Principles of
Good Software Development

High-level Objectives

From a customer's perspective, good software should play a part in achieving the following for their business.

Cheaper

- Optimise Costs
- Optimise people
  - Prefer automation over expensive humans

Smarter

- Eliminate inefficiencies
- Simplify; reduce complexity
- Minimise operational overheads

Better

- Deliver what the customer needs
  - (business, stakeholders, clients etc)
  - Quality of service to customers is primary
- Minimise the need for End-User Developed Applications
  - In a large enterprise, proliferation of EUDAs can become a burden

Principles: Culture

- Embrace Agile
  - small teams, self-sufficient, driven by product visions, serving customers, that can change direction quickly
  - without tight coupling to the rest of the org / architecture
- Embrace DevOps
  - which means get rid of IT Support teams, get rid of Infrastructure teams
  - combine dev & ops as one
  - not through org or process, through culture and constant collaboration
- Incentivise teams to fix issues rather than work around them
  - If you "address" the issue but the solution is more complex, with more process, or more people, you've failed
- Use the "5-why's"(six-sigma) to properly understand an issue
  - Properly understand the actual low-level cause not just a symptom
- Federate architecture (enterprise/systems/data)
  - Prefer good culture, cross-team collaboration to ivory towers
  - Facilitate with good information sharing tools (best of breed wiki's, best of breed CASE tools, open access to information)
- Embrace (internal) Open Source
  - If you build something useful, share your service with the community
  - which could be the world, or within your own organisation
- Think LEAN

Principles: Culture/People

- Hire good people
- Let them do their job
- Keep good people
  - Hire motivated, smart people and provide an environment for them to deliver properly
- Foster openness and evolution (via natural selection)
  - Enable the good people to collaborate, and innovate
  - Prefer incremental innovation (where good ideas/solutions survive) to big, preconceived solutions
- Take inspiration from the natural/'outside' world
  - Observe, learn, apply rather than trying to solve problems in isolation
- Don’t tolerate mediocrity
  - Proactively exit non-performers, annually

Principles: Architectural Drivers

- Identify a simple architecture
  - If you can't express an idea simply, it probably isn't simple
  - Make it is as simple as possible, but not simpler
- Design for automation
- Consider scalability - up and down!
- Think about how cost effective it is (easy to understand, optimise)
- Focus bespoke engineering effort on differentiating services
  - Consider off-the-shelf products and re-use for routine services

Principles: Architecture

- Think outside your company; look at what companies/individuals in the outside world do.
- Prefer open standards to proprietary inventions
- Prefer open source software to vendor lock-in
- Prefer established design patterns to one-off solutions
- Prefer ("real-time") API interfaces to ("batch-based") file interfaces
  - A message-bus can be an API
- Prefer collaboration and knowledge sharing to committees and process
- Prefer time-boxed focused spikes/prototypes/proof-of-concepts to debate
- Prefer delivery to presentation
- Prefer engineering diagrams and code to PowerPoint, Word etc
- Prefer evolution to revolution (less risky)
- When integrating a software package into a proprietary stack, abstract
  - Avoid vendor lock-in by abstracting the integration points
  - Allow for technology swap-outs in the future
  - Use the core functionality of the software package, not every feature
- Plan for operational monitoring early
  - expose run-time performance metrics
  - KPIs with SLAs etc
  - eg to enable continuous monitoring of STP

Principles: Architectural Corollaries

- Don't fall into the trap of building vertical (asset specific) solutions
- Don't fall into the trap of building siloed (function specific) solutions
- Be aware of building a complex, siloed utopian architecture

Principles: Enterprise Architecture

- Consider layers:
  - Data services
  - Computation services
  - Workflow services
  - User interaction services
  - Collaboration services
  - ...
- Look for similarities and synergies
  - with a view to building cross-function, cross-asset services
  - Avoid clear duplication; try to re-use systems, services, components etc

Principles: Data

Data Modelling

- Data are facts
- Facts should have keys (identifier)
- Facts should have a master source
- There can be many master sources, but only one master source for a given fact
- A fact should be versioned
- The history of facts should be recorded
  - eg bitemporal, tuple versioning

Data Quality

- Perform Root Cause Analysis and fix data errors/inconsistencies at source
- Prefer prevention and proactive detection rather than allowing errors to propagate
- Enrichment of facts should be via the keys of the facts
- Minimise manual data adjustments and corrections; reduce operational overheads

Principles: Data Corollaries

- A fact should have a single master source; there should be a single *version* of "the truth"
  - Reduce data duplication
  - Minimise the need for reconciliations
  - Where possible only amend data at its master source
  - Data "copies" can lead to bifurcation, divergence, staleness and the need for reconciliation
- Publish data from the master source once
  - Don't forward data
  - Don't pass data through intermediate systems
  - All functions across the organisation should use the same data source(s)
- Leverage consolidation of data (stores)
  - Fully realise the value of the data, and minimise the costs of owning/managing the data
  - Realise this via storing the truth once, without duplication or lossy copies

Principles: Data – “grown up” bits

Data Governance, Architecture and Documentation

- Metadata associated with Master sources should be documented and maintained
  - This governance could be called Data Architecture

Data Legal, Regulatory and Control Concerns

- Data must take into account all applicable local legislation, statutory obligations and relevant company policies
- Data must abide by all relevant IT, audit, data secrecy and security policies
- Consider data obfuscation, for instance when publishing for example by copying from production to test.

Principles: Software & Automation

- Optimise what you have
  - Prefer incremental improvement to new build / replace
  - Unless the 5-why's and these rules clearly indicate build/buy is better
- Reduce complexity rather than building on it
- Automate processes rather than employing people to provide operational processing
- Use extensive comprehensive detailed system monitoring (hardware, software, data, workflows) tightly coupled with the DevOps team to fix issues quickly and effectively
  - Minimise/remove handovers, written SLAS, email, "over-the-fence" mentality
  - Codify all important metrics/MI/control rules/measures; don't rely on people to interpret/escalate etc

Principles: Software/Testing

- Incorporate testing into development so that unit testing is standard
- Implement assurance tests (agreed with product owners, stakeholders etc)
- Automate testing
- Promote visibility and transparency of the test results data to all stakeholders

Principles: Software/Services

- Read Jeff Bezos' mandate
- Use open-standards APIs
  - eg REST, SQL, ODBC, JDBC
- Provide fit-for-purpose APIs
  - eg Authentication, Authorisation, Enablement, Perimissions
  - eg CRUD, Query, Streaming
  - eg Transactions support
- Converge on common APIs (technologies, patterns)
  - where it makes sense, and in order to ease interoperability
  - but not at the expense of innovation or evolution

Principles: Software/Data Services

Data Services

- Prefer enterprise strength technology solutions
  - For example, spreadsheets can rapidly get out of control
- Prefer shared services
  - For example, bifurcated and stale copies on email, can rapidly get out of control

Data Service Corollaries

- Data should be made available via quality services that meet organisational requirements
- Do not (force application teams to) store local copies or duplicate data
- Prefer application teams to obtain data from the master sources
  - at run-time, on-demand, as required, rather than caching locally

Principles: Infrastructure (hardware)

- Prefer cloud over managing and maintaining your own data centers / servers / infrastructure
- Prefer tools over manual processes/procedures
- Prefer scripts/code over documentation
- Prefer machines over people

Principles: Buy vs Build

Services, Software Packages and Building Software

When it comes to composing a software system, there are a myriad ways of acquiring, integrating and running software.

Software offerings

In general, one could consider:

1. Prefer buying a *platform* over buying a *service*
2. Prefer buying a service over buying a software package
3. Prefer buying a *software packags* over *building software*

Buying a vendor platform, typically involves paying a vendor to manage the software system for you. It may be ran on-site or off-site, but the vendor will manage it all for you. Your users just use the platform.

Vendor platforms and vendor services typically both entail paying an annual license for a period (years), and in return getting a *managed service*.

- Vendor Platform (e.g. see: Gartner's Trading Platforms "magic quadrant")
- Vendor Service
- Vendor software package
- Vendor software component

Another option is to buy a service, and integrate that service into your system.

Principles: Buy vs Build

Service offerings (explained)

- Platform-as-a-Service
- Software-as-a-Service
- Storage-as-a-Service
- Compute-as-a-Service
- Infrastructure-as-a-Service

Beyond platforms and services, you can integrate a smaller scale vendor software package or some software (components) into your own system. Vendor software and components may incur an initial license (with perpetual rights to use) or an ongoing license fee.

Principles: Buy vs Build

Software aspects

- All software has various aspects.
- Free vs Freemium vs Paid
- Open Source vs Closed Source
- Open license vs Proprietary License

Principles: Buy vs Build

Considerations

- There are many considerations when determining how to run a software system.
- Initial cost
- Ongoing cost
- Support & Maintenance (including cost)
- Expected lifetime
- Features
- Infrastructure requirements
- Operational requirements (how is your operating model affected)
- Customisability (what can be customised)
- Evolvability requirements (what may need to be changed)
- Upgrade paths
- Cost of change
- Cost to switch (decommission, replace, vendor lock-in)
- Appetite to managing software development
- Regulatory constraints (eg off-site hosting of data, encryption)
- Service Level Agreements
- Contractual Rights
- Expertise required (in-house)
- Risk of function fragmentation; bought "bundled" functionality duplicating what already have, so risk of fragmentation