



## VBUG Conference 2006

18<sup>th</sup> October 2006

Microsoft Campus, Reading, UK

---

### Designing A Mobile System For Maintainability

Martin Lixenfeld

[martin.lixenfeld@atosorigin.com](mailto:martin.lixenfeld@atosorigin.com)

## Introduction



- Today's new technology is tomorrow's legacy
- Mobile Technology is maturing
- Mobile systems need to be maintained long-term
- How can we design maintainability into any system ?
- How can we design maintainability into a mobile system ?

We will look at

- Functional vs non-functional requirements
- Desirable architectural features
- Best practice considerations
- A brief look at Patterns, Practices and Software Factories
- The software development process

## Mobile Technology



- Has grown up with Microsoft's Pocket PC 2003 and Windows Mobile 5
- Still some way off the maturity of Windows Forms
- But: Is now mature enough to be a serious business tool
- Mobile systems are being developed alongside Windows and Web applications
- Or systems are designed to incorporate mobile functionality
- Mobile systems are moving under support and need to be maintained long-term

## Outline



- **Systems Maintenance**
  - A closer look at maintenance work
- Designing Any System For Maintainability
- Maintainability
- Sidebar: Patterns, Practices and Software Factories
- Systems Architecture
- Architectural Differences Between LAN Based and Mobile Applications
- Development Process
- Summary

## Maintenance Work



- Most developers don't like it
- Most will have to look after an existing system at some time
- Somebody will have to maintain the systems we write today
- How to design maintainability into any system from the start ?
- How to design maintainability into a mobile system ?

## Typical Maintenance Scenario



- User raises an incident through the helpdesk or first line support
- First line support record the problem and assign it to second / third line support
- Developer picks up the incident and investigates
- Recreates problem
- Identifies and codes solution
- Issues Fix

## Problem Determination and Fix



- 80/20 rule – time required for replication vs fix
- If data read issue, maybe re-run in production system
- If data write issue, replicate in test system, using state information
  - files
  - database records
  - user log-ins (relevant for permissions)
- In LAN based system one would typically have access to the data and/or the state that caused the problem
- If working inside the company where the system is run, there may be face-to-face access to users

## Delivery of Code Fix




- Once the situation can be clearly replicated, all the developer has to do is
  - Find the offending line of code
  - Fix it
  - Test it / regression test it
  - Release a new version of the software
- In LAN based system possibly simply deliver across LAN
- In mobile system, think about delivery (more about this later)

## Outline



- Systems Maintenance
  - A closer look at maintenance work
- **Designing Any System For Maintainability**
- Maintainability
- Sidebar: Patterns, Practices and Software Factories
- Systems Architecture
- Architectural Differences Between LAN Based and Mobile Applications
- Development Process
- Summary



## Functional vs Non-Functional Requirements

Functional

- The WHATs of the system
- The features the customer pays for and expects to see

Non-Functional

- The HOWs of the system
- The more implicit features the user cannot see
- Should not emerge by accident
- Can be divided into constraints (must not...) and qualities (must...)

[www.vbug.net](http://www.vbug.net) VBUG Conference 2006

## Non-Functional Requirements



### Development-time vs Run-time qualities

#### Development-time

- Localisability
- Modifiability
- Extensibility
- Evolvability
- Reusability

#### Run-time

- Usability (ease-of-use, learnability, memorability, efficiency)
- Configurability
- Reliability
- Availability & Fault-tolerance
- Scalability
- Maintainability (incl testability)

## Outline



- Systems Maintenance
  - A closer look at maintenance work
- Designing Any System For Maintainability
- **Maintainability**
- Sidebar: Patterns, Practices and Software Factories
- Systems Architecture
- Architectural Differences Between LAN Based and Mobile Applications
- Development Process
- Summary

## Maintainability (Definition)



‘The ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment.’  
[IEEE 90]

## Maintainability cont'd



Maintainability is a **design** parameter and includes

- Servicability
- Reliability
- Supportability incl testability, i.e. we can easily determine
  - If something is broken
  - When it is fixed

## Maintainability cont'd



Maintainability will be supported through

- Consistent Architecture
- Good Documentation
- Consistent Error Handling
- Logging
- State Recording
- General Software Craftmanship

## Consistent Architecture



A consistent and easy to understand architecture

- Every problem is solved once
- Solutions are re-used
- Similar problems have similar solutions
- Once a solution to a particular problem has been understood, this understanding can be used for other parts of the system

## Consistent Architecture cont'd



IT systems generally collect, process, display and store data, why are we all writing similar systems ?

- Industry standard, tried and tested solutions to well-known problems exist
- Patterns, Practices, Application Blocks, Software Factories (more about this later)

## Documentation



- Document conventions
- Put every function into context
- The reasoning behind decisions
- At a high and a low level, i.e. comments in code !
- A developer can mostly figure out what code does, but
- The questions are: Why is this the way it is, why here, why now ?

## Error Handling



- Should be done in a consistent way
- Errors should be self-describing
- Too often one error is the result of some other error caused by some other error and the message received is meaningless
- Errors should be logged

## Logging



There are several approaches

- Only log errors, as long as this is documented and consistently applied, it is helpful – no log entry, no problem
- However, we might want to know what has been going on in the application before the error occurred
- Therefore, maybe log things that have gone right, if they are of significance
- But, how to know if an event is significant ?
- Log everything in case it might be useful ?
- Use 'Data driven' logging, i.e. make it dynamically configurable what is being logged
- Use Logging Application Block

## Serialising state



- Serialise state regularly
- Serialise state at intermittent points in long processes
- Break long processes into sub-tasks and write intermediate results out to temporary storage for later examination
- Make that state available for examination
- This allows to find out how far a long process got before going wrong

## Software Craftmanship



- 'Write programs for people first, computers second' (McConnell: Code Complete 2, 2004)
- Write self-documenting code
  - Naming conventions
  - Formatting conventions
  - Location conventions



---

'But this is all Common Sense'

## Common Sense



... is the least common of the senses

## What are we maintaining ?



- User authentication code - ✓
- Infrastructure / Navigation code - ✓
- Database access code - ✓
- Exception management code - ✓
- Logging code - ✓
- Caching code - ✓
- Configuration code - ✓
- Business objects - ✓
- Other custom entities - ✓
- Custom validation code - ✓

## Outline



- Systems Maintenance
  - A closer look at maintenance work
- Designing Any System For Maintainability
- Maintainability
- **Sidebar: Patterns, Practices and Software Factories**
- Systems Architecture
- Architectural Differences Between LAN Based and Mobile Applications
- Development Process
- Summary

# Patterns



## Patterns in Architecture

- Come from architecture (the bricks and mortar kind)
- Christopher Alexander: A Pattern Language: Towns/Buildings/Construction, 1977
- Concerned with building design and town planning
- Definition: 'Each pattern describes a problem which occurs over and over in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice'
- Concerns itself with 'Design Re-use'
- Well used in industry (e.g. car design)
- Still hard to do in software

## Patterns cont'd



### Patterns in Software

- 'Gang of 4' wrote book applying this to software - Gamma, Helm, Johnson, Vlissides: Design Patterns, 1995
- Initial implementations in Smalltalk and C++
- At the class / object level
- Slowly but surely the ideas got picked up and disseminated elsewhere
- Microsoft and others have expanded this

### Patterns and Practices

- Is Microsoft's effort
- Microsoft and others have produced downloadable Application Blocks

## Application Blocks



Enterprise Library January 2006 contains the following general purpose application blocks:

- Caching Application Block
- Cryptography Application Block
- Data Access Application Block
- Exception Handling Application Block
- Logging Application Block
- Security Application Block

Enterprise Library also includes a set of core functions, including configuration, instrumentation, and object builder services. These functions are used by all other application blocks.

## Software Factories



### Microsoft's Definition:

- 'Factories use automated guidance packages, including domain specific languages and recipes, to automate development activities partially or in full. The factory schema defines the recommended approach to building the specified type of applications, expressed in terms specific to that type and not in general terms'
- Can be defined across application or business domains
- Several now available for different application domains

## Software Factories cont'd



### Mobile Client Software Factory

- Encapsulates patterns and guidance packages specific to the mobile application domain, including the following application blocks specific to the mobile scenario
  - Composite UI
  - Connection Monitor
  - Disconnected Agent
  - Orientation Aware
  - etc

## Working at increasingly higher levels of abstraction



- Pattern – individual, small-scale problem and solution pair: MVC / Facade
- Application Block – may use several patterns to solve higher level problem: Data Access
- Guidance Automation – automates usage of larger patterns with templates and plug-ins into development tools
- Software Factories – Higher level abstraction again:  
Relevant to a specific application or business domain:  
Mobile Client, On-line Banking
- Allows assembly of complete application architectures

## What are we maintaining ?



- User authentication code - x
- Infrastructure / Navigation code - x
- Database access code - x
- Exception management code - x
- Logging code - x
- Caching code – x
- Configuration code - x
- Business objects - √
- Other custom entities - √
- Custom validation code - √

## Outline



- **Systems Maintenance**
  - A closer look at maintenance work
- **Designing Any System For Maintainability**
- **Maintainability**
- **Sidebar: Patterns, Practices and Software Factories**
- **Systems Architecture**
- **Architectural Differences Between LAN Based and Mobile Applications**
- **Development Process**
- **Summary**

## Systems Architecture



Architecting a system is a difficult task, because

- The most fundamental decisions need to be taken at the position of least knowledge about the system, therefore
- **Design from the intended outcome and work backwards**
- **This is not normally done as we design from requirements forward**
- Non-functional requirements even more than functional ones need to be designed into the system from the start

## Non-functional requirements



### Extensibility

- If not designed into the system, adding functionality at a later stage will be difficult if not impossible

### Maintainability - first design

- Error handling
- Logging infrastructure
- How the GUI represents the state of the system to the user when things go well and when things go badly
- Then design the functional features on the back of this skeleton

It may well be impossible to retro-fit this later !

## Outline



- Systems Maintenance
  - A closer look at maintenance work
- Designing Any System For Maintainability
- Maintainability
- Sidebar: Patterns, Practices and Software Factories
- Systems Architecture
- **Architectural Differences Between LAN Based and Mobile Applications**
- Development Process
- Summary

## LAN-based System



In a LAN based system, a number of assumptions are made:

- Machines have a large processor / RAM and storage
- Resources on the network are generally available
- Communication can be synchronous (e.g. connecting to a database)
- Connections can be made to a number of resources in different places
- Connections can be bi-directionally, i.e. my client can also play server
- Most processing can be considered 'on-line' as we are connected all the time
- New software or patches can be delivered across the LAN

## Mobile System



- Mobile systems are designed for a connected or dis-connected scenario
- Delivery mechanism of data is 'Store And Forward', i.e. if captured data can not currently be delivered, it will be stored and later re-sent
- Typical limitations
  - Machines are not connected through a LAN
  - Communicate asynchronously with a server via GPRS
  - Connect through one or a number of (Web Service) end points
  - Will only periodically send data
  - Communication is uni-directional
- Software / patch delivery is through the air
- This needs to be designed carefully

## Mobile Device Limitations



Mobile computers have a number of limitations compared to a PC such as:

- Resource constrained
- File system of limited size
- Limited amounts of memory
- Limited amounts of screen space
- Battery driven and can only be operated for a limited amount of time before requiring reconnection to mains power
- Consider serialisation of in-memory state on power-down

## Mobile System Maintainability - Additional Considerations

- User feedback
- Logging
- Housekeeping of logs
- Secure / locked down devices

## User Feedback



- In LAN based system a lot of 'system health' information can be hidden from the user
- In a mobile system **the user needs to be in on the act**
- Developer can only get at the information through the user

## User Feedback – Scenario:



- Mobile user hits a problem and phones in for help: s/he should have access to the following information:
- Version of software (there could be several versions in operation)
- Is he in GPRS range ?
- Are all services available on the device ?
- Where exactly in the processing is s/he ?
- Are any errors being reported ?
- Can they be quoted ?
- That is, the user should have ready access to this information
- Manage user and developer requirements for error messages carefully

## Secure / locked down device



- Device is mobile, so needs to be more secure
- May be operating in kiosk mode or locked down in some other way
- This means
  - It may not be possible to get at all required information
  - Logging needs to take this into account
  - May make fault determination harder

## Logging



- Logging should be designed into the application from the start
- How to get hold of the logs in a mobile scenario ?
- We can not normally connect to an individual device
- Avoid / reduce physical returns as much as possible
- It should be possible to send logs up to a server, either on demand or
- Logs are sent regularly and can be centrally examined
- Consider housekeeping of logs

## Housekeeping of Logs



- Client or server or both will have to do housekeeping of logs, e.g.
- The size of any logging is fixed and as new logs are created old logs are deleted
- Logs can be assigned a particular lifetime
- This will probably not help with a particular error, as it may have occurred just after the last log from the device has routinely been dispatched and what happened just now is not logged
- It would be useful to also design into the application an automatic log purge when particular catastrophic events occur
- However, catastrophic events rarely leave time to invoke the 'catastrophic event' code that could do logging or purging

## Outline



- Systems Maintenance
  - A closer look at maintenance work
- Designing Any System For Maintainability
- Maintainability
- Sidebar: Patterns, Practices and Software Factories
- Systems Architecture
- Architectural Differences Between LAN Based and Mobile Applications
- **Development Process**
- Summary

## Development Process



Almost all formal development processes (UP/RUP or Microsoft's MSF) cover the front end of the life cycle, including

- Requirements capture
- Analysis
- Design
- Implementation (MSF ends at deployment)

## Development Process



Back end of the life-cycle (i.e. most of the software's life) is rarely considered

- Deployment environment constraints
- Resource configuration
- Testing setup (harnesses, test cases etc)
- Instrumentation for performance metrics
- Maintenance & Enhancement

Yet 'The back end of the life cycle harbors massive complexity' (Greenfield/Short)

## Development Process cont'd



Development processes also need to be maintained.

Current typical process weaknesses include

- We too often 'do what we've always done'
- Process does not keep in step with new technology (e.g. procedural analysis method front-ends OO development)
- Often key technical roles are sourced from outside the organisation and leave at end of project, consequently
- Knowledge is lost from project to project
- There is still a major reluctance to spend effort up-stream of the process where it cannot be directly linked to return

## Development Process cont'd



Considering non-functional requirements properly takes time and

- Requires more work up front
- There is every temptation to get on with 'what we are being paid for' ('WISCY')
- Consideration should be given to
  - Who develops the system
  - Who uses it
  - Who maintains it
- Input is required from all three parties, the third one at this point is often not on the scene yet and/or not considered
- Maintenance is often considered a 'low value' activity, when this is where the software pays for itself (PPP financed projects)
- The harder software is to maintain, the more expensive

## Development Process cont'd



### Teams and the split of responsibilities:

- Development and support teams are not the same
- Could be different companies, input may be impossible
- Could be part of the same company, contract for new development and subsequent maintenance
- There should be synergies between the teams
- Support developers could be 'embedded' into the development team for knowledge transfer
- The maintenance team should have its own maintenance architecture and
- Should have input into the (non-functional) architecture of the new system

## Development Process cont'd



- How to ensure that developers follow the rules and guidelines that they should with regard to style / usage of language features etc ?
- Set up project templates, guidance packages etc to force compliance
- Supervise developers properly
- Undertake regular code reviews
- Peer based development
- Agile methods etc

## Outline



- **Systems Maintenance**
  - A closer look at maintenance work
- **Designing Any System For Maintainability**
- **Maintainability**
- **Sidebar: Patterns, Practices and Software Factories**
- **Systems Architecture**
- **Architectural Differences Between LAN Based and Mobile Applications**
- **Development Process**
- **Summary**

## Summary



- Maintainability and other non-functional requirements need to be designed into a system from the start
- The added complexity of the mobile platform makes this imperative
- The development process, not just software, needs to be maintained
- The process needs to be improved from project to project
- More emphasis needs to be given to the process back-end
- Patterns, Practices and Software Factories can help and
- Will move software development into the industrial age

## Books

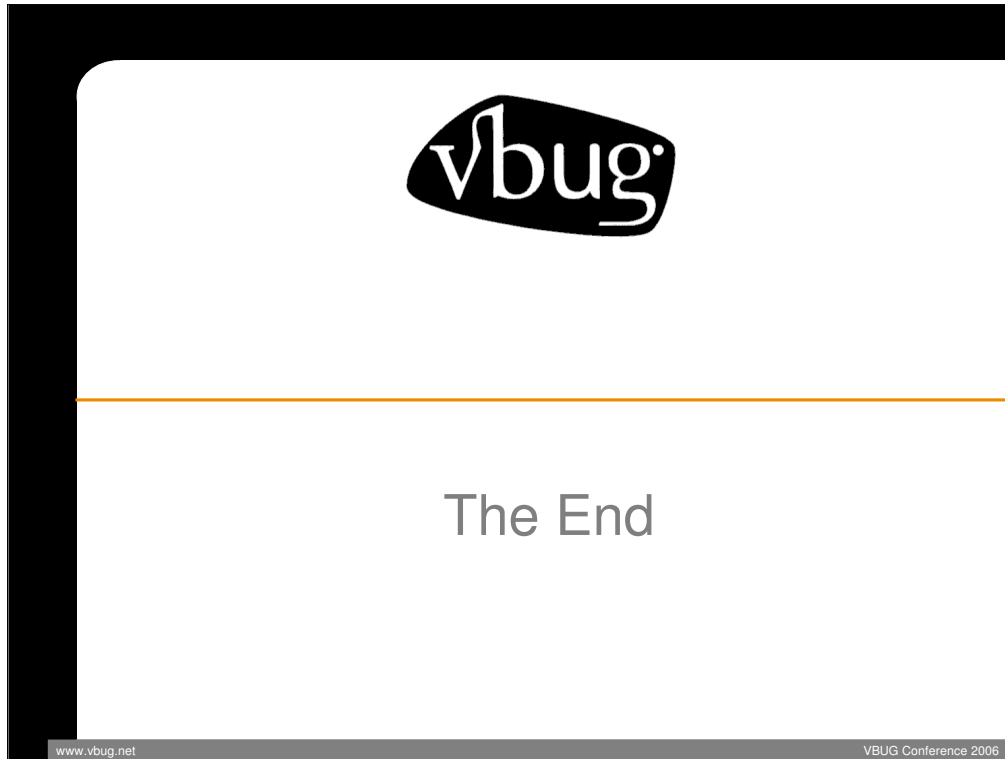


- Gamma, Helm, Johnson, Vlissides: Design Patterns, Addison Wesley, 1995
- McConnell: Code Complete 2, Microsoft Press, 2004
- Microsoft Press 'Patterns and Practices' books and on-line resource
- Greenfield & Short: Software Factories, Wiley, 2004

## Other Resources



- [www.rational.com](http://www.rational.com)
- [msdn.microsoft.com/windowsmobile/](http://msdn.microsoft.com/windowsmobile/)
- [msdn.microsoft.com/practices/](http://msdn.microsoft.com/practices/)
- [www.softwarefactories.com](http://www.softwarefactories.com)





Questions ?



## VBUG Conference 2006

18<sup>th</sup> October 2006

Microsoft Campus, Reading, UK

---

### Designing A Mobile System For Maintainability

Martin Lixenfeld

[martin.lixenfeld@atosorigin.com](mailto:martin.lixenfeld@atosorigin.com)