Chapter 1 – Introduction and Motivation

W. Morven Gentleman
Professor
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia
CANADA

Morven.Gentleman@dal.ca

Today's NATO military systems depend on large, complex software with the need to be built and deployed more rapidly and cheaper than traditional development methods can deliver. Moreover, because military commanders depend on these systems, they must be more predictable and trustworthy than traditional development methods can deliver for the available time and cost investments. However this requirement is not quite compatible with the traditional project-oriented view of software development, which is prevalent in today's military acquisition methods.

Today's systems are typically integrated from components. These components may themselves contain flaws, originating in specification, design or implementation errors, or in miscommunication between different teams involved in the development. More seriously, the integration process itself may be flawed, as when pre-existing components are used for purposes that their developers had not envisioned, and the integrators misunderstand the detailed behaviour of the components. This situation can arise in the NATO context, for instance, when coalitions are formed quickly, and complex systems must be integrated from subsystems supplied by different nations: the components might be flawed, they might be misused, and the integration might be inappropriately performed.

A different perspective comes from the recognition that components sometimes are systems in and of themselves, and that these systems may not lose their identity when integrated into a "System of Systems" but as well as being expected to fulfill their role in the System of Systems, may continue to retain their original purpose, with their independent management, independent operational needs, and independent evolution [Meier 1999][1]. Frictions, even conflicts, can obviously arise in such mixed circumstances. Interoperability failures between different national systems often are of this form.

Experience with interoperating commercial products, especially in the context of the Internet, indicates that robustness to fallible components and fallible integration can be achieved without centralized predictive coordination. Appropriate software architecture, redundancy in functional components, and enforcement of critical interface standards appear to be key elements of success. Improved registry and plug-and-play concepts can help automate integration and reduce configuration problems.

If we are to try to build infallible systems with fallible construction methods, there is a need to review the advances in software development in the commercial market. There is also a need to evaluate the requirements of military software development, bring forth lessons to be learned and to identify areas of research and draw projections especially for the procurement community.

RTO IST-064/RWS-11 was a cooperative international workshop specifically aimed at investigating how to usefully work with software that is believed to be faulty. The workshop was intended to be a truly interactive workshop rather than only a mini-conference of presented papers. That is, it was to be a working meeting to produce a specific deliverable: a report summarizing the state of the art. Attendees

[1] [Meier 1999] M. W. Meier, "Architecting Principles for Systems-of-Systems", Systems Engineering, 2:1, 1999.

were expected to participate fully in identifying the critical issues for successfully working with software believed to be faulty, determining community challenge problems, applications and case studies, and helping to set a research agenda for the area. The workshop was structured around a set of topics and issues identified by the participants. Prior to the workshop, all invited participants were asked to submit a position paper outlining their perspective, and to categorize their own position paper and their related work. The categorizations chosen helped to suggest session themes. The workshop consisted of several focused sessions, which set the stage for scheduled and impromptu presentations by participants and follow-up discussion.

The Technical Activity Proposal authorizing this workshop had defined possible topics to be covered:

1) Choices of software architecture for robustness.
2) Integration process and tools.
3) Critical interface standards.
4) Interoperability with complementary or related products.
5) Empirical behaviour investigation through testing.
6) Oracles to ascertain plausibility of results.
7) Scalability of fault recovery.
8) Autonomic systems, self-healing, dynamic re-configuration.
9) Coping with component evolution.
10) Aids to retraining users.
11) Scaffolding reuse.
12) Regression tests, integration tests, integrity testing, consistency testing.
13) Project metric tracking.
14) Implications for cultural change.
15) Architectures for defect tolerance.
16) CORBA reliability.

In planning the workshop, the overlap with software fault tolerance was noted, so topics for discussion were suggested where there was a difference today from situations that had been considered in the past with respect to fault tolerance:

1) Use of pre-existing components.
2) Excess computing capacity that can be used:
    a) For training and simulation to ensure proper functioning when needed;
    b) For audit routines to monitor integrity of run-time and persistent data structures; and
    c) For self-management (AI techniques).
3) Decentralized operation and control.
4) Conflicting, unknowable, diverse requirements.
5) Continuous evolution and deployment.
6) Heterogeneous, inconsistent, changing elements.
7) Indistinct people/system boundary.

8) Failures considered normal.
9) New forms of acquisition and policy.
10) Use of signatures to identify potential upcoming problems:
    a) Correlation only yet causality may be needed to be able to take corrective actions.
11) Depth of exposure of commanders to computers.
12) Display of uncertainty.
13) Size and level of expertise of development teams.
14) Increasing use of concurrency (and lack of exception handling mechanisms for concurrency).
15) Voting mechanisms may not work (properly) because:
    a) Answers not always being unique or uniquely represented;
    b) Answers that cannot be compared automatically; or
    c) Answers that cannot guarantee to be compared within certain time limits.
16) Classical engineering use linear models:
    a) What does linearity and continuity mean for software systems?
    b) Can we build linear models for software?
    c) If not, can we handle non-linear models?
17) Agent-based architectures and the publish/subscribe mechanism decouple components. Does this make the understanding of the system as a whole not more difficult?

Some new opportunities for research were also suggested:

1) Revise/reopen previous research (e.g. strongly guarded data structures).
2) New ways of doing distributed computing.
3) Information synthesis/fusion (area also covered in grid computing field).
4) Taking sociological issues into account.

Unfortunately, it proved impossible to arrange participation in the workshop by any representative of one obvious research group relevant to the topic, the Stanford-UC Berkeley initiative on Recovery Oriented Computing. This group has explicitly taken the position that failures are inevitable, if only because of inappropriate input by humans such as system operators as well as end-users. Consequently, their approach has been that rather than concentrating on avoiding failure, attention should be focused on making recovery from failure faster and more reliable. Perhaps surprisingly, there is as yet very little other work from this perspective. In the absence of a representative of this group, the workshop discussions deliberately had to attempt to anticipate the reactions this group would have.

The original intent was that the report would include:

1) Summary of workshop results;
2) Position papers;
3) An identified research agenda;
4) Community challenge problems; and
5) Plans for future cooperative activities.