How never to learn from failure

Ulf Wiger, CTO Erlang Solutions Ltd
ACCU, Oxford 2011
About me

- 6 years working with Military C² and Disaster Response in Alaska
- 13 years as Software Architect at Ericsson
- 2 years at Erlang Solutions as CTO
To not learn from history

- “We learn from history that we do not learn from history”
  G.F.W. Hegel

- “Human history is a drama in which the stories stay the same, the scripts of those stories change slowly with evolving cultures, and the stage settings change all the time.”
  Fred Brooks, “Mythical Man-Month, Anniversary Ed.”
Programmers are Optimists

• “All programmers are optimists. Perhaps this modern sorcery especially attracts those who believe in happy endings and fairy godmothers”
  Fred Brooks, “Mythical Man-Month”

• Possible problem: Why learn from others’ mistakes, when it is so much fun to make your own?
Taxonomy of Programming

- Survey
- Select
- Enhance
- Ship
Taxonomy of Programming

Programming

Product Development
- Survey
- Select
- Enhance
- Ship

Study
- Experiment
- Research

© 2011 Erlang Solutions Ltd.
AC2SMAN - My formative years

- Alaskan Command & Control System
  Military Automated Network
  - Built in 4 months by a fighter pilot from Memphis, and some geeks
  - First ever “Overall Outstanding” rating given by NORAD 1989
Cool Lesson

• Running exercises with 50,000 soldiers
• Number of exercise controllers went down
  - from 900 without the system
  - to 30 with the system
• Later, during Desert Storm
  - The first ever fully simulated battle exercise
• Huge potential for reducing admin overhead
The C2 System Design Challenge

- Mission-critical
- Soft real-time
- Inconsistent data input
- Varying operating conditions
- Potentially global scale
- No single point of failure (40+ sites)
- Live, simulation and exercise – sometimes simultaneously
Rewind: The Feed Aggregation Problem

- Real-time subscription feed for tactical map workstations
- Messaging server was a big pile of C++ code
- Single point of failure
- Ran out of memory daily
- (Not due to programmer incompetence)
  - Purify was invented in 1990
I was Searching for a Solution

- Tons of approaches evaluated
  - CASE Tools, Client-Server middleware, AI middleware.

- Eventually landed in telecoms 1992
  - "Computers in Telecommunications" course at KTH, Stockholm
  - Teachers: B Däcker, R Virding

25-lines switchboard, Natal Province, South Africa 1897
Cross-switchboard calls required human interaction.
Erlang, Intuitively

http://video.google.com/videoplay?docid=-5830318882717959520#
Erlang, Intuitively

- One concurrent process for each naturally concurrent activity
**Erlang, Intuitively**

- One concurrent process for each naturally concurrent activity
Client-server in Erlang

1. Client monitors server
2. Client sends a request
3. (Blocks while waiting)
4. Server sends reply
Client-server in Erlang

Client monitors server

1. Client sends a request
2. Blocks while waiting
3. Server sends reply
4. call(S, Request, Timeout) -> Mref = monitor(process, S), S ! {call, Mref, Request}, awaiting_reply(Mref, Timeout).

awaiting_reply(Mref, Timeout) ->
    receive
        {Mref, Reply} -> Reply;
        {'DOWN', Mref, _, _, Reason} -> error(Reason)
    after Timeout ->
        error(timeout)
    end.
Ericsson – The Mythical Project

• I joined Ericsson 1996 to work with Erlang
• A very large project had just been canceled
  - A very public failure
• Distributed real-time, fault-tolerant complex systems in C++
Why did it crash?

- No obvious single culprit
  - Discussions about what went wrong dragged on for years
- Obviously, the size of the project was a problem
  - But why so large?
- OO mania, featuritis, hubris?

- My thought: failure to contain the problem
AXD301 – The Pickup Project

- 200 people put into one building
- Mission: Build a product within 2 years
  - “Something in the ATM domain with Telecom Characteristics”
- Erlang/OTP
Pragmatic thinking

- Shell shocked from previous project
- Fall back on what’s known to work
- Straight and simple took us pretty far
  - Design for what we need right now
  - Rework later if necessary
Some figures

- Up to $16 \times 16 = 256$ interconnected boards
- Up to 32 control plane processors
- Up to 500k simultaneous phone calls
- $> 99.999\%$ consistent uptime
  - (including maintenance & upgrades)

Pretty big and robust...
Failed evangelism

• We estimated 4x fewer lines of code, compared to similar systems in C++
  - Same fault density
  - Similar LOC/hr productivity
  - 4x higher quality and productivity

• Later, we reduced the fault density by another 2.5x, while adding functionality

• This had little impact on our political standing
Life in a Big Company

• Big possibilities, big frustrations
• Big companies are like small societies
  - Complete with politicians and all
• Size drives hierarchies
• Hierarchies need middle-men
• Middle-men mainly relay and aggregate information
  - How do you ensure that the “right” information is conveyed?
Flow of Information

• “An organization loses its intuition when the person who has the answer isn’t talking to the person who has the question”

  Tim Berners-Lee: “Weaving the Web”

• The key information flow is bottom-up—not top-down
Flow of Information

• “An organization loses its intuition when the person who has the answer isn’t talking to the person who has the question”

  Tim Berners-Lee: “Weaving the Web”

• The key information flow is bottom-up—not top-down
State Machine Hell

...but quite doable with Erlang

Legacy Phone Switch

Switch Emulator and Voice-over-ATM Controller

Extremely complex state machines
Aggregation/suppression of messages
Abstractions for non-determinism

• We were building complex distributed message-passing systems

• Key challenge: contain the non-determinism!

• Prevent explosion of the state-event matrix

• This had been identified by Ericsson already in the late 70s…
  - First experienced in the 60s
  - Identified and explained late 70s
  - Coloured EriPascal, Erlang, CHILL, et al
Some similar projects

- In one (mature) UML/C++ project, 10% of all bugs were related to unexpected order of events
- Inadequate methods for abstracting away accidental ordering
- Confusion as to whether OO abstractions actually helped this issue
Analogy: Tetris Management

- The age-old classic has coined a new time management method
- The idea: learn to keep the pile small
Tetris Management

• Used in a derogatory sense at a major software development project

• As in “reactive management without a plan”

• Basically, don’t let your project become a tetris game
A different kind of puzzle

• What if your puzzle resembles this?
• Would you attack this problem with a Tetris approach?

http://www.worldslargestpuzzle.com/hof-008.html
Event-handling Strategies

- Twist and place the next piece before it lands
- In cheat mode, you get to peek at the next piece
- Otherwise, hope for the best
- Search for a specific piece
- Put aside pieces that don’t fit
- Keep at it until fitting piece found
Event-handling in Software

- FIFO Run-to-completion event handling
- Not allowed to block
- Fine, as long as the pieces (messages) fit
- Blocking, selective receive
- Wait until the next *desired* message arrives
- Buffer unknown messages
(Movie tip)

- Memento (2000)
- Human FIFO Run-to-completion event handling
- Storing context for future reference

Attempt at Pedagogy

• Demo system used in Ericsson’s Introductory Erlang Course
  - Write a control program for a POTS subscriber loop

• Here: rewrite the control loop using different semantics
  - Selective message passing
  - Event dispatch

• A few minds converted...
The Simon P-J Test

• Invited to talk at WG2.8 at West Point 2004
• Topic: A plea to teach this pattern in college

• Tried the idea on a severely jetlagged Simon Peyton Jones (ICFP, Snowbird)
• He verified that it is not well known

• Not sure if it is in the curriculum now...
One Wonders...

- Why several projects, even when approached with this explanation, chose to try their own event-based C++ variant?
  - They all invariably fell into the same hole
- Problems not apparent in early prototypes
- The complexity sneaks up on you
  - As you start implementing the exception flows
  - As you add new protocols and features
  - As increased load changes timing aspects
Putt’s Law

• “Technology is dominated by two types of people:
  - those who understand what they do not manage and those who manage what they do not understand.”

• Corollary:
  - “Every technical hierarchy, in time, develops a competence inversion.”

  Archibald Putt: “Putt’s Law and the Successful Technocrat”

• If you’re out of your depth, being wrong is scary
Big organisation—Bell Curve

• Ideally, the few top designers/architects should drive concept and architecture work

• In practice, it tends to be driven by people closer to the middle
Division of Labour—Wissenwurst

- Knowledge is chopped into pieces
- Rather than grown continuously
- People can deal with enormous complexity if given time to digest
In Conclusion

• Many non-technical issues interfere with learning from our past mistakes

• Transparency in communication is vital

• Continuity of learning

• Dare to be wrong!