



Building Business-Critical Software with Open Source

Greg Olson, Principal Consultant
golson@opensource-sense.com

© 2018 Open Source Sense, LLC

Agenda

- Introductions
- Defining business-critical in the context of open source
- Technology-centric and process-based approaches
- Forking and the cost of technical debt
- Building community visibility to support your product roadmaps

Open Source Sense

Maximize your ROI with Open Source Software

Open Source Software Strategy

- Business Strategy
 - Monetization
 - Community Development
 - Business Development
- Technology Strategy
 - Evaluation & Selection
 - Alignment & Ecosystems
 - Legacy Migration



Open Source Software Management

- Assessment & Optimization
- Program Development
 - Policy
 - Process
 - Implementation Planning
- Training


Our consulting team has engaged with over 350 organizations to develop practical open source strategies and efficient management programs

Business-Critical Software

- Software systems or applications that
 - Are important to the strategy or operation of a business
 - Must operate properly whenever the business is operating
- Implied requirements
 - A high-availability architecture
 - A recovery capability
 - Comprehensive security capabilities
 - Expert support available to manage these systems



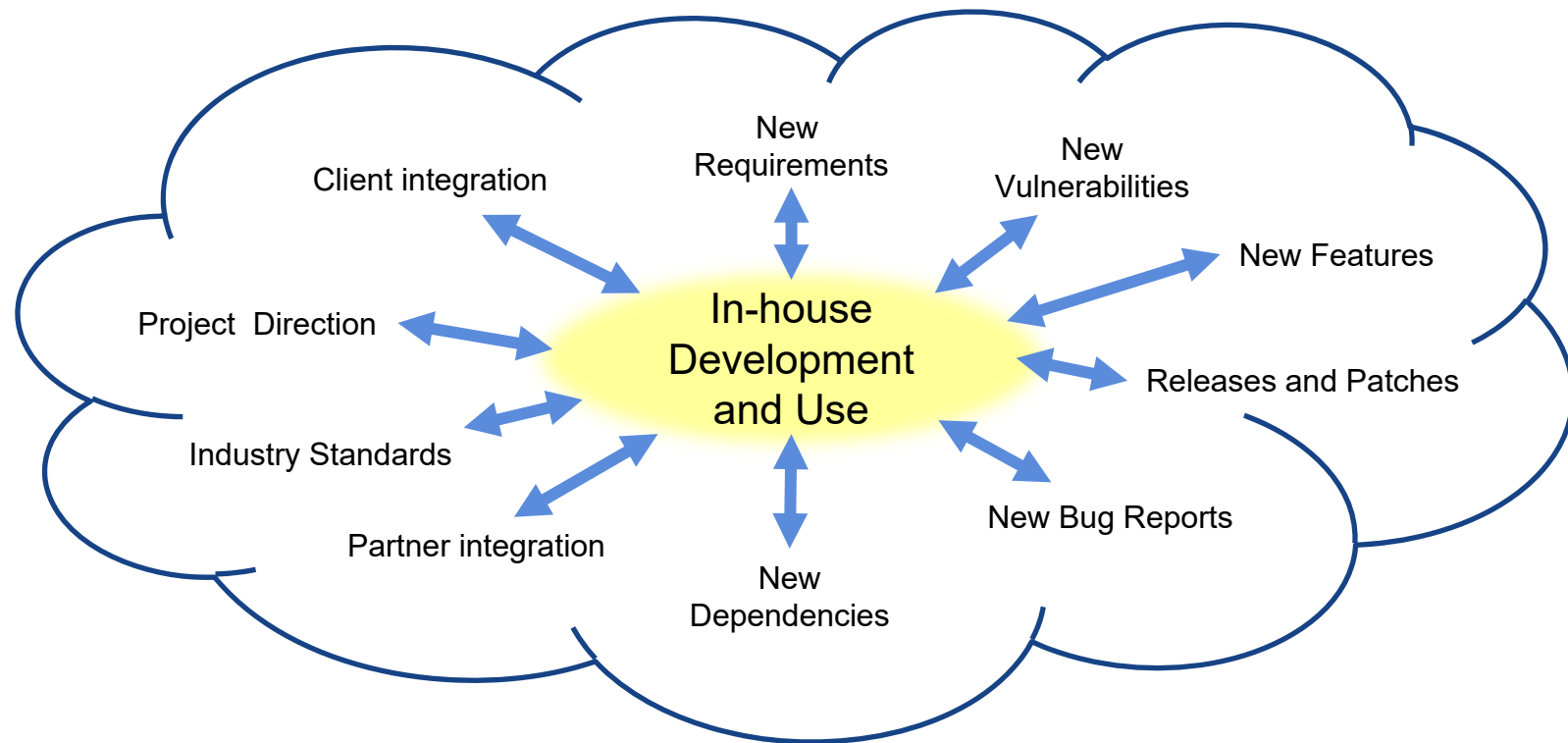
Best Practices for Business-Critical Development



In-house
Development
and Use

Mature Practices Exist in Most Organizations

Plus Open Source Collaboration



Companies and OSS Projects

Product Companies Industrial Process

Top-Down
Management



Formal
Methodology



Processes
Metrics

Revenue
Contracts
SLAs



Customer
Organizations



Conceptual Divide



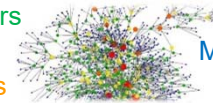
Open Source Projects Collaborative Process



Consensus around
Technical Merit

Sponsors

Developers



Members

Diffuse
Processes



Tools
Scripts



Repository
Wiki
Lists



Users

Strategic vs. Tactical Open Source

Strategic ←————→ Tactical

- Unique ecosystem position
- Unique functionality
- Uniquely compatible implementation
- Prohibitive switching cost

- No ecosystem preferences
- Multiple alternative implementations
- Multiple implementations available
- Interfaces standardized or easy to swap out

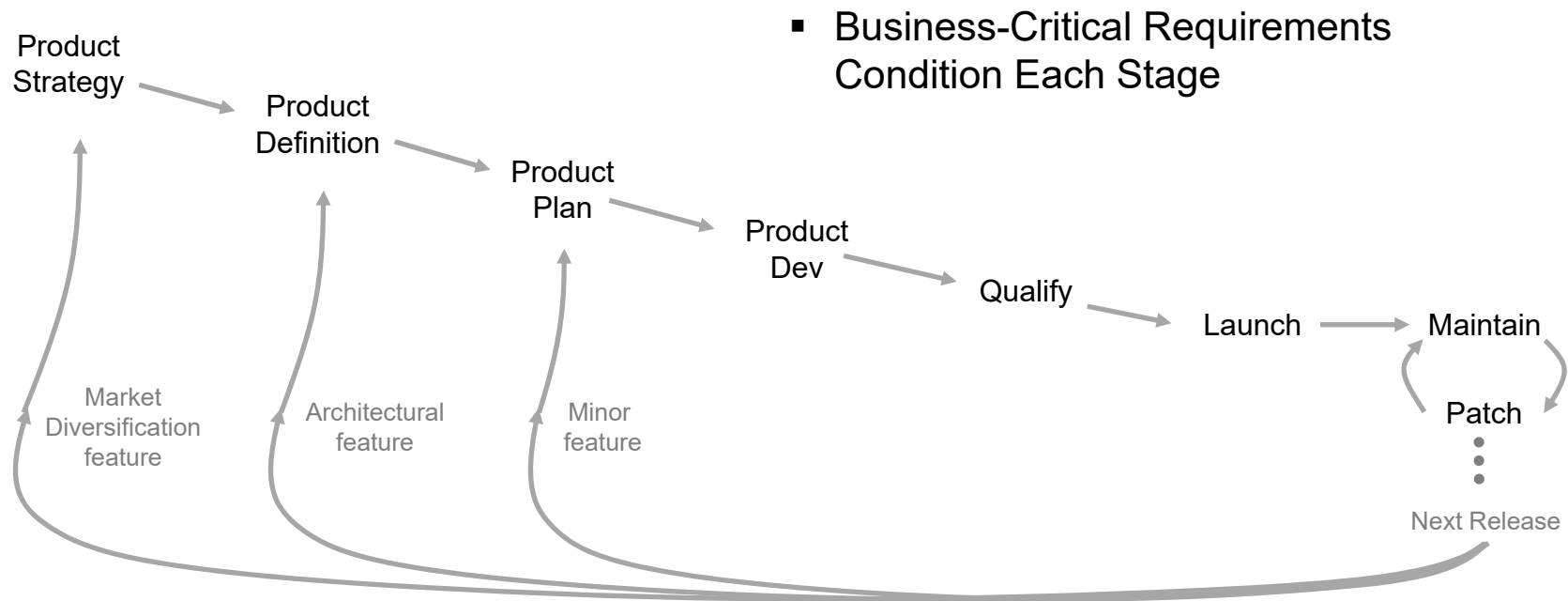


Modification and Technical Debt

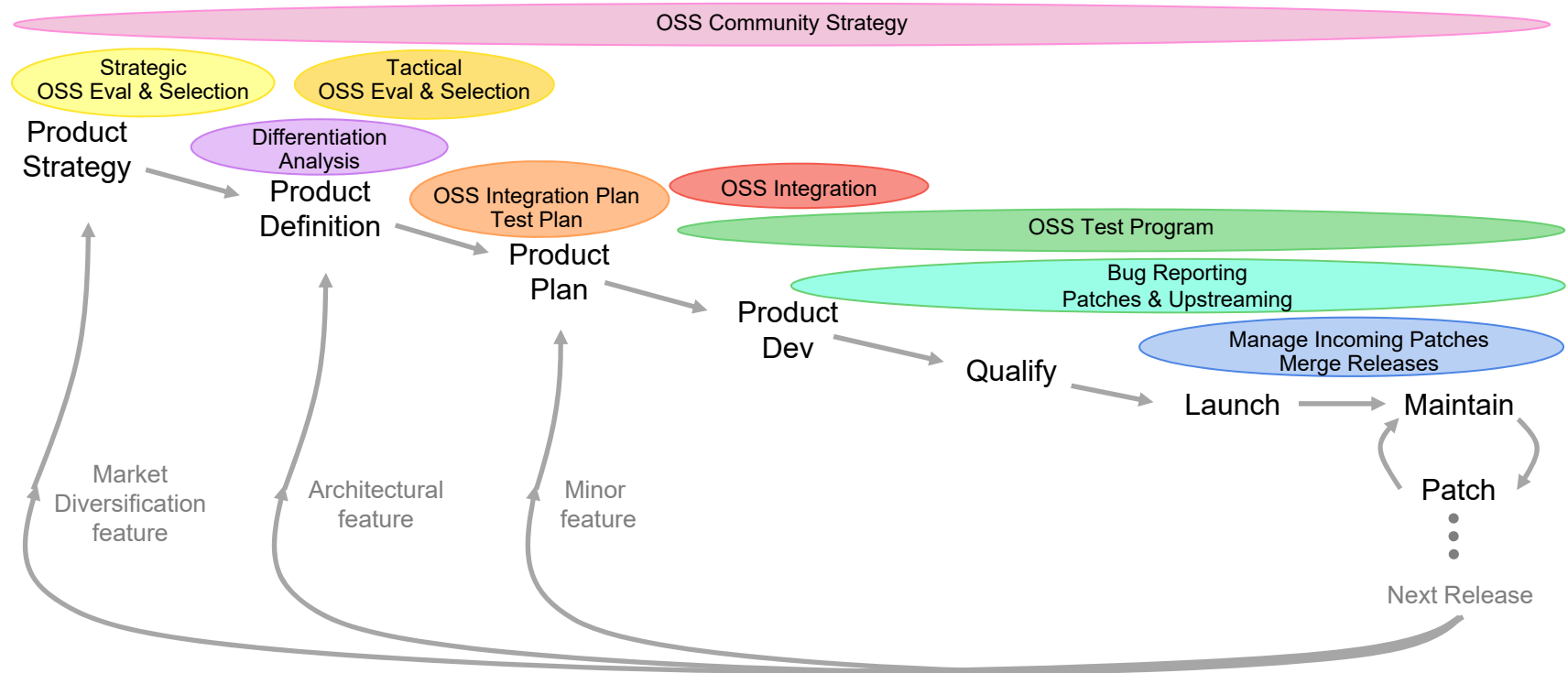
- Modifying OSS code is very expensive
 - Lifecycle costs include
 - Initial design, coding and test
 - + Integration and test with every new OS patch and release (called Technical Debt)
 - A modular approach (limiting patches to hooks) can reduce cost
 - Costs at 15-25% modification level is typically greater than completely private code over a lifecycle
- Forking (not synchronized) incurs significant disadvantages
 - OSS project brand, community contributions, integration confidence lost
 - New functionality or changes to the OSS version may leave you marooned



Typical Commercial Product Development Cycle



Product Development with OSS



1. Community Strategy

- Community strategies typically evolve organically but benefit from conscious planning
- Identified Best Practices
 - Select strategically important OSS projects for focus
 - Seek committer / maintainer roles in identified project communities
 - Adapt your development teams to OSS project culture, practices and tools to succeed with their strategically important projects



2. OSS Evaluation & Selection

- There is tremendous leverage in choosing the right OSS project/community at the outset
- Most companies required at least some of “due diligence”
- Tactical and strategic OSS decisions require different evaluation

Tactical

- Architectural and functional fit
- License compatibility
- Security vulnerability history and status
- Code quality
- Documentation quality
- Community maturity and stability

Strategic

- All tactical criteria
- + OSS project stature in target market
- + Direction compatible with company strategy
- + Other project participants and level of commitment
- + Opportunity to participate in project leadership



3. Differentiation Analysis



- Should a new feature be proprietary or open source?
 - A constant activity with proprietary products built with OSS
 - What best supports your company's product and market strategies?
 - Even previous decisions should be re-evaluated periodically to accommodate changes in product landscape and competitive strategies.
- Any features or customizations not accepted by an OSS Project are inevitably proprietary
- Identified best practices
 - Develop a standard multi-dimensional evaluation technique
 - Apply to each proprietary feature proposal

4. OSS Integration and Test Planning

- Best practices

- When most of the code for a product is sourced from a single OSS project, normalizing your own engineering practices with those of that project greatly simplifies integration
 - Seamless interoperability with code repo, bug tracking, release process, etc.
 - Faster on-boarding of contributors to the relevant OSS project
- Test and QA acquired OSS code AND post-integration together with dependencies and value-added product software and hardware
 - Utilize OSS project test code when available
 - Develop tests for OSS where needed to meet business-critical requirements
 - Plan to contribute enhanced test code to projects



5. OSS Integration

- Identified best practices
 - Large and small organizations integrate directly from OSS trees
 - Product teams typically given freedom to choose appropriate versions
 - Strictly minimize customization of OSS to keep patch loads manageable
 - Modularize changes, extensions to the OSS wherever possible
 - Implement for automated continuous integration



6. OSS Test Program

- Need to test OSS standalone and integrated
 - OSS module unit testing
 - OSS project / sub-system and/or platform testing
 - Final product/service testing with integrated open source code
- Successful organizations integrating open source
 - Contribute all OSS test code to projects so releases arrive pre-tested
 - Develop relationships with OSS project leaders to facilitate upstreaming

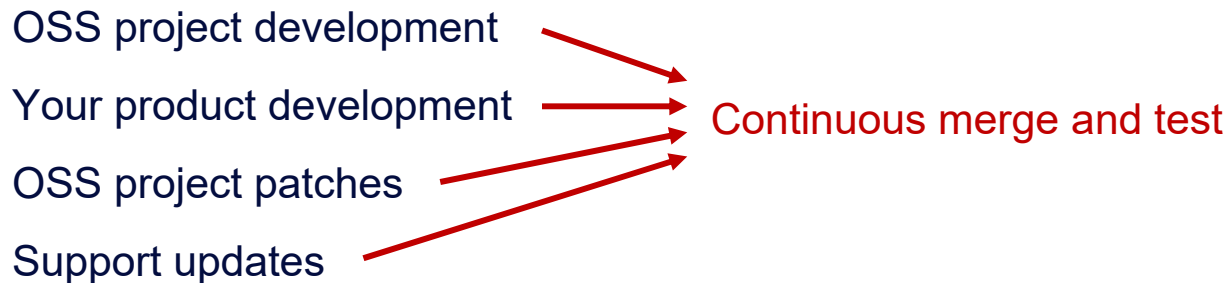


7. Bug Reports, Patches and Upstreaming

- Common core practices for upstreaming
 - Most successful organizations invest in upstreaming early
 - Build community / maintainer relationships
 - Retain minimal forked code as “value-added”
 - Large Orgs (Samsung, Red Hat et al.)
 - Company ID does not guarantee upstream patch acceptance
 - Committers assigned to the projects improve the odds
 - Small Orgs (smaller OEMs, integrators, companies)
 - Patches reviewed on merit, as with large contributors
 - Even more important to consider project style, roadmaps, etc.



8. Manage Incoming Patches/Releases



- Complexity of the problem often leads to slow and expensive processes
- Identified best practices
 - Strictly minimize customizations in order to keep the patch load manageable
 - Keep retained changes small and modular to streamline merging
 - Cultivate OSS project relationships to enhance communication and minimize skew
 - Invest in project test code to minimize quality issues in OSS updates
 - Use available tools merging capabilities (patch, git/github, etc.)

Summary

- Many techniques to meet the requirements of business-critical software with open source have been proven in the industry
- These techniques
 - Rely heavily on well-established life-cycle development practices
 - Add processes to couple OSS project dimensions with each step of the life-cycle
- By employing these techniques organizations can realize the benefits of open source software in their most business-critical systems and applications





Questions?