Delivering mission-critical systems

BCS Wiltshire, 8th Nov. 2018, Swindon

Leading systems engineering projects in extremely demanding environments

Colin Butcher CEng FBCS CITP
Technical director, XDelta Limited
www.xdelta.co.uk

xdelta

XDelta – what we do

- Lead mission-critical systems projects:
  - Technical leadership
  - Strategic planning
  - Project direction
- Minimise risk of disruption to business:
  - Design for change while in continuous operation
  - Prepare in advance for ease of transition
- Ensure long term success through skills transfer
  - Mentoring and teaching

Agenda

- Introduction
- Systems engineering – design principles
- Availability and performance
- Risk and failure analysis
- Testing, transition and operation
- Project planning and management
- Leadership and people
- Summary

What are mission-critical systems ?

Systems that are relied on to get something done, without data loss or corruption. They need to deliver continuous operation without disruption to service. Usually there is some kind of severe impact for systems failure.

This requires careful design and planning to be able to make changes while the systems are running. For example, hardware maintenance or replacement, operating system updates, application software changes and even moving data centres.

Terminology

- Availability:
  - Probability of system being available for use when needed
  - Function of MTBF (reliability) and MTTR (repair time)
- Disaster tolerance:
  - Surviving major site outages without loss of service
- High availability:
  - Surviving failures at a site without loss of service
- Disaster recovery:
  - Restarting systems after loss of service (however brief), typically from another location (DR site)

Mission critical system characteristics

- Mission critical systems need to be able to:
  - Survive failures (resilience and failover)
  - Survive changes (adapt and evolve)
  - Survive people (simplify and automate)
  - Never corrupt or lose critical data (data integrity)
- What is the “operational window” ?
- Safety-critical systems also have to be “fail-safe”
- Real-time systems also have precise performance targets

Mission critical system examples

- Telephony and communications
- Power generation and distribution
- Air traffic monitoring and control
- Railway signalling
- Traffic flow control (cities, motorways, etc.)
- Healthcare (logistics, blood, x-ray, administration, etc.)
- Stock exchanges and finance data
- On-line services (email, purchasing, membership, tax, etc.)

Systems are now pervasive, only the scale and severity change.
Everyone expects 24x7 operation.

Some important challenges

- Protect the data: no loss, no corruption
- Reduce the probability of failure
- Minimise the extent of a failure
- Understand how and why systems fail
- Understand what failure looks like
- Keeping everyone involved ready to respond
- Systems have to keep working, no maintenance windows
- Keeping up to date

“Survivability test”

Cause of Outage	Planned (Maintenance)	Unplanned (Failure)
Hardware	?	?
Operating System	?	?
Network	?	?
Application Software	?	?
Data	?	?
Environment	?	?
People	?	?

xdelta

System failures – how are you affected ?

- How do you cater for failures ?
  - What level of outage is acceptable ?
  - What level of data loss is acceptable ?
- How do you set expectations of what's possible ?

The closer you get to 100% uptime the harder and more expensive it is.

Systems engineering

It’s a multi-disciplinary and holistic approach to creating something to meet a specific purpose.

From the NASA Systems Engineering Handbook, June 1995:

“[Systems engineering] is a field that draws from many engineering disciplines and other intellectual domains. The boundaries are not always clear, and there are many interesting intellectual offshoots.”

Your “duty of care”

- Be utterly clear about your “duty of care”
- Helps you exercise judgement and make clearer decisions

- Need to understand what you’re designing for, why you’re involved and the context it will operate in.

- Need to understand the possibilities and limitations of the available technology.

- Need to understand the level of expertise available to look after it all when it’s in service

Stages in the life and death of a system

- Requirements
- Design
- Implementation
- Test
- Regulatory approval
- Transition into service
- Operation, support and change management
- End-of-life and transition out of service

It is by no means a smooth and linear progression from stage to stage.

The design process

- Understand the requirements and your “duty of care”
- Start with the “ideal design”
- Understand any constraints you have to deal with
- Think ahead to minimise problems later
- Build the whole system “on paper”
- Have a well-structured overall architecture
- Understand the details, complexities and interactions
- Remain flexible and adapt to change

Requirements – it all starts here

- Clarity is essential
- Understand the problems you're trying to solve
- What are the acceptance criteria ?
- Minimise complexity
- Do not over-specify!
  - 99.999% = 5 minutes per year (24x365)
  - 99.99% = 52 minutes per year (24x365)
  - 99.9% = 8 hours 46 minutes per year (24x365)

Be prepared to reconsider if you don't like the price or the timescales – but know what compromises you can safely make.

Design decisions

- All design decisions are compromises and require you to exercise judgement
  - Big decisions which have long-term implications and constraints
  - Small decisions which seem big at the time
  - There will be requirements and constraints you don't yet understand or know about
- Make careful assumptions as needed to get started
- Establish meaningful naming conventions
- Document your design and decisions

Problem solving concepts - TRIZ

	Before	Now	After
Environment			
System			
Component			

See www.triz.co.uk (and several others) for a lot more information!

“теория решения изобретательских задач”
Teoriya Resheniya Izobretatelskikh Zadach”
“Theory of inventive problem solving”

xdelta

Availability, performance and security

A system that fails to meet its performance or security requirements is a system that's not working properly, so it becomes unavailable.

Performance related failures are often transient and exceedingly difficult to fully understand and resolve.

Frequent security attacks can affect performance. Increasing security can increase system overheads, affecting performance.

Systems have to have sufficient capacity and performance to deal with the workload in an acceptable period of time under normal, failure and recovery conditions.

What is “fast” and “slow” ?

- Bandwidth – determines throughput
  - It’s not just “speed”, it’s “units of stuff per second”
- Latency – determines response time
  - Determines how much data is in transit
  - “data in transit” is at risk if there is a failure
- Jitter - determines predictability of response, or “diff latency” (variation of latency with respect to time)
  - Important for establishing timeout values
  - Latency fluctuations will cause system failures under peak load

Current technology trends - parallelism

- Move from low core count, high clock rate processors to high core count, low clock rate processors
- Implicit assumption is that parallelism can be achieved
- How does our workload break down into parallel streams of execution ?
- Serialisation, synchronisation and intercommunication

Capacity and scalability

- Contention and saturation – running out of capacity
  - Queuing theory
  - What else are we sharing our capacity with ?
- Increasing the capacity of the overall system:
  - “Scale up” or “vertical scaling” – adding resources to a machine or buying a bigger machine (CPU count, memory, I/O adapters, etc.)
  - “Scale out” or “horizontal scaling” - adding more machines

Designing for performance

- Size systems to cope with peaks in workload
- Eliminate wait states as far as possible
- Think parallel, not sequential

- A faster machine just waits more quickly
- Don’t make it go faster, stop it going slower
- The fastest code is the code you don’t execute
- The fastest IO is the IO you don’t do
- The “idle loop” is anything but idle

Designing and writing very good code requires very good programmers.

Designing for availability

- Which parts of the system are mission-critical ?
- Which parts of the system are safety-critical ?
- How can we detect failures ?
- What kind of failure do we prefer ?
- What state transitions occur ?

- Segment the systems to limit failure propagation
- Build in the ability to make changes when in service
- Protect the data

- Build “proof of concept” systems and simulators

Designing for security

- What regulatory framework applies ?
- What monitoring and logging is necessary ?
- How can we detect security failures ?

- Understand authentication and access control

- Segment the systems to limit extent of breaches
- Protect the data “at rest” and “in transit”

- Have realistic and effective security policies
- Develop, test and enforce them

Risk and failure analysis

Risk is a combination of probability of occurrence and worst-case effects for a given failure scenario.

Allow for failure – success is only one of many possible outcomes.

The risk continuum

- What is the probability of a situation occurring ?
- What is the impact if that situation occurs ?
- What are the long-term consequences ?

- Most projects handle medium risk well enough
- Many projects over-specify to cater for low risk issues
- Some projects under-specify and fail to cater for high risk issues

- We need to identify critical components / people
- We need to identify critical stages during the project

Risk identification and assessment

- Can we identify specific scenarios of interest ?
- Can we test all the conditions ?
- What happens to our data ?

- How can we start to identify what the risks might be ?
- How can we look for single points of failure ?
- How can we look for modes of failure ?
- How can we analyse how failures will ripple through ?

Testing, transition and operation

A mission-critical system hardly ever fails, so we need the people responsible for its operation to have a good understanding and ‘feel’ for the way it works.

Most failures result from a combination of several things and can thus be complex and difficult to understand and resolve.

Failure and recovery

- We need to know how the system behaves
- We need to know the ‘warning signs’ of incipient failure
- We need to know how to return the system to its normal operational state without data loss or data corruption

- We need to regularly rehearse and test our procedures and plans to ensure that we stay current

- We must have a representative offline test environment

Testing

- Understand the requirements and acceptance criteria
- How do we generate a typical workload ?
- How do we generate representative data sets ?

- Test under normal, failure and recovery conditions
- Don’t just confirm that the system behaves as expected
- Must test for scalability as well as functionality

“Everyone has a test environment. Some are lucky enough to have a separate environment for production.”

Transition into service / out of service

- Minimise risk of data loss
- Minimise risk of loss of service
- Migrate user connectivity
- Migrate live data + historic data

- How can we split transition into manageable steps ?
- Is anything a one-way step ?
- How much can we do in advance ?
- How could we revert to the original system ?

Finding and fixing problems

- How can we spot a problem early on, eg: data corruption ?
- What evidence can we look at ?
- Can we recreate the problem in a test environment ?

- Time synchronisation across the whole system is essential

- Continual monitoring and event logging is essential

- Knowledge of the whole system is essential

Project planning and management

Some useful adages:

“Proper Planning and Preparation Prevents Piss Poor Performance”

“Time spent in reconnaissance is never wasted”

“No battle plan survives first contact with the enemy”

Key issues

Most of what we deliver has no physical reality. This kind of work requires good conceptual skills and the ability to communicate ideas clearly. Everyone involved needs to have a sufficiently similar conceptual model, which relies on good communication.

Most projects go wrong through mis-match of expectations and lack of understanding.

Team size is limited by the necessary level of communication and the skills of those involved.

Planning and implementation

"More software projects have gone awry for lack of calendar time than all other causes combined."

"The mythical man-month" – Brooks

- Estimating and planning are key
- You cannot know everything up front
- Make effective assumptions to get started
- Beware assuming that everything will go well
- Cumulative discrepancies add up very quickly
- How will you monitor progress ?
- Checklists are essential, especially under pressure

Management

- Concentrate on quality of information and decision making
- Be thoughtful, not reactive - do not rush to respond
- Regular briefings, in person, phones off, no e-mail
- Need people to be committed
- Never have one person working on their own
- Find good people, guide them and trust them
- Good administrative support lets people focus on their work
- Good management enables people to get things done

Leadership and people

“Never tell people how to do things. Tell them what to do and they will surprise you with their ingenuity.”

General George Patton

“The best executive is the one who has sense enough to pick good people to do what he wants done, and self-restraint enough to keep from meddling with them while they do it.”

Theodore Roosevelt

People and behaviours

- Build groups of excellent people who work well together
- Choose people who are willing to share information and help each other
- Confidence is good; arrogance is bad
- Give them the support and help they need so that they aren’t distracted by trivia
- Much harder with widely distributed teams, outsourced providers and different cultures
- We’re all in this together!

Design, leadership and management

- Design and implementation:
  - Have clear objectives. Think ahead as far as you can. Have a well-structured systems architecture. Understand the constraints. Focus on the core functions. Implement them as well as is possible.

- Project leadership:
  - Ensure that everyone involved maintains a consistent understanding of the project. Plan ahead as best you can.

- Budget and Schedule:
  - They have to be appropriate for the problems you're trying to deal with. Don't set them first!

Procurement and responsibility

- Procurement – do enough work up front:
  - Understand what is technically feasible
  - Understand what is strictly necessary
  - Clearly establish the scope
  - Define clear objectives
  - Define acceptance criteria

- Avoid split responsibility and be absolutely clear where the “duty of care” lies

Key success factors

- Small team of the best people you can find
- Collaboration and willingness to share information
- Build a “proof of concept” early on and learn from it
- Minimise complexity
- Well-architected system:
  - Structured design
  - Clearly defined interfaces
- Document your decisions (what, how, why)
- Make life as easy as you can for those who come after you

Delivering mission-critical systems

BCS Wiltshire, 8th Nov. 2018, Swindon

Thank you for your participation

Colin Butcher CEng FBCS CITP
Technical director, XDelta Limited
www.xdelta.co.uk