

Software Process and Architecture

An architectural perspective on
the Software Development
Lifecycle

Murray Knox

Fortress IT Ltd

murrayk@fortressit.com 

Introduction

- The need for process
- Types of Process
- Process selection and customization
- Process Tips
- Enterprise Architecture and SDLC

Need for Process

- Repeatable development
- Predictability
- Risk management
- Reduce chance of failure
- Improve quality

Cause of Failure

- From the Standish Group Chaos Report:
 - ◆ Lack of user involvement
 - ◆ Incomplete requirements
 - ◆ Unrealistic expectations
 - ◆ Changing requirements and specs.
 - ◆ Lack of, or poor planning
 - ◆ Lack of executive support
 - ◆ Lack of resources
 - ◆ Unclear objectives
 - ◆ Unrealistic time frames
 - ◆ New technology problems

Process Requirements

- Easy to understand
- Adaptable to changing situations
- Provide concrete guidance
- Suited to the task at hand
- Minimal overhead
- Must support project management
- Should allow measurement to improve process

Types of Process

- Ad hoc/Informal
- Planned
- Agile
- Other classifications:
 - ◆ heavyweight/lightweight
 - ◆ Formal/informal
 - ◆ Disciplined/undisciplined

Ad Hoc

- No Documented steps
 - ◆ Developers left to their own ingenuity
- Planning is difficult, if not impossible
- Progress measurement is difficult
- No risk management
- Most organisations start this way

Planned

- Up-front planning and design
- Development doesn't start until requirements are understood
- Can suffer from analysis paralysis
- Engineering based approach
- Explicit requirement capture
- Can use mixture of abilities
 - ◆ No process can substitute for good developers

Agile

- Short construction iterations
- Emphasizes communication and tacit knowledge
 - ◆ Forgoes planning and documentation
 - ◆ Assumptions and miscommunication cause problems
- Self-organizing teams
- Requires skilled and disciplined developers

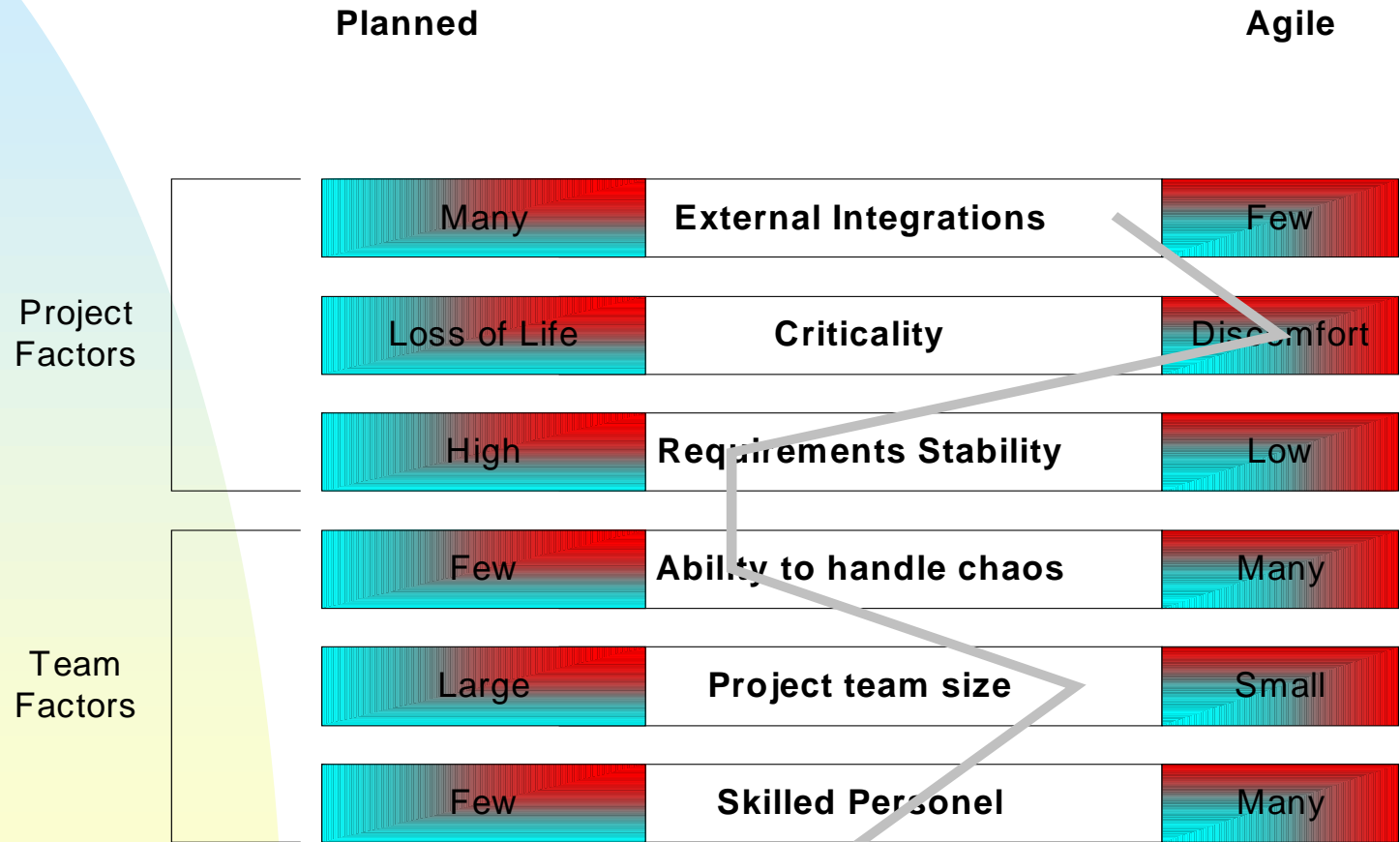
Agile Manifesto

- Prefer
 - ◆ *Individuals and Interactions* over processes and tools
 - ◆ *Working Software* over comprehensive documentation
 - ◆ *Customer Collaboration* over contract negotiation
 - ◆ *Responding to Change* over following a plan

Risk-Based Process Design

- No process is perfect for every situation
- Every project is different and has a different risk profile
- Use project risks to determine appropriate process framework

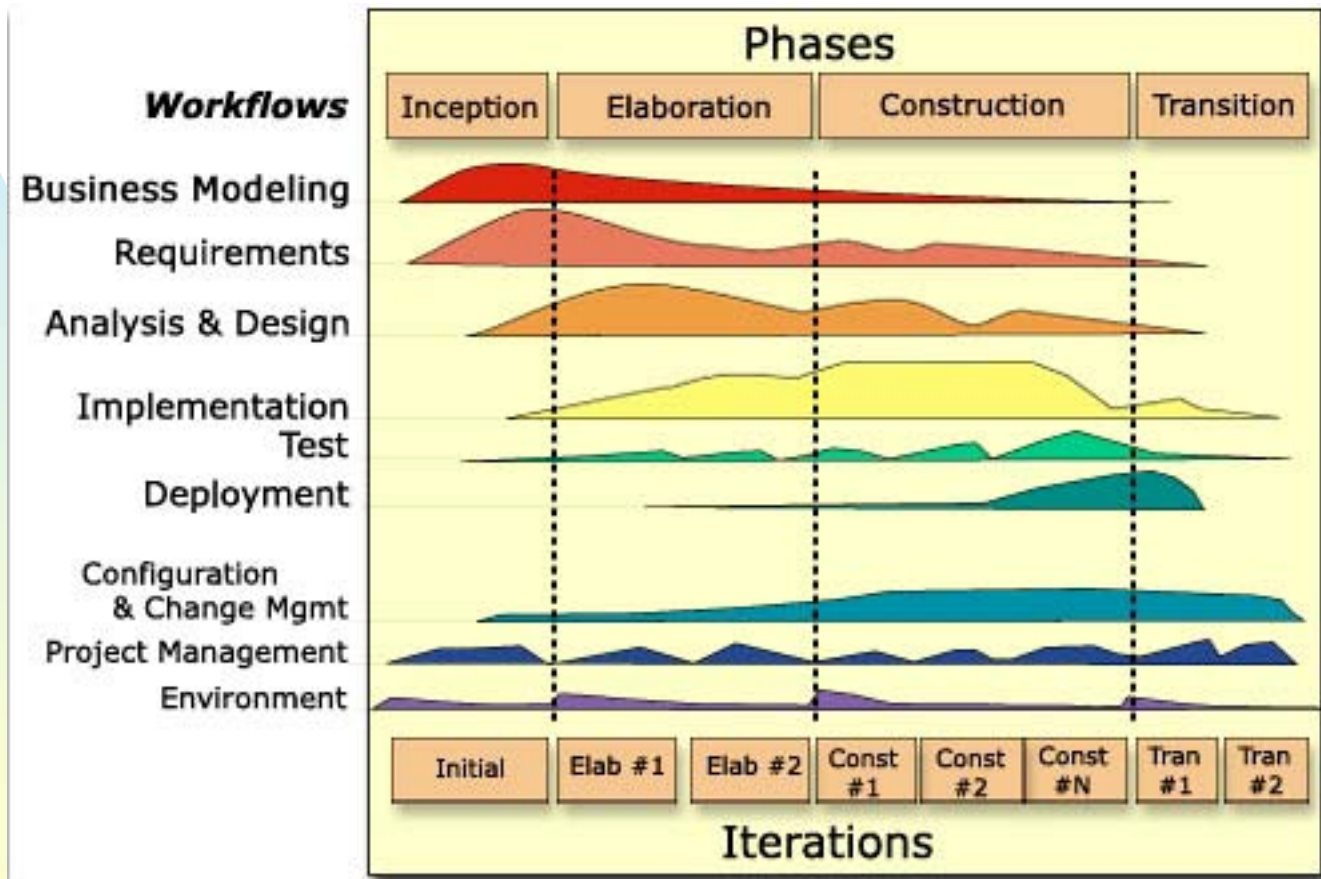
Risks



Iterative Development

- Develop over multiple iterations
 - ◆ cf. Waterfall - 1 iteration through each phase in turn
 - ◆ Waterfall is still appropriate in a few cases
- Reduces risk
- Can improve productivity
 - ◆ Has an overhead in each iteration
- No need to release every iteration
- First iterations should concentrate on building a vertical slice of the application

Project Phases



Project Phases

- Initiation
 - ◆ Project start
- Inception
 - ◆ Establish goals and scope
- Elaboration
 - ◆ Determine what to build
 - ◆ How to build it (architecture)
- Construction
 - ◆ Build and test
- Transition
 - ◆ Deploy to production

Communication

- Good communication between users and developers is essential
- XP mandates having a user representative as part of the development team
- FDD has domain experts on call
- Consider locating the team with the users/domain experts
- Don't restrict access to users/experts

Progress Management

- Measure progress against the plan
- Provide meaningful metrics to management
 - ◆ We've implemented 420 classes 🙌
 - ◆ We've implemented 33 business features 🙌
 - ◆ Closed off 42 defects
- Provides useful information for process improvement

Documentation

- Documentation should serve a purpose
- Simple and concise
- Accessible
 - ◆ project intranet, wiki, posters on the wall
 - ◆ Searchable
- Updated and maintained with the project
- Code is no substitute for good documentation
 - ◆ No such thing as self-documenting code

Essential Documentation

- Scope
- Glossary
- Requirements
 - ◆ Business Rules
 - ◆ Non-functional requirements
- Software Architecture
- Domain model
- Completion criteria

Inception Phase

- Project start up tasks
- Define Scope
- Identify Project Risks
 - ◆ Risk mitigation plan
 - ◆ Determine appropriate process
- Establish project team
- Initial plan

Project Scope

- Define what the purpose of the project is
 - ◆ Objectives
 - ◆ Goals
 - ◆ Constraints
- Without a properly defined scope there is a high risk of failure:
 - ◆ Personnel won't understand what they're trying to achieve
 - ◆ May build the wrong system

Elaboration Phase

- Identify requirements
- Identify key non-functional requirements
- Build Glossary
 - ◆ Build and define common language
 - ◆ Use the customers terms
- Perform Domain Modeling
- Proof of Concept for risky modules
- Establish base level architecture
- Build initial project plan and estimate

Requirements Management

- Starts at Inception and continues until final acceptance of system
- Requires good communication between developers and users
 - ◆ Business Analysts can aid communication and rationalization
- Iterative analysis
 - ◆ Implement simple requirement and enhance as users get a better understanding
- Many methods of capturing requirements:
 - ◆ Use cases
 - ◆ FDD Business Feature Lists
 - ◆ User Stories

Use Cases

- Useful for communicating with the user
- Not so useful for planning or development
- Not sufficient to define design
- UC descriptions 1-3 pages long not 40-50
- Should not contain design or implementation details
 - ◆ Screen shots, database schema etc.
- Use case diagram is useful for envisaging system

UI Prototyping

- Helps users to visualize and think about the system
- Should be done with the same technology as the final system
 - ◆ Visio is the worst UI prototyping tool
- Don't get hung up on L&F and touchy feely stuff
- Ensure users don't think the prototype is the system

Domain Modeling

- Capture the business domain model
 - ◆ Not the application specific model
 - ◆ Not solution model
- Group activity
 - ◆ Domain experts
 - ◆ Modeling experts
 - ◆ Developers

Business Features

- RUP and FDD have Business Feature Lists
- Identify user valued business functions
- FDD Feature List is 1st class requirements and planning artefact
 - ◆ Defined using template:
 - ✦ <action> <result> <object>

Non-Functional Requirements

- Performance, Scalability, security etc.
- Quality Attributes: Maintainability, flexibility etc.
- Essential input to architecture and testing
- Define Strawman NFR and have the customer validate them

Architecture

- All systems have an architecture
- Make it explicit
 - ◆ Others need to understand your decisions
 - ◆ Security, performance, scalability can not be added on later
- Architecture is integral to risk minimization
- The appropriate degree of architecture can be determined via the project risk analysis
- Is not Big Design Up Front (BDUF)
- XP doesn't have an architect but does have architecture (Metaphor)

Architecture

- Should be concise and accessible
 - ◆ No need for 150 page tomes
- Updated frequently
- Descriptive where possible. Prescriptive where necessary
- Include developers in the architecture development
- Be pragmatic
 - ◆ Don't sacrifice success for ideals and purity
- Review and validate early
- Multiple views: Logical, Physical, Data...

Architecture Validation

- Should be done early in the lifecycle
 - ◆ Can be done later
- Should ensure the architecture will satisfy both functional and non-functional requirements
- Use traceability to ensure all requirements are considered
- Can be as simple as a 1/2 hour peer review
- Can be as complex as a 5 day Architecture Tradeoff Analysis Method (ATAM) review

Planning

- Simple plan based around architecture
 - ◆ Identify the key activities and milestones
- Don't plan down to the micro detail
- Revise plan when needed
 - ◆ After every iteration
- User/Customer should be involved in prioritising features
- Plan for consistent and coherent feature releases

Estimation

- Provides customer with a view of what they are getting and when
- Should be done by the people doing the work
- Should be done prior to giving delivery date
- Use at least 2 estimation methods and validate
- Revise estimates as new knowledge is uncovered

Construction Phase

- Design and construct application
 - ◆ Detailed requirements analysis
- Build and Deployment
- Test
- Inspection

Design

- Just In Time
 - ◆ Prototype tricky functions
- Group design sessions with a whiteboard
- Don't design everything. Only what is tricky or special
- Design for testing
- Review prior to implementation
- Update domain model as needed

Simplicity

- Keep it as simple as it needs to be but no simpler
- XP promotes “You’re Not Going To Need It”
 - ◆ As simple as possible to satisfy the requirements
 - ◆ Excludes allowing for future enhancements
 - ✦ Even anticipated requirements
- The simplest designs are not necessarily the most maintainable
 - ◆ Layers, MVC etc add complexity whilst increasing maintainability

Testing

- Testing is useful but not the be all and end all of verification
- Not feasible to provide 100% test coverage
- Aerospace systems: DO178/B has less stringent testing requirements than DO187/A
 - ◆ DO178/B systems have no more defects than DO178/A systems

Unit Testing

- Developer tool
- Average defect detection rate 24%
- Can require more effort to develop than the code to be tested
- How do you know the test is correct?
- Automate - Integrate into build system
- Don't test everything
 - ◆ Test non-trivial classes and methods
 - ◆ When a bug is reported write a test for it

Functional Testing

- Test the system against the requirements
- Average defect detection rate: 35%
- Functional tests should be written from the requirements
- Automate if possible but not at the expense of productivity

Integration Testing

- Ensure the system works with other systems
- No surprises
- Average defect detection rate: 45%
- Should be performed in environment as close to production as possible
- For client apps test with multiple configurations

Performance Testing

- Different aspects
 - ◆ Performance
 - ◆ Load
 - ◆ Stress
- Should be performed regularly
- Requires concrete metrics to test against
- Test in similar environment to production
- Micro-benchmarks as part of unit testing to uncover poor algorithms

Inspection Definition

- Definition:
 - ◆ ‘...a formal evaluation technique in which software requirements, design, or code are examined in detail by a person or group other than the author to detect faults, violations of development standards, and other problems...’

-ANSI/IEEE Std. 729-1983,
IEEE Standard Glossary of Software Engineering Terminology

Design/Code Inspection

- Perform as key part of construction phase
- Average defect detection rate: 55%
(design) 60% (code)
- Use static analysis tools where possible
 - ◆ 30% of defects currently detectable
 - ◆ New tools may detect up to 70%
- Focus on errors and quality rather than style

Design/Code Inspection

- Takes time and effort
 - ◆ Rewarded by substantial reduction in defects
- Can be seen as criticism
 - ◆ Be impartial and non-personal
- Doesn't scale well to large code bases
 - ◆ Review as you go
 - ◆ Automate

Build and Deployment

- Automated build and deployment tools
 - ◆ Integrate testing with build system
- Use a dedicated build server
- Deployment can be difficult for complex systems
 - ◆ Deploy early and often
- Build from fresh source tree from SCM

What to Measure

- Number of requirements implemented
 - ◆ Features/ user stories/ use cases
- Number of defects raised
- Number of defects closed
- Number of defective/changed requirements

Progress Report Sample

Implementation (MD)								
Business Activities	Total Features	Not Started	In Progress	Behind Schedule	Completed	Inactive	% Completed	Completion Date
Authorisation	15	0	0	13	2	2	51	Mar 1999
Change Current Account for OD Line	12	12	0	0	0	0	0	Mar 1999
Establish Disbursement Details	35	0	0	4	31	1	91	Mar 1999
Establish Implementation Instruction	27	1	0	12	14	1	70	Mar 1999
Establish Line Implementation Details	21	0	0	4	17	0	89	Mar 1999
Establish a Loans, HP, Leasing Loans, BlockDiskcounting Line Impl.	7	0	0	0	7	0	100	Mar 1999
Establish a NIF RL Disbursement	8	6	0	0	2	1	25	Mar 1999
Establish a Trade or Gtee Line Implementation	9	0	0	0	9	0	100	Mar 1999
Establish an FX Line Implementation	5	0	0	0	5	0	100	Mar 1999
Establish an OD Line Implementation	13	0	0	0	13	0	100	Mar 1999
New Feature	3	0	0	0	3	0	100	Ongoing
Subject Area Total	155	19	0	33	103	5	74	
Progress Summary for Problem Domain								
	Total Features	Not Started	In Progress	Behind Schedule	Completed	Inactive	% Completed	
	1013	23	0	33	957	100	96	

Transition

- Product release
 - ◆ Deploy to production environment
- Configuration and change management
- Project review

Deploy to Production

- Should have been practiced many times during construction
- Environment should be tested and stable
- Should be no surprises

Project Review

- Collect and analyse metrics
- Feedback to improve process
 - ◆ What worked
 - ◆ What didn't work
 - ◆ What could we do better
- XP -Project Retrospective

Enterprise Architecture

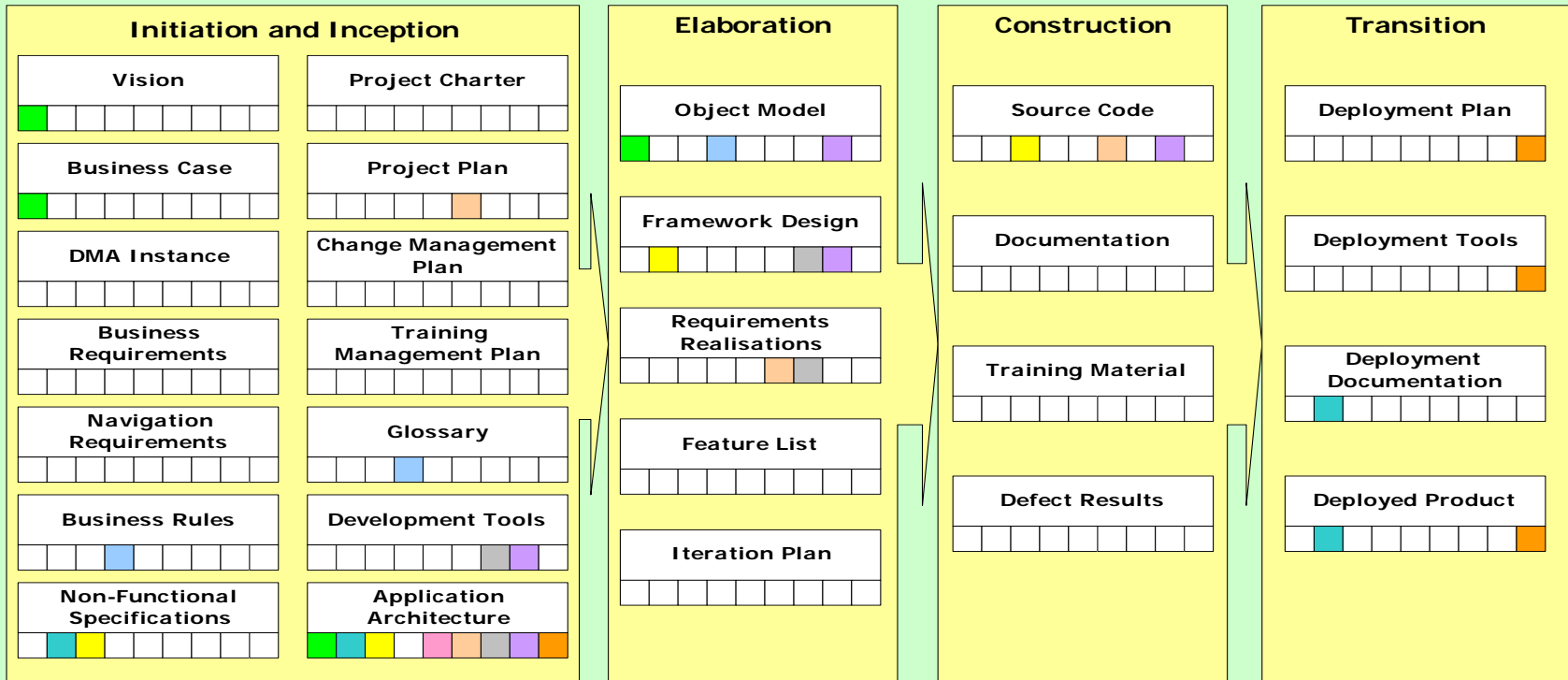
- Provides a road map to IT developments
- Aligns IT with the business
- Provides
 - ◆ Technology vision
 - ◆ Standards
 - ◆ Principles
 - ◆ Patterns
- Projects often proceed without reference to EA

EA Integration with the SDLC

Enterprise Information Architecture



Development Management Approach (DMA) Phases

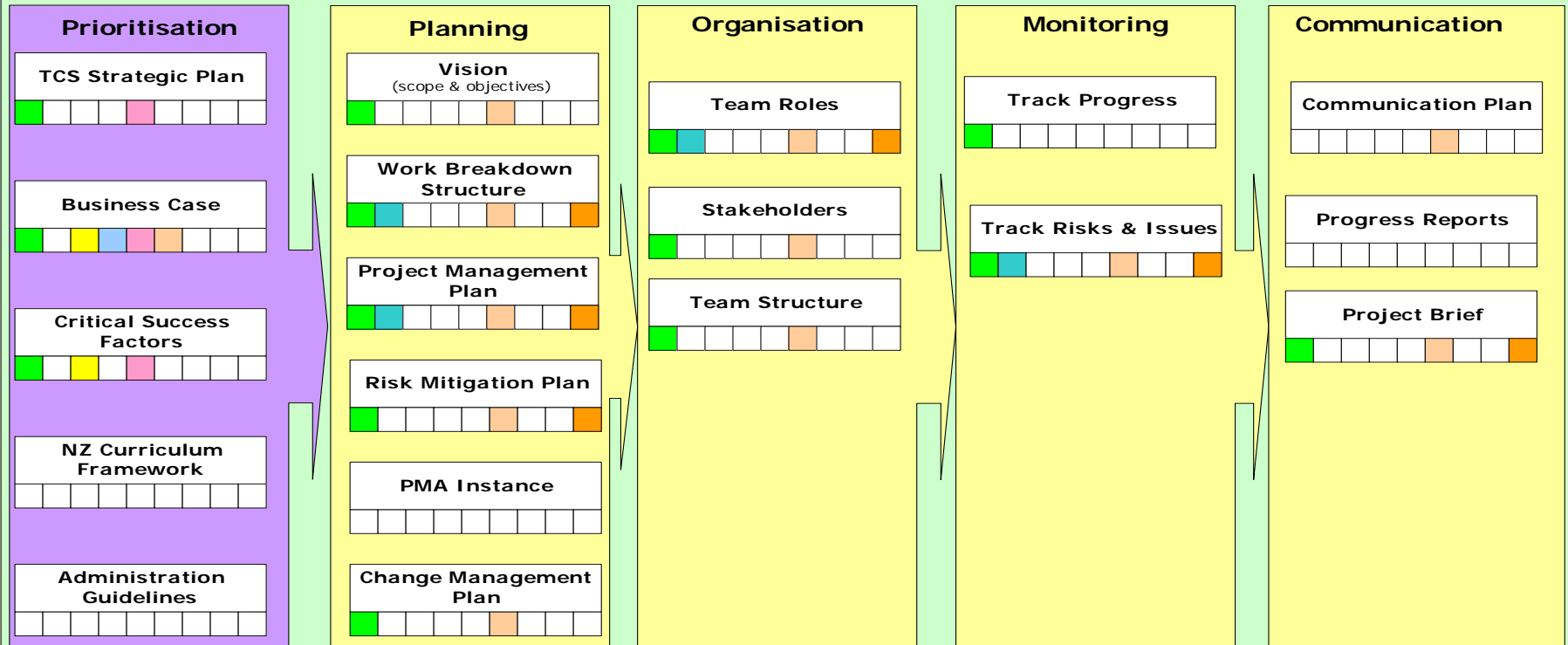


EA and Project Management

Enterprise Information Architecture



Project Management Approach (PMA) Phases



Aspects not Covered

- Development team
 - ◆ Training and mentoring
 - ◆ Organisation and skills
- Coding practices
 - ◆ Refactoring
 - ◆ Pair programming
- Configuration management
- Defect Tracking
- Tools

Summary

- No one size fits all process
- Determine appropriate process from project risks
- Start with a simple process and add tasks as needed
- Improve process from feedback
- No silver bullet and no substitute for good people

References

- www.featuredrivendevelopment.com
- www.xprogramming.com
- <http://www-306.ibm.com/software/rational/>
- <http://www.donald-firesmith.com/>
- http://www.projectsart.co.uk/docs/chaos_report.pdf
- <http://www.sei.cmu.edu/cmmi/>