TRAVELER | ARTISAN | EXPERT | PROFESSIONAL | MASTER

CONTINUUM LEVEL UP TOOLKIT — LEVEL UP — v1.0

STRATEGY ALIGNMENT

1.
- Any alignment with strategy is coincidental or opportunistic.

2.
- Upfront engagement with stakeholders to ensure business and technical alignment.
- The product/project vision is explicitly aligned to strategy
- Post-implementation review to confirm strategy alignmment.

3.
- Occasional engagement with stakeholders throughout delivery cycle to review business and technical alignment.
- Backlog items are created to deal with strategy alignment issues.

4.
- Frequent engagement with stakeholders to review business and technical alignment.
- Team presents product to achitecture and design teams with the intent of picking up alignment issues.
- Metrics to measure strategy elements defined and tracked.

5.
- All major strategy alignment backlog items have been resolved.
- All requirements are attached to a business metric.
- Metrics are tracked over time and improvement targets set and achieved.
- Team independenty innovates by creating new features or optimisations in support of or extending the strategy.

PLANNING AND REQUIREMENTS

1.
- Team is not involved in estimates.
- No specific prioritisation of requirements.
- Iteration lengths are erratic based on the amount of functionality
- The team does not know their velocity.
- No stakeholder management in place.

2.
- Team performs estimates up-front.
- Requirements are prioritised based on business value.
- Iteration lengths are fixed.
- The team knows their velocity.
- Stakeholders have been identified and a communication plan is in place

3.
- When the team estimates the estimates include all activities to reach 'Done'.
- Team performs estimations iteratively.
- The MVP and MVFS have been defined.
- The team velocity is predictable.
- All backlog items are sized by the team.
- Stakeholders attend showcases.

4.
- Team tracks performance against estimates.
- Requirements are developed on a just-in-time basis.
- Analytics are implemented to determine the effectiveness of requirements.
- The amount of functionality for each iteration is determined by the team's velocity.
- Technical Debt and Defects are tracked on the backlog and form part of the estimated team velocity
- Stakeholders actively participate in retrospective

5.
- Requirements are defined with an expected outcome with an objective measure.
- Release planning is performed based on the team's current velocity.
- A process is in place (e.g. a formal beta program) to allow extended stakeholders and customers to evaluate the software and provide feedback.

CODING PRACTICES

1.
- No standards or mechanism for ensuring code quality
- There is no awareness or transparency of current risks and issues in the team.

2.
- Guidelines and/or standards are defined.
- Consistent training process for new team members is in place.
- Each identified risk has been assigned a risk mitigation or action plan.
- Risks are discussed as part of the iteration planning process.

3.
- There are mechanisms in place to ensure that standards are followed.
- Process and practices are understood and followed by all.
- Metrics are defined but not necessarily reviewed and acted on.
- The team proactively improves the code.

4.
- Code Metrics are part of build automation and continuous integration.
- Code Metrics are tracked for trends and adjustments made on a continuous basis.
- The team regularly performs katas with the objective of improving their skills.
- Code is regularly refactored as part of the iteration.

5.
- Standards are regularly reviewed and updated.
- Processes and Practices are regularly reviewed and updated.
- The code metrics are assessed and backlog items created to drive improvement.
- Technical Debt is minimised.

CONTINUOUS INTEGRATION

1.
- No version management of artefacts and reports.
- Deployments & Rollbacks are manual.
- Build is performed manually and infrequently.
- Build is owned by a specific person.
- Testing is manual
- Development environments manually provisioned and shared between teams.

2.
- Source code is under version management.
- Build is automated and tests are run as part of the build.
- Testing partially automated, and code coverage greater than 0

3.
- Team checks code into version management system on daily basis.
- All artifacts are under version management.
- Build and deployment to development environment are automated.
- Build status and broken builds are visible to all.
- Anyone in the team can start build at any point in time
- Testing partially automated - greater than 50% code coverage.
- Release and Change Management partially automated.

4.
- Automated build and test cycle every time a change is committed.
- Build metrics gathered, visible, and acted on.
- Builds are not left broken and code is not commited on a broken build.
- Deployment to Test environment is automated.
- Non-functional testing is automated.
- Testing is automated as much as is practical.
- Environments can be provisioned at on demand.

5.
- Build and deplyment are automated as much as is practical.
- The build pipeline extends directly into production.
- Tests are run in parallel across multiple machines.
- Trunk-based development is the standard practice and integration happens continuously. Branches are rarely created and are short lived.
- Check-ins occur multiple times each day.
- Environment & infrastructure specifications are managed and versioned along with all other artefacts.
- The CI build creates and provisions environments to allow scalability for testing.
- Release and Change Management fully integrated into deployment process.
- The product is always in releasable state.
- Automated release process, release to customers is a business decision.

INCIDENT MANAGEMENT

1.
- The process for handling problems and incidents is ad-hoc.

2.
- An incident management process is in place and understood.
- Key people are identified for incident management
- The number of handoffs and teams involved in incident resolution is minimal.

3.
- Incidents and problems feed into backlog
- Operational requirements are identified and tracked in the backlog.
- Problem management includes an urgency to address root cause
- Root cause analysis is consistently performed.
- Level 1, 2, 3 support structures in place
- Instrumentation (monitoring) in place

4.
- Defects are resolved once and deployed automatically across all environments
- Defects resolution includes full regression tests
- System health is proactively monitored
- Root cause analysis is highly valued and regularly trended.

5.
- Feature Teams do own incident management
- Fail forward (failing in a way that enables you to identify and overcome underlying problem, encapsulate the way forward and reduce the likelihood of failure next time around)
- Team actively manages, monitors and reviews what happens in production, feedback loop is enabled and acted on.

RISK AND ISSUE MANAGEMENT

1.
- Risk management does not exist or is just a box to tick in order to get through a process (i.e. something done to keep risk and audit people happy).
- There is no awareness or transparency of current risks and issues in the team.

2.
- Risks have been identified and are captured using an appropriate artefact such as a risk story wall or risk register.
- Each identified risk has been assigned a risk mitigation or action plan.
- Risks are discussed as part of the iteration planning process.

3.
- Risks, issues and blockers are discussed and updated in appropriate detail as part of all sessions (stand-ups; iteration planning; showcases; steering committee meetings).
- Unmitigated or uncontrolled risks are easily identifiable
- Risks, controls and action plans are assigned to appropriate owners: someone who understands the risk, has responsibility and accountability for managing the risk, has interest in the risk, and has the authority to implement controls and actions.

4.
- The cost and benefits of risk mitigations and actions are evaluated and recorded before they are implemented
- Risk management roles, responsibilities and accountabilities have been defined and agreed upon.
- Regular check points with business stakeholders to ensure they are aware of all risks and issues

5.
- Risk mitigations and actions are explicitly designed to increase positive outcomes and reduce negative outcomes
- The business participates in developing mitigation strategies
- Risk management is used to identify potential opportunities and drive strategy by thinking about risk in terms of uncertainty (which could be positive or negative)
- Measures and metrics exist to demonstrate the effectiveness of risk management and where improvement is required
- The financial impact of risk and issues is clearly understood

TRAVELER | ARTISAN | EXPERT | PROFESSIONAL | MASTER

CONTINUUM LEVEL UP TOOLKIT — LEVEL UP — v1.0

SOFTWARE DESIGN

1.
- Speculative design and technical decisions are made on an ad hoc basic. There is no vision or long term technical planning.
- BDUF - Big Design Up-Front
- Solutions are tightly coupled and brittle.

2.
- Regular Design reviews in place.
- Design assumptions are tracked and validated during iterations.

3.
- Design issues are tracked and prioritised in the backlog.
- Non-functional requirements are documented and tracked.
- Clear interfaces defined between modules.
- Design is owned by the team.

4.
- There is partial traceability between design and requirements.
- Design requirements are expressed as automated tests.
- Non-functional requirements are defined and associated metrics tracked automatically.
- Where practical all modules of the system can be simulated.
- Where possible design is captured in a model and code generated.

5.
- There is full traceability between design and requirements.
- Design activities are performed on a Just In Time basis.

TEAMING

1.
- The team cannot make decisions without consultation.
- The team goals and vision are unclear.
- The team doesn't clearly understand entire stakeholder landscape.

2.
- Team vision and goals are agreed, captured and understood.
- There is a clear understanding of who are the team's stakeholders. (Consumers, partners and clients)
- There is a clear definition of what the team's success is (definition of 'done').

3.
- The team actively seeks feedback in an effort to achieve the goals.
- Transparency and high level of collaboration to and from the stakeholders are in place.
- Impact of blockers and dependencies are minimised and managed and the team knows what these are.
- Transparency and high level of collaboration within the team are in place.

4.
- Balanced participation is in place. Team members contribute as appropriate and each member's opinion is valued.
- Team uses adequate measures to monitor success.

5.
- All decision-making is internal to the team.
- Complete collective ownership is in place, both when team is successful and when it fails.
- Team is actively improving. Retrospectives are driving improvement.

RELEASE MANAGEMENT

1.
- Cycle time is extremely long
- Deployment is mostly manual, very complex and requires high level of coordination between stakeholders on the day of the release
- Separate build artefacts are created for each environment.
- Pace and delivery of work is inconsistent throughout the release

2.
- Releases are planned to a regular cycle and features are tracked and managed to the regular time frame.
- Planned release dates are seldom moved out.
- The process for handing the release to operations and support teams is in place.
- Standard build artefacts are defined and produced (release notes, binaries, configurations, test coverage reports...)

3.
- Releases and deployments are mostly automated (might require manual configuration and coordination between teams).
- The confidence level for the release is high.

4.
- Release notes are automatically generated.
- Releases are not reliant on the knowledge of the key individuals.
- Release metrics and trends are tracked
- Operations teams collaborate closely with the delivery team and contribute to minimize the risk of failure in production.

5.
- Releases can be deployed at any time and are fully automated
- The product owner is able to determine when release goes into production
- The team controls the release
- Releases are Routine. No roll backs, always roll forward.
- Operations team is part of the product/program team
- Dark launching is enabled

QUALITY ASSURANCE

1.
- Testing is manual.
- There is a high amount of rework due to testing late in the lifecycle.
- Non-functional testing is not considered upfront.
- Test cases are designed based only on the requirements document, mainly by testing team
- Absence of unit/integration testing.
- There is very little reuse of Quality Assurance assets.

2.
- Regression test packs are partially automated.
- Regression packs run frequently but on an ad hoc basis.
- Test cases designed by the team, with testing, development and business analysts working together

3.
- Regression test packs are fully automated.
- Regression test packs are run as part of the build.
- 50% coverage automated testing
- Automated smoke testing is in place
- Automated performance testing is in place
- Test cases designed by the team as part of the iteration.
- Non-functional requirements are defined and measured
- Unit tests are created using TDD

4.
- Automated infrastructure testing is in place
- Automated security testing is in place
- Tests are automated as much as is practical.
- Quality metrics and trends are tracked.
- Anyone can execute the tests.

5.
- Active process is in place to understand root cause and respond to it
- Test artefacts are treated with the same importance as code and continually refactored and maintained
- Performance testing metrics are defined early in the process
- Tests drive release decisions

ENVIRONMENTS

1.
- Data migrations un-versioned and performed manually
- Test data is not controllable.
- Environments are heavily governed
- Environments not refreshed regularly
- Test environments and test data are frequently unavailable
- Environments are provisioned manually.
- Key man dependencies

2.
- Changes to databases done with automated scripts versioned with application.
- A separate test environment is available.
- Test environment is shared with other teams.
- An external group maintains the environments.
- Test environments are available during specific dates/time slots
- Ability to deploy basic virtual environments
- Team has a dedicated development environment

3.
- Database changes are performed automatically as part of deployment process.
- Test environments are readily available and can be reproduced with manual work and coordination between operational teams.
- Access to a production-like environment is restricted to a small group of closely-related application teams, and delays at this level are unusual
- Some virtual environments have interfaces available 'stubs' to run end-to-end testing
- Environment provisioning partially automated

4.
- Database upgrades and rollbacks are tested with every deployment.
- Database performance is monitored and optimized
- Minimal manual work is required to replicate and configure environments but this can be completed within hours.
- Multiple test environments are readily available for the exclusive use of the team, including a production-like environment that allows a reasonable level of non-functional and cross-system integration testing and reliable acceptance testing.
- Dev workstations are easily configured and can be built in an automated manner.
- Virtual environments have all interfaces available 'stubs' to run end-to-end testing

5.
- Data and environments are treated as code, versioned and frequently re-loaded
- Release to release feedback loop of database performance and deployment process is in place.
- Proactive monitoring is in place with shortest possible time to resolve failure.
- IP is built into the process, including automated deployment, provisioning, configurations and data. Integral part of the code build pipeline
- Environments are managed by the team, the team able to create complete simulated environment on demand
- All environments are managed effectively.
- Provisioning is fully automated.
- Main dependencies are simulated across the board
- Virtualization is used if applicable.

FEATURE TEAMS

1.
- The team doesn't have all cross-functional and cross-component knowledge and skills to complete an end-to-end customer feature.
- The team members are not dedicated to the team.
- There is a low level of collaboration between team members.
- There are multiple handoffs and dependencies on the external teams and stakeholders.

2.
- The key skills are part of the team. This includes cross-functional and cross-component skills.
- Handoffs are identified.
- Triad is established across three functions, quality, product and technical. Triad is committed to execute a strategy together that gets specific, intended results.

3.
- The team members are dedicated to the team.
- Cross-functional and cross-component skills are within the team, the team works on a complete feature, across all components and disciplines (analysis, programming, testing, ...).
- New members are efficiently integrated into feature team.
- Triad is fully-functioning and not dependant on external validation and external authority).
- Goals are clearly prioritized and timeously achieved.

4.
- T-shaped skills emerge (team member can perform more than one function).
- The team is self-organizing. The team is self-managed and has necessary autonomy to modify the design of their team and/or aspects of the organisational context in which they operate
- Metrics are in place to ensure constant improvement.

5.
- All decision-making is internal to the team, who can make decisions on how to deliver artefacts, and as to quality measures applied to artefacts.
- T-shaped skills profile is a majority.
- Team is responsible for all aspects of feature delivery, from requirement requisition to production.
- Continuous improvement is in place, driven by retrospectives within the team and across the teams.