SUCCESS FACTORS FOR SOFTWARE REUSE

Name	ID	Section
Metsehafe Eyasu	ETS0852/13	C
Michael Engida	ETS0859/13	C
Mohammed Mehad	ETS0907/13	C
Mohammed Restem	ETS0911/13	C
Mohammed Shemim	ETS0912/13	C

SOFTWARE REUSE

- Focus: Technical factors enabling effective software reuse in enterprise environments
- Literature review: Recent sources (2022–2025)
- Deliverables: Summary of key themes & best practices (report and slides)

WHY SOFTWARE REUSE IN ENTERPRISE?

- Definition: Reuse = leveraging existing software assets (code, services, etc.) in new applications
- Benefits: Faster development, reduced costs, improved quality (use of proven components)
- Challenges: Requires planning and infrastructure – ad-hoc reuse can fail (integration issues, duplicated effort)

MODULAR ARCHITECTURE (MICROSERVICES)

- Modularity = Reusability: Design systems as independent modules/services for easy reuse
- Microservices: Small, single-purpose services communicating via APIs – fosters loose coupling
- Impact: Enables teams to reuse or replace services without impacting others (independent deployment)
- Example: Fine-grained microservices at Ericsson improved reuse and maintainability

CODE DESIGN AND ENCAPSULATION

- Write components with high cohesion, low coupling (self-contained functionality)
- Follow SOLID principles, separation of concerns – easier to plug into different contexts
- Design for reuse: Externalize config, avoid hard-coded logic, use abstraction so code is generic
- Upfront investment: Extra effort to make components reusable (estimated +60% cost), but pays off over time

API DESIGN & DISCOVERABILITY

- Clear APIs: Well-defined interfaces/contracts for modules/services encourage reuse (easy integration)
- Stable versioning and backward compatibility to support multiple consumers
- Documentation: Provide API docs, examples, usage guides – lowers adoption barrier
- Discoverability: Internal service catalog or marketplace so teams can find reusable components

DEVOPS INTEGRATION (CI/CD & AUTOMATION)

- CI/CD Pipelines: Automate build, test, deploy of reusable components across projects
- Shared artifact repositories (packages/containers) for easy access to libraries and services
- Containerization: Encapsulate services (e.g., Docker, Helm) – consistent deployment everywhere
- Continuous integration tests: Immediately verify a shared component with all dependent systems

TESTING AND QUALITY ASSURANCE

- Rigorous testing = confidence to reuse
- Unit & integration tests for each component (ensure it works in isolation and with others)
- Contract testing: Verify services meet API expectations of consumers (no breaking changes)
- Automated regression tests and static analysis (catch issues early for any update to shared code)

STANDARDIZATION & GOVERNANCE

- Coding Standards & Templates: Uniform development practices (structure, naming, frameworks) for shared components
- Design rules: e.g., Ericsson's reuse guidelines & maturity model for microservices
- Governance: Assign ownership/maintainers for reusable assets (accountability for updates & quality)
- Manage versions, dependencies, and avoid duplicate components (reuse vs. reinvent decisions)

INNERSOURCE & COLLABORATION

- InnerSource: Apply open-source model internally – open repositories, cross-team contributions
- Breaks down silos, improves knowledge sharing and reuse of code across org
- Contributors from other teams help improve shared components (fixes, features) – scaling reuse maintenance
- Culture shift: Encourage reuse/contribution via recognition, metrics (e.g., track reuse rates)

CONCLUSION – KEYS TO SUCCESSFUL REUSE

- Plan ahead: Incorporate reuse in architecture & design (modularize, define reusable services)
- Enable with tech: Invest in CI/CD, tools, and catalogs that make reuse easy and safe
- Maintain quality: Test thoroughly and enforce standards for all reusable assets
- Encourage culture: Promote collaboration (InnerSource), reward reuse – make it part of “how we build software”
- Result: When done systematically, reuse yields long-term quality and productivity gains that far outweigh the initial costs

THANK YOU