Finding Better Strategies for Software Intensive Projects: The Efficiency Vs. Robustness Dilemma

Ricardo Peculis, CSC, SEEC-UniSA
Derek Rogers, SEEC-UniSA
Peter Campbell, SEEC-UniSA
Topics

- Overview
- Software Intensive Projects
- Social and Technical Systems
- The Development of Linux
- Strategies for Complex Projects
- Summary
- Conclusions
- Questions
Overview

- Software intensive projects continue to fail
  - cost overruns,
  - schedule delays,
  - and reduced quality.

- The causes of failure can be attributed to
  - Complexity of today’s technical systems, and
  - Even more complex social system that are required to engineer them.

- The greater the complexity, the higher is the chance that the system will encounter situations that have not been considered.
Overview

- Complex systems need diversity to be robust and adapt to unplanned conditions.

- Projects aim to be effective and efficient.

- Efficiency Vs Robustness Dilemma.

- The development of Linux.

- Sacrificing efficiency for robustness can be a sensible approach for strategic software intensive projects.
Software Intensive Projects

- Apply the process of engineering to develop products highly dependent on software.
- Systems of strategic importance impose strong schedule requirements.
- Safety critical systems impose strong quality requirements.
- Cost is always a strong requirement.
  - Software developed by business organisations is meant to be profitable.
Software Intensive Projects

- Software developed under altruistic initiatives like Open Source and Free Software
  - Good Quality
  - Flexible
  - Low Cost
  - No business constraints
  - Virtually unlimited resources
  - Harnessing the power of software developers that give their time for “free”.

SETE 2007 Sydney
Software intensive products are complex technical systems
- Interconnected technical components (documents, hardware, software).

Require a complex social system to engineer them
- People associated in teams and organisations.
- Bounded by hierarchy, relationships, processes and contracts.
To be effective, a system has to be at least as complex as the task it performs (Bar-Yam, 2003 from the Law of Requisite Variety, Ashby, 1957).

- Technical Systems have to be at least as complex as the task they are meant to execute and the need they should satisfy.
- Social Systems have to be at least as complex as the technical systems they are tasked to engineer.
Complex systems need diversity (variety) to be robust.
- Adapt to unexpected conditions and continuing achieving their goals.

Diversity increases robustness of technical and social systems.
- Diversity in technical systems:
  - Features that are not needed immediately, but may be needed in the future;
  - Spare computational capacity to allow expansions.
- Diversity in social systems:
  - Human resources that provide knowledge, expertise and ideas.
Social and Technical Systems

- Effectiveness
  - How well the system achieves its goals.

- Efficiency
  - How much waste the system produces to be effective.

- Each system has a nominal efficiency to achieve maximum effectiveness (Hitchins, 2003).
  - A system can be highly efficient and ineffective.
Social and Technical Systems

- Ideal system
  - 100% effective.
  - 100% efficient.

- Real systems sacrifice efficiency to be effective.

- Technical Systems are not ideal
  - Spare computational resources (e.g. 30% spare capacity).

- Social Systems that engineer Technical Systems are not ideal.
  - Planned “enabling tasks” (e.g. reviews, testing).
  - Planned and unplanned “rework”.

SETE 2007 Sydney
Social and Technical Systems

- Software intensive projects aim to be effective and efficient.
  - Effective: delivers a product that satisfies a need (meets customer’s expectations).
  - Efficient: reduces costs, increases value for money and profit (meets business expectations).

- As complexity increases, technical and social systems need to be robust to cope with situations that have not been anticipated.
  - Robustness requires diversity, increases costs, reduces efficiency.
Diversity provides capacity in excess that may or may not be required.

Spare capacity may be the difference between success of failure.

Increase in diversity is not always seen as a benefit, and often perceived as a detriment.

Efficiency Vs Robustness Dilemma

- Risk of investing into diversity to make the system more robust.
The Development of Linux

- The development of Linux
  - Software development as a CAS.
  - No commercial interests.
  - No cost or schedule constraints.
  - Produced a good and robust product.
  - Linux community is also a robust social system.
  - Seems to be inefficient.
The Development of Linux

- The development of Linux as a CAS
  - Developers interact, adapt and execute their tasks without central control.

- Three key processes that provide the basis for adaptation (Axelrod & Cohen, 2000):
  - Variety
    - Linux Community (motivation and knowledge)
  - Interaction
    - Etiquette of the “open source” community.
  - Selection
    - Emergent QA processes.
    - Intervention by a select team (decides what will be included in the official release).
The Development of Linux

- The real costs of Linux(*)
  - Linux Kernel 2.6 is estimated to be between US$50 and US$176 million.
  - Whole Linux distribution (Kernel, API Library and Core OS Utilities) is estimated to be in the order of US$1 billion.

(*) Applied default COCOMO model for 4,287.45 KSLOC (Wheeler, 2006).
The Development of Linux

- The approach used to estimate the cost of Linux is questionable

  - Highly motivated programmers working part-time and for free is unlikely to have the same productivity as the industry standard.

  - It was not taken into account that the development of Linux is highly redundant.

  - Only a portion of the code developed is selected to be included in the official release.
The Development of Linux

- The estimation shows that the productivity of the Development of Linux is at least 3.5 times less than the average industry standard.

- Taking into account the code that was developed and not used, the efficiency of the development of Linux could be even lower.

- The Linux community seems to be a robust, effective and an inefficient social system, which confirms that systems that are robust trade efficiency for robustness (Hitchins, 2003).
Strategies for Complex Projects

- **Evolutionary Engineering** (Bar-Yam, 2003)
  - Adopts principles extracted from complexity theory.
  - Incremental iterative changes, redundancy and parallelism.
  - Fosters diversity.
  - Expected to be more effective and less efficient than traditional engineering methods.

- **Open Source**
  - Effective and robust CAS.
  - Variety, redundancy and parallelism.
  - Effective interaction and selection processes.
  - Low efficiency caused by high diversity and redundancy.
Summary

- Each system has a nominal efficiency to achieve maximum effectiveness.
- Complex systems need to be robust to cope with unplanned situations.
- The social system has to be at least as complex as the technical system it is tasked to engineer.
- Complex systems need to be robust to cope with unplanned situations.
- Robustness require diversity, which reduces efficiency and creates the Efficiency Vs Robustness Dilemma.
- The development of Linux shown to be a robust, effective and apparently inefficient.
Conclusion

- Failure of software projects can be associated with the complexity of technical and social systems and our ability to deal with them.

- As complex systems need to be robust, trading efficiency for robustness can to be a sensible strategy to assure success of strategic projects.

- Effectiveness and Efficiency can be improved by changing the system.
  - Better socio-organisational design,
  - Better processes,
  - Better prepared professionals.
Questions?