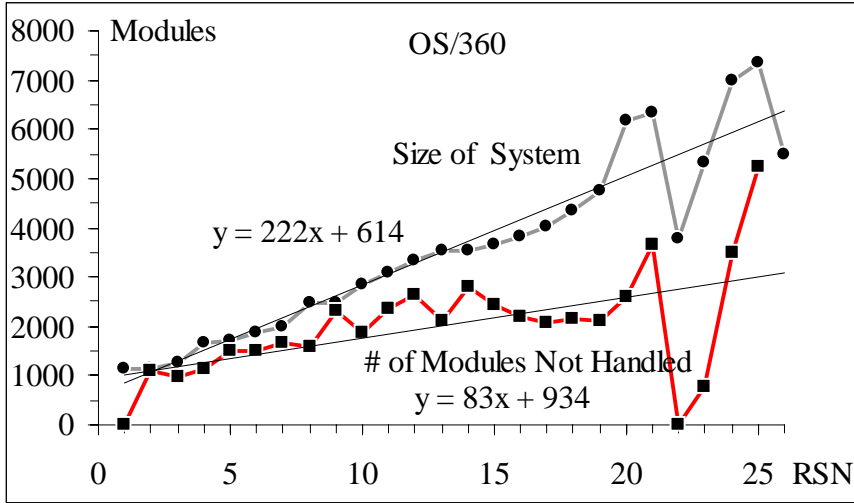
# Observation 3: fraction of system handled for stable growth

- Fraction of system handled constant or decreasing - modules handled, defined in chart 4, appear as distance between grey and red lines - a characteristic of *healthy evolution*?



# Observation 3: fraction of system handled and stable growth (cont.)

- Example of unstable evolution
- Divergence of *growth* and *modules not handled* trends is apparent from RSN 16 onwards - could have provided an early warning?



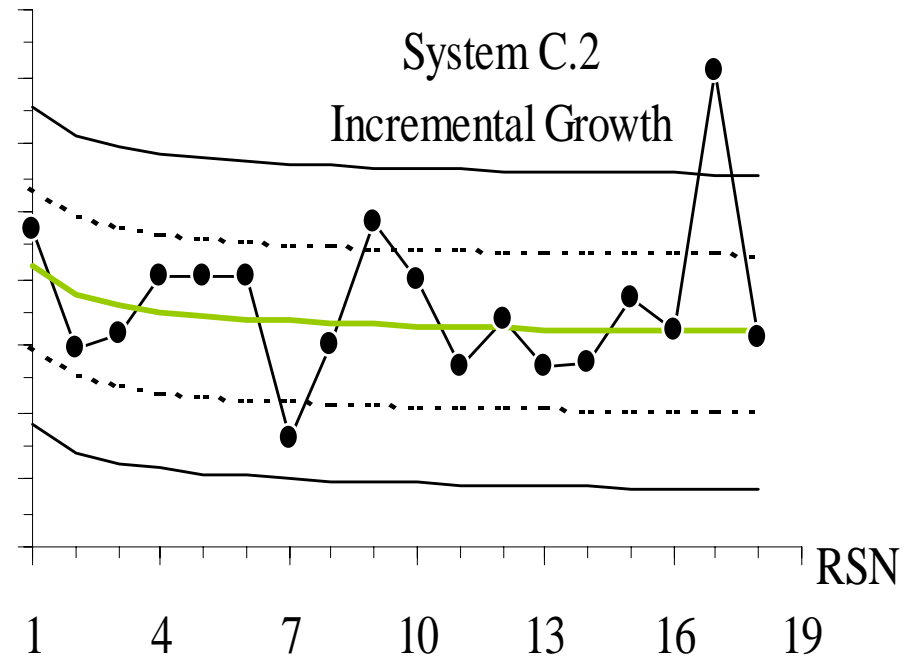
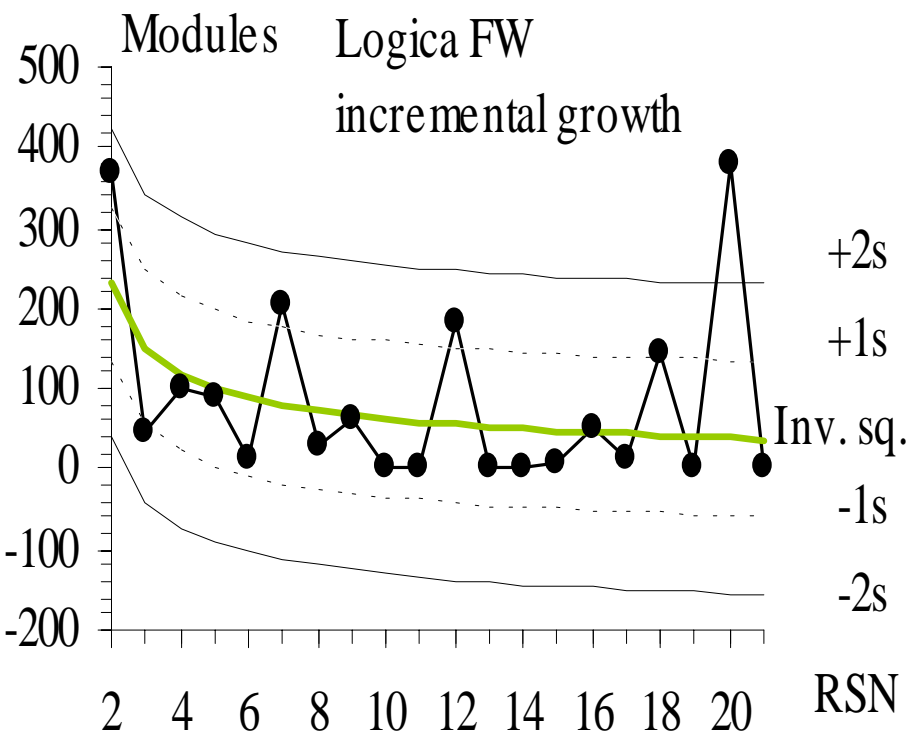
## Observation 4: three levels of incremental growth

---

- Three levels of release incremental growth identified
  - **small** increment of growth can be adequately validated release to users and safely followed by a further satisfactory release
  - **somewhat larger** growth is likely to lead to delays in release or high fault levels after release and therefore to one or more follow-up releases
  - **too large** a growth is likely to lead to serious problems, in the limit to "catastrophe"
- We have here deliberately refrained from defining small, somewhat larger, and too large
- There is some evidence that  $\ll m, \sim m, > m+2s$  threshold frequently observed and used in statistical process control is relevant here
  - $m$ , for example, as an *incremental* inverse square model and  $s$  as standard deviation of the residuals

# Incremental growth patterns (cont.)

- Incremental inverse square model and standard deviation of residuals as means to identify levels  $\ll m, \sim m, m + 2s$



- Incremental growth outside thresholds is likely to follow by a small or even negative increment - behaviour interpreted as system dynamics self-stabilisation of the global process

## Summary of status of support for the laws

---

- Direct metric support for of 6 of 8 laws or their slightly refined versions
- Seventh law supported by theoretical reasoning
- Anomalies have been observed with respect to fourth law
- Fourth law: *segments* in module handled rate - have prompted a refinement of the law
- In the face of theoretical reasoning, eight law simply **cannot be denied** - the questions remain what class of feedback system, what are the underlying mechanisms, what are their implications and, finally, how to exploit them
- FEAST has made some progress in addressing these questions

# Development as evolution

---

- Matra BAe Dynamics defence system
  - an example of **development** as **evolution**
- System size **constrained**, in fact constant or reducing in spite of intensive continuing development effort over five years or so
- An anomaly with respect to sixth law?, a **different** type of system?
- System **not installed in the operational domain** until development finishes
- **Work intentionally done to reduce size** of code
  - One can argue that in spite of this functional power is increasing
- Most probably systems represents a form of **evolutionary development** as proposed by Tom Gilb
- Evolution of the application takes place over **generations** of the defence system

## **Hypothesis: influence of the outside loops**

---

- Hypothesis based on phenomenological reasoning about the process, from our observations and from system dynamics modelling effort:
  - the process is a **multi-level feedback** system
  - the **behaviour** of the **global process** as seen from the outside is likely to be **highly sensitive** to **properties of higher levels** or **outer loops** of the process
- This needs a lot of explanations, definitions, etc, etc.

# Practical outcome 1: Rules and Guidelines for Software Evolution Planning & Management

---

- | Item | Description                                                                                                                                                                      |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.   | Prepare comprehensive specifications and <i>maintain</i> them updated.                                                                                                           |
| 2.   | Formalise specifications (long term goal).                                                                                                                                       |
| 3.   | Capture, document, structure and retain assumptions in specifications.                                                                                                           |
| 4.   | Verify assumptions as part of verifying specifications.                                                                                                                          |
| 5.   | Capture, update and fully document assumptions during design, implementation, integration and validation work.                                                                   |
| 6.   | Develop and use tool support for specification work and recording of assumptions in structured form (as database), classified by appropriate categories, throughout the process. |
| 7.   | Create and update comprehensive documentation to minimise the impact of growing complexity.                                                                                      |
| 8.   | Apply conscious effort to control and reduce complexity and its growth.                                                                                                          |
| 9.   | Document assumptions, design and implementation rationale underlying a change.                                                                                                   |
| 10.  | Periodically review the assumption set.                                                                                                                                          |
| 11.  | Plan for clean-up releases after the addition of major increments of and/or large changes to functionality.                                                                      |
| 12.  | Observe safe change rate limits in planning and implementing change and evolution.                                                                                               |
| 13.  | Constrain scope and size of a release functional increments based on past successful incremental growth.                                                                         |
| 14.  | Alternate major (functional enhancement, extension) and minor (clean-up, restructuring) releases.                                                                                |
| 15.  | Validation of all changes including local and incremental changes, must also address interaction with and impact on that part of the system that is not changed.                 |
| 16.  | Such validation must include a check of the continued validity of assumptions.                                                                                                   |
| 17.  | Assess domain and system volatility and take this into account in implementation.                                                                                                |
| 18.  | Manage and control complexity in its many aspects.                                                                                                                               |
| 19.  | Systematically evaluate the need for and undertake <i>anti-regressive</i> work such as, for example, complexity control.                                                         |

# Practical outcome 1: rules and guidelines for software evolution planning & management

---

(cont.)

Item Description

20. Collect, plot and model historical data to determine patterns, trends and rates of change and growth.
21. Establish baselines of key process and product measures over time and releases.
22. Use recent data to assess, recalibrate and improve metric-based planning models.
23. Use the ' $m+2s$ ' or similar criteria to determine safe, risky or unsafe growth increments.
24. When a release with large incremental growth appears to be required, split into several releases as in *evolutionary development*.
25. Develop automatic tools to support data collection and modelling.
26. To minimise interaction between system elements, follow recognised software engineering principles, *information hiding* for example, when implementing and evolving functionality.
27. Consider, model and manage the *global* process that embeds the technical process.
28. Consider, model and manage formal and informal organisational links as part of the global process.
29. Model the system dynamics of the global process.
30. Use dynamic models to identify interactions, improve planning and control strategies.
31. Use dynamic models to plan further work.
32. Use dynamic models when seeking process improvement.
33. Estimate likelihood of change in the application domain and its sub-domains, and their impact on assumptions.
34. Conduct both periodic and event-triggered review and assessment of likelihood of change in assumption sets.
35. Improve questioning of assumptions by, for example, using independent implementation and validation teams.
36. Provide ready access by evolution teams to all appropriate domain specialists.

## **Practical outcome 2: effort estimation in software evolution context**

- Estimation in the context of software evolution
- Estimation are feedback control conjoint activities which cannot be seen in isolation
- Any industrial process operates *under closed loop*
- Feedback control as a cost driver
  - management policies play an important role in determining cost behaviour
  - in many cases, such control is distributed, not formalised
- Based on observations, a black box approach is suggested
  - detect segments of similar behaviour
  - calibrate models to most recent segment
  - check for changes and re-calibrate as required
- Piecewise models as an indicator of process change
  - in fact, a test for the reality of any process improvement, for example

**Practical procedure for estimation follows**

## Practical outcome 2: effort estimation in software evolution context

---

- **Extract** metric data from historical records
  - starting with measures of effort applied and evolution activity rate
- **Compile** historical records
  - as derived from, for example, configuration management databases, change-log records
- **Plot** data, **detect** behavioural changes and segment model accordingly
- **Detect** outliers
- **Identify** candidate model structures - linear, inverse square, other, etc
- **Obtain** model parameters for each of the relevant segments
- **Assess** predictive power of model
  - preferably using data subset different to that used to get model parameters
- If model is sufficiently precise, **apply** as **an** estimator
- If model is not sufficiently precise,
  - **refine** metrics list and the models until an acceptable level is reached or no further significant improvement can economically be achieved
- Periodically, **check** whether the model **retains** its **accuracy**
  - If there has not been significant **structural** process change, re-calibrate existing model.
  - **refine** existing model or **try** different model structure
- Periodically, **check** for large changes in activity rate, effort applied, and significant **structural changes** and
- **Update** model - once sufficient data on a new segment has been obtained
  - ‘Other factors’ may be considered during iterations of this procedure

## Related work

---

- Bennett K and Rajlich V - Durham U.
  - Stages in software evolution, case studies
- Cook S, Hi Y and Harrison R, U. of Reading
  - Software evolution, software evolvability and architectural issues
- Godfrey *et al* - U. of Waterloo, Canada
  - Open source systems (e.g., LINUX)
  - Observations of exponential growth in LINUX - since in open source evolution many of the feedback controls that exist in a traditional organisation do not exist, exponential growth in open systems may provide indirect support to the 'outside loops' hypothesis in chart 15. This remains to be further investigated
- Kemerer C and Slaughter S - Pittsburgh U. and CMU
  - Analysis of patterns
  - Methodological lessons and rigour in empirical studies of software evolution
- Shepperd M *et al* - Bournemouth U.
  - Dynamics of software maintenance

## Current view: summary

---

- In general, *E*-type software processes are **dynamic feedback systems**, observation of industrial process reveals many feedback loops
  - evidence, though qualitative, informal, anecdotal, is overwhelming
- The term *feedback* triggers different views according to people's backgrounds - this is may be useful but may as well lead to confusion!
  - work in *formal definitions* essential and to be tackled in the future
- Challenge is to refine the feedback hypothesis:
  - what type of **feedback** system?
  - how **disciplined** and **regular** is the **phenomenon**?
  - how **common** are the regularities?
  - what are the **underlying mechanisms**?
  - what **models** and **metrics** best capture the phenomenon?
  - what are the **implications**?
  - how to **exploit** it in practical settings?

## Current view: summary (cont.)

---

- Majority of lessons learnt by FEAST to date are qualitative, such as extension of phenomenology, refinement of laws, rules and guidelines
- Progress has also been made in quantitative system dynamic models
- In this regard, models show that **relative amount of resources** applied to **complexity control** can have a significant impact on the growth rate
- It suggests **anti-regressive work** as an important **control knob** of the evolution process - others may exist
- Decision to undertake complexity control in industrial processes may be **distributed** amongst many agents involved
- Topic addressed in Dr Kahen's presentation to follow

# Acknowledgements

---

- Many thanks are due to the industrial collaborators for having provided data on their systems
- Thanks are also due to Professors Dewayne Perry and Wlad Turski, SVFs, for their contributions. Plots in chart 9 emerged during interactions with Professor Perry
- This work has been funded by the UK EPSRC under grants numbers GR/K86008 Feast/1 1996-8, GR/M44101 Feast/2 1999-2001 and SVFs GR/L07437 6/96 - 6/97, GR/L96561 4/98 - 4/99 and GR/N02412