Fly me to the moon ... 

Professor Mike Hinchey
66 Years Ago ...
EDSAC

- 650 instructions per second.
- 1024 17-bit words of memory in mercury ultrasonic delay lines.
- Paper tape input and teleprinter output at 6 2/3 characters per second.
- 3000 valves, 12 kW power consumption, occupied a room 5m by 4m.
- "Operating system" occupied 31 words of read-only memory.
- Early use to solve problems in meteorology, genetics and X-ray crystallography.
Difference Engine
Errata, detected in Taylor’s Logarithms. London: 4to, 1972 [sic]

... 6 Kk Co-sine of 14.18.3 – 3398 – 3298

Nautical Almanac (1832)

... In the list of ERRATA detected in Taylor’s Logarithms, for cos. 4 18’ 3”
read cos. 14 18’2”.

Nautical Almanac (1833)

ERRATUM of the ERRATUM of the ERRATA of TAYLOR’S Logarithms.
For cos. 4 18’3”, read 14 18’ 3”.

Nautical Almanac (1836)
First Programmer

Augusta Ada King, Countess of Lovelace
Challenges for Software Engineering

- Increases in demand for greater, more complex functionality;
- Stricter (required and desirable) constraints on performance and reaction times;
- Attempts to increase productivity and reduce costs while constantly pushing requirements to the limit;
- Requirement of regular change and evolving systems.
Any intelligent fool can make things bigger and more complex …

It takes a touch of genius and a lot of courage to move in the opposite direction.

Albert Einstein
Evolving systems are software systems which exhibit change over time.

Software is supposed to change... otherwise it would be in the hardware!
Evolving Critical Systems

- have evolved from legacy code and legacy systems, or
- result from a combination of existing component-based systems, possibly over significant periods of time, or
- evolve as a result of a focused and intentional change in organization and architecture to exploit newer techniques believed to be beneficial;
- they require that the system adapt and evolve at run-time in order to react to changes in the environment or to meet necessary constraints on the system that were not previously satisfied and possibly not previously known.
Critical systems are systems where

- failure or malfunction will lead to significant negative consequences;
- these systems may have strict requirements for security and safety, to protect the user or others;
- alternatively, these systems may be critical to the organization’s mission, product base, profitability or competitive advantage.
ECS Research Agenda

An ECS Research Agenda addresses several core research topics in the evolving critical systems field.

The central research topic is building software that
(a) is highly reliable, and
(b) retains this reliability as it evolves, without incurring prohibitive costs.
Evolving Critical Systems

Tooling

Plan

Evolve

Assess

PEA+T
The diagram illustrates the MAPE (Monitor, Analyze, Plan, Execute) cycle in the context of software engineering. It shows the various components and processes involved in managing and optimizing software systems. The diagram includes the following sections:

- **Sensors**: Components for gathering data and monitoring the system, including Analysis Engines, Policy Validation, Policy Resolution, and Rules Engines.
- **Effectors**: Components for implementing changes and executing plans, including Policy Interpreter, Policy Transforms, Plans Generators, Workflow Engine, Service Dispatcher, Scheduler Engine, and Distribution Engine.
- **Knowledge**: Components for storing and managing system information, including Topology, Calendar, Recent Activity Log, Policy, Simple Correlators, Metric Managers, Filters, and Simple Correlators.

The MAPE cycle is represented by arrows connecting the different components, indicating the flow of information and actions from monitoring to analysis, planning, and execution.

Source: IBM, AC Blueprint 2003
Evolving Critical Systems

Evolving

- Software is meant to change, both at design/revision time and at run-time
- Lero’s research focuses on methods and tools for designing software that can be changed or that can change itself without degradation

Critical

- Much of today’s software is mission-, safety- or business-critical
- Lero is researching methods and tools to improve the integrity of and confidence levels in critical software

Systems

- Expertise in software engineering needs to be coupled with domain knowledge
- Lero has existing partnerships & expertise in medical devices, space, future cities and financial services
- We will continue to seek domain-related partners for collaborative research
Key Focus Areas

- A: Methods & Standards for High Integrity Systems
  - Lean, Agile & Global methods
  - Open Sourcing & Innovation
  - Standardised SW Development processes
  - Model-based approaches
  - Formal methods & safety use cases

- B: Adaptive & Autonomous Systems
  - Systems that learn & respond to their environments

- C: Software Performance
  - Large complex systems
  - Multicore embedded & massively parallel systems

- D: Security & Privacy
  - New approaches to security and privacy and the trade-offs between them
  - Digital forensics
Hub and Spoke Model

A: Methods & Standard for High Integrity Systems
B: Autonomous & Adaptive Systems
C: Software Performance
D: Security & Privacy

Software for Innovation & Business Processes
Software for Manufacturing Competitiveness
Software for Medical Devices & Healthcare
Software for Smart Cities & Buildings
Software for Financial Services
Software for Cloud Computing
Software for Analytics & Security
An ECS Scenario

• Space Exploration
  • Some of the most complex and expensive software applications to date.
  • High Levels of Autonomy.
  • Significant consequences for failure.
Swarm Technologies

• Inspired by swarms of bees and flocks of birds in nature;

• Many application areas:
  – drug discovery;
  – communication systems;
  – environmental monitoring;
  – exploration.
Coordinated swarms of smaller spacecraft will offer:

- More effective use of solar power;
- Access to areas where large craft could not go;
- Ability to perform more complex tasks;
- Greater accuracy and flexibility.
Three concept sub-missions:

1. Lander Amorphous Rover Antenna (LARA)
2. Saturn Autonomous Ring Array (SARA)
3. Prospecting Asteroid Mission (PAM)
Tet Walkers
ANTS Concept Mission - PAM
ECS Contributions

1. Formal Methods
2. Autonomic Computing
3. Software Product Lines
4. Automatic Code Generation
Swarm Formal Method Model and Outline

\[ \Phi = \{ \text{SendMessage, ReceiveMessage}, \{ \text{Reason, Process} \} \} \]

is a set of (partial) transition functions where each transition function maps

\[ \text{Memory} \times \text{Input} \rightarrow \text{Output} \times \text{Memory} \]

\[ \text{memory}' = (\text{Goals}', \text{Model}', \text{CommsTracking}) \]

[Communicating] Reasoning Deliberative (Leader) [Reasoning]

[Reasoning] SendMessage (Leader, Worker) [Communicating]

[Processing] SendMessage (Leader, Worker) [Communicating]

<table>
<thead>
<tr>
<th>Agent State</th>
<th>Actions leading to the agent state</th>
<th>f</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communicating</td>
<td>Identity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SendMessageWorker</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>SendMessageLeader</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>SendMessageError</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ReceiveMessageWorker</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ReceiveMessageLeader</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ReceiveMessageError</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Reasoning</td>
<td>Reasoning Deliberative</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Reasoning Reactive</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>Processing</td>
<td>Processing Sorting And Storage</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Processing Generation</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Processing Prediction</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Processing Diagnosis</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Processing Recovery</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Processing Remediation</td>
<td>17</td>
<td>2</td>
</tr>
</tbody>
</table>
ASSL Specification

```plaintext
AEIP {
    MESSAGES { ... }
    CHANNELS { ... }
    FUNCTIONS { ... }
    MANAGED_ELEMENTS {
        MANAGED_ELEMENT worker {
            INTERFACE_FUNCTION getDistanceToNearestObject { RETURNS { DECIMAL } }
        }
    }
} // AEIP

METRICS {
    METRIC distanceToNearestObject {
        METRIC_TYPE { RESOURCE }
        METRIC_SOURCE { AEIP.MANAGED_ELEMENTS.worker.getDistanceToNearestObject }
        DESCRIPTION { "measures the distance to the nearest space object" }
        MEASURE_UNIT { "KM" }
        VALUE { 100 }
        THRESHOLD_CLASS { DECIMAL [0.001 ~ ] }
    }
}
```
1. Formal Methods
2. Autonomic Computing
3. Software Product Lines
4. Automatic Code Generation
Inspiration from the human/mammalian autonomic nervous system.

**Fight or Flight**

- *sympathetic* (SyNS)

**Rest and Digest**

- *parasympathetic* (PaNS)
The Autonomic Environment

Autonomic Communications Channel

I am healthy

I am alive

Stay awake

ALice

sSleep

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*

AE

MC

AM

S*

S*

S*
1. Formal Methods
2. Autonomic Computing
3. Software Product Lines
4. Automatic Code Generation
Feature Model

If father present, the heir is:

- Mandatory
- Optional

Dependency:

- Only one of them
- At least one of them

Explore Universe

Move

Send Data Earth

Explore and Discover

Set Objective and Approach

Analyse

Self-Protection

Avoid Crashing

Avoid run a out of power

Protect from solar storms

Fight

Walk

Inform objective

Search new objective

Evaluate Interest

measuring

measure image

measure X-ray

Gas prop.

Use Sail to Orbit and flight

Snake

Amoeba

Rolling

Digital Camera

Optical Camera

Lero © 2010

THE IRISH SOFTWARE ENGINEERING RESEARCH CENTRE
ECS Contributions

1. Formal Methods
2. Autonomic Computing
3. Software Product Lines
4. Automatic Code Generation
Requirements expressed as scenarios

(Mathematical laws of concurrency)

(existing model extraction (reverse engineering) tools)

(existing code generating tools)

(reversed)
Current Status
Benefits of the Method

- Automation of entire development process;
- Significant increase in quality;
- Ability to do formal proof on properties of implementations;
- Ability to do formal proof of correctness;
- Automated means for requirements analysis;
- Guaranteed correspondence between requirements and their implementation as code.
Applications

- End-to-end automatic code generation of provably correct systems;
- Automatic reimplementation after any requirements change;
- Exploiting re-use across platforms;
- Reverse engineering legacy systems to a mathematically sound model;
- Analysis and documentation of existing systems (e.g., expert systems);
- Re-engineering of legacy systems to a provably correct new implementation.
Domains (to date)

• Agent Based Systems;
• Wireless Sensor Networks;
• ANTS;
### HRSM Procedures

**Start of Day 1a**

<table>
<thead>
<tr>
<th>0600:00</th>
<th>0600:15</th>
<th>0600:30</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WFC3 Tool Caddy (19:13h)</strong></td>
<td><strong>WFC3 Tool Caddy (19:13h)</strong></td>
<td><strong>WFC3 Tool Caddy (19:13h)</strong></td>
</tr>
<tr>
<td>Command EMI cover storage door open</td>
<td>Release Brakes (00:20h)</td>
<td>Move rear EM tool storage location (00:10h)</td>
</tr>
<tr>
<td>-</td>
<td>Set Brakes (00:10h)</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Release Brakes (00:00h)</strong></td>
<td><strong>Release WFC3 Tool Caddy (00:20h)</strong></td>
<td><strong>Release WFC3 Tool Caddy (00:20h)</strong></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- -

**WFC3-Installation**

<table>
<thead>
<tr>
<th>GAs</th>
<th>DR-2a</th>
<th>DR-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- -

**Assumptions**

- MTC brake Parking brake needs to be made after WFC3 installation (for thermal protection)
- 1 tool case
- 1 WFC3 Tool Caddy (ground Tool, Connector Tool, Stabilization Tool, remote in HST FRD), Socket Extension Tool (remote on ECU harness)
- 1 WFC3 Interface Plate, Harness Tool (remote on ECU harness)
- 1 WFC3 Interface Panel
- User protection required for HST KIT cavity, WFC3, and rear EM tool bays
- HST-S Ax position at 06:00 due to alignment - Y-Var

---

**Lero © 2010**

**THE IRISH SOFTWARE ENGINEERING RESEARCH CENTRE**
```
channel brakerelease, brakeset, stabilize, wftoolaquire : T;
GA = brakeset ! 0 -> GA;
DRone = brakeset ? x -> brakerelease ! 0 -> stabilize ! 0 -> brakeset ! 0 -> DRone;
DRTwo = brakeset ? x -> wftoolaquire ! 0 -> brakeset ! 0 -> DRTwo;
System = DRone || || || DRTwo || || || GA;
```
Transaction boltrelease = new Transaction(2);
Transaction brakeset = new Transaction(2);
Transaction stabilize = new Transaction(2);
Transaction wfc_toolaquire = new Transaction(2);
GA GA_init = new GA(brakeset);
DRone DRone_init = new DRone(boltrelease, brakeset, stabilize);
DRtwo DRtwo_init = new DRtwo(boltrelease, brakeset, wfc_toolaquire);
DRone_init.start();
DRtwo_init.start();
GA_init.start();

EXECUTING:
brakeset
brakeset
DEADLOCK DETECTED!
Conclusions

- Software must evolve.
- There is a tension between reliability, predictability and cost and this need for evolution.
- There is a need for an Evolving Critical Systems research effort.
- Lero and others are driving that effort.
Any problem in computer science can be solved with another layer of indirection.

But that usually will create another problem.

David Wheeler
Thank You

http://www.lero.ie/ecs/whitepaper