UNIVERSITY OF
TECHNOLOGY SYDNEY

EVALUATING ARCHITECTURES

Software architecture

Tom McBride

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

In this lecture

- Why evaluate
- A general evaluation method
- Architecture Trade-off Analysis method (ATAM)
- Example of an ATAM review
- HAZOP analysis

WHY EVALUATE

Why do we evaluate anything

- We misunderstand, we make mistakes, we forget things
- A review of our work by one or more people finds these mistakes
- Find mistakes fast and fix them fast, while we have it all in our head
- More rigorous reviews find more mistakes, and usually cost more

Modes of interacting with the world

Knowledge based
Conscious

- Unskilled or occasional user
- Novel environment
- Slow
- Effortful
- Requires considerable feedback
- Causes of error
  - Overload
  - Manual variability
  - Lack of knowledge of modes of use
  - Lack of awareness of consequences

Skill-based
Automatic

- Skilled, regular user
- Familiar environment
- Fast
- Effortless
- Requires little feedback
- Causes or error
  - Strong habit intrusions
  - Frequently invoked rule used inappropriately
  - Situational changes that do not trigger the need to change habits

A continuum of actions and their potential errors

Knowledge-based

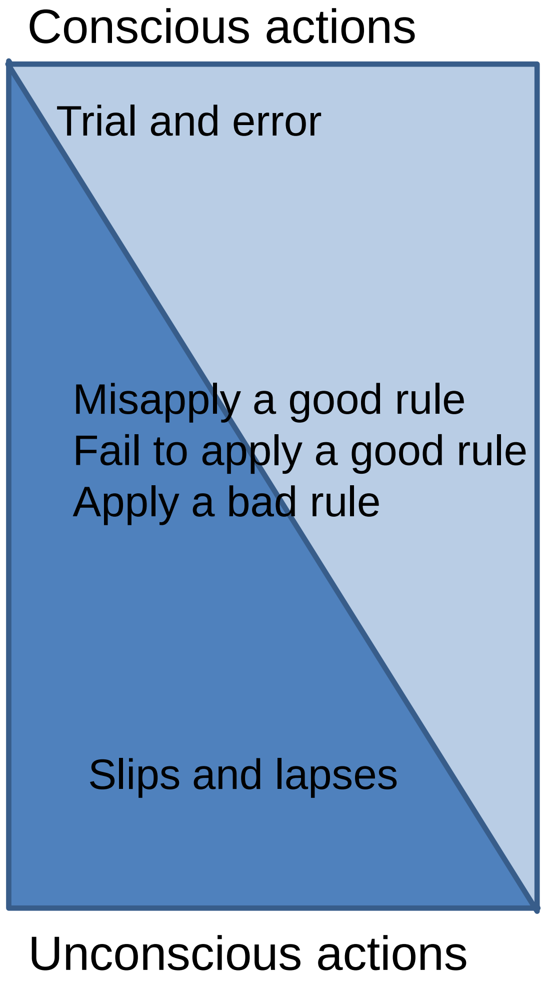
- Improvisation in unfamiliar environments
- No routines or rules available

Rule-based (Good or best practice)

- Protocol driven behaviour
- Process, procedure

Skill-based

- Automated routines
- Requires little conscious attention



Likely slips, lapses and mistakes

- Misconceptions
  - Architect and client have different concepts of the problem
- Rules
  - Inappropriate architecture pattern used
  - Good practice rules broken (cohesion and coupling guidelines)
- Slips and lapses
  - Incorrect diagram or diagram element
  - Something missing
  - Incomplete change
  - Inconsistency between different views

Goals of evaluation

- Will it work?
  - Usually check that something will achieve its objectives within its constraints at an acceptable risk level
  - Assumes that implementation will not alter the architecture's intention

Goals of evaluation

- Will it work?
- Will it fail? Can it fail? What would make it fail?
- Architectures last a long time so need to be robust in the face of their challenges
- A failure focussed review will require a lot of imagination to envisage scenarios in which the architecture might fail.
- The goal is to identify and acknowledge the possibility of failure and its circumstance, not to eliminate them. That decision comes later.

Summary

- To err is human
- There will be slips, lapses and mistakes
- Evaluation should find as many of them as possible
- Goal 1 – Will it work
- Goal 2 – Can it fail

TO ERR IS HUMAN; TO ADMIT IT, SUPERHUMAN.

Doug Larson
American Cartoonist

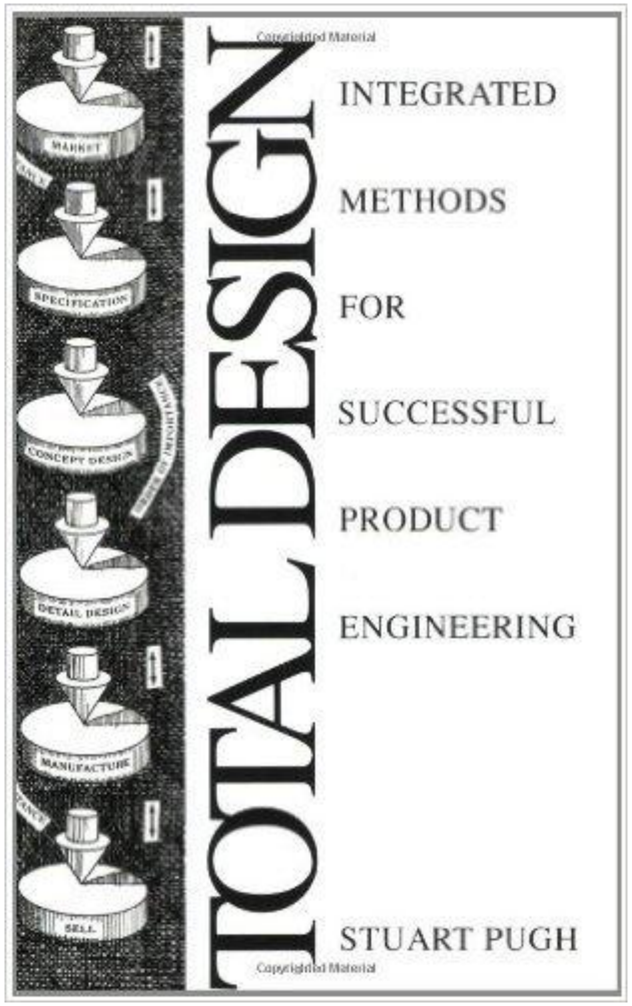
QUOTEHD.COM

Stuart Pugh’s systematic evaluation method

CONTROLLED CONVERGENCE

Is this architecture better than that one?

- Programs function properly or not.
- Architectures are satisfy their objectives better, or worse, than alternative architectures.
- Qualitative judgments are difficult to compare across alternatives.



Establish the criteria

- Start with the functional and quality requirements

	Alternatives		
	A	**B**	**C**
Functional requirements	0 - 5	0 - 5	0 - 5
Requirement 1			
Requirement 2			
Requirement 3			
Requirement 4			
Quality requirements	0 - 5	0 - 5	0 - 5
Reliability			
Performance			
Modifiability			
Security			
Usability			
Testability			

Rate this architecture

- Rate how well each requirement (or function or feature) is able to be implemented
- Rate how well each quality can be achieved

	Alternatives		
	A	**B**	**C**
Functional requirements	0 - 5	0 - 5	0 - 5
Requirement 1			
Requirement 2			
Requirement 3			
Requirement 4			
Quality requirements	0 - 5	0 - 5	0 - 5
Reliability			
Performance			
Modifiability			
Security			
Usability			
Testability			

Can the architecture be improved

- Should the architecture be improved to better satisfy a requirement
- Is there an alternative architecture that can better satisfy the requirements

	Alternatives		
	A	**B**	**C**
Functional requirements	0 - 5	0 - 5	0 - 5
Requirement 1			
Requirement 2			
Requirement 3			
Requirement 4			
Quality requirements	0 - 5	0 - 5	0 - 5
Reliability			
Performance			
Modifiability			
Security			
Usability			
Testability			

Example: Dragon boat regatta system

- Features
  - Event management
  - Race management
  - Crew management
  - Results
- Qualities
  - Highest importance (1) to lowest (6)

	Alternatives		
	MVC	**ADR**	C
Functional requirements	0 - 5	0 - 5	0 - 5
Event management	3	3	
Race management	3	3	
Crew management	3	3	
Results	3	3	
Quality requirements	0 - 5	0 - 5	0 - 5
Reliability (2)	4	4	
Performance (5)	2	3	
Modifiability (3)	4	5	
Security (4)	2	2	
Usability (1)	3	3	
Testability (6)	3	3	

UNIVERSITY OF TECHNOLOGY SYDNEY

A method for evaluating critical architectural decisions

ARCHITECTURE TRADE-OFF ANALYSIS METHOD

ATAM review

- Architecture trade-off analysis method (ATAM)
- Structured method for considering how the architecture handles a range of scenarios
- Requires scenarios that exercise critical parts of the system
- Designed to focus attention to those parts of the system that are central to achieving the system’s goals
- Similar to a Fagan inspection

ATAM review success factor

- ATAM reviews are highly dependent on the quality of the scenarios that drive them.
- Good scenarios identify
  - Circumstances of competing qualities
  - What is meant by “fast”, “secure”, “usable” etc.
  - Circumstances when failure would be critical
- This can result is a large number of scenarios – but they should be part of the requirements anyway

ATAM participants

- The evaluation team.
  - This group is external to the project whose architecture is being evaluated. It usually consists of three to five people. Each member of the team is assigned a number of specific roles to play during the evaluation.
- Project decision makers.
  - These people are empowered to speak for the development project or have the authority to mandate changes to it. They usually include the project manager, the customer and the architect.
- Architecture stakeholders.
  - Stakeholders have a vested interest in the architecture performing as advertised. They are the ones whose ability to do their jobs hinges on the architecture promoting modifiability, security, high reliability, or the like. Stakeholders include developers, testers, integrators, maintainers, performance engineers, users, builders of systems interacting with the one under consideration, and others.

ATAM phases

- Preparation
  - Prepare architecture description, scenarios
  - Assemble the evaluation team
  - Persuade the evaluation team to study the documentation
- Evaluation
  - Walk through each scenario
- Follow up
  - Deal with issues raised

ATAM - Evaluation

- Say how the evaluation will be conducted
- Present the business drivers
- Describe the architecture to an appropriate level
- Identify architectural approaches
- Prioritise scenarios
- Review the architecture against the prioritised scenarios
  - Architect describes how the architecture handles each scenario
  - Review team identify any issues with the architecture
- Present the results

ATAM outputs

- A concise presentation of the architecture.
- Articulation of the business goals.
- Quality requirements in terms of a collection of scenarios. Mapping of architectural decisions to quality requirements.
- A set of identified sensitivity and trade-off points.
- A set of risks and nonrisks.
- A set of risk themes.

Summary

- A structured review that considers the effect of the architecture on a selection of scenarios
- ATAM reviews are formal and structured
- ATAM reviews are similar to code inspections

UNIVERSITY OF TECHNOLOGY SYDNEY

EXAMPLE OF AN ATAM REVIEW

Example of ATAM Review

- Plug-in Architecture for Mobile Devices (PAMD) is an architecture framework1 for application interchange of data and control. It details the architecture, syntax, and semantic behaviour of applications in order to support application level data interchange. Within this framework, PAMD-aware applications can transparently extend their functionalities beyond what was conceived by their designers through the use of the PAMD plug-ins.
- PAMD provides:
  - A specification for application interchange of data and control
  - Common interfaces for PAMD-aware applications to access PAMD plug-ins
  - PAMD plug-in development framework

PAMD - Context

- **Customer and Users**
  - Not flexible: The customer has a vision and a specified product application.
- **Development organization**
  - Flexible: The developers have control over the team’s management structure and the development process.
- **Technical environment**
  - Not flexible: Few tools are available that help with development of Palm OS applications.
  - Palm OS does not provide extensive libraries that help developers build applications. E.g. lack of exception handling functionalities on OS level.
- **Architect’s experience**
  - Not flexible: The team is composed of architecturally immature developers. Developers’ previous programming experiences are based on the object-oriented concepts and methods whereas structured method is used to develop Palm OS applications.

PAMD – Functional requirements

1. Plug-ins shall provide services to other applications.
2. Plug-ins shall provide services to other plug-ins.
3. Application can use a plug-in as long as there is a match in data description.
4. Application shall not provide services to other application.
5. PAMD shall provide a list of compatible plug-ins to the applications.
6. PAMD shall not keep track of applications.
7. Allow plug-ins to provide services to PAMD aware applications and other plug-ins.
8. PAMD shall provide the service description standards for the plug-ins to abide by.
9. PAMD shall provide the data description standards for the applications to abide by.
10. PAMD shall provide API for developers to develop PAMD aware applications and PAMD compatible plug-ins.
11. PAMD shall provide a human readable description of the enabled/disables services.
12. PAMD shall provide a user interface for user to manage plug-ins.

PAMD – non-functional requirements

1. Plug-in shall be activated within n seconds.
2. Applications shall receive a plug-in list within n seconds.
3. There should not be any limitations to the number of plug-ins that can be installed, except by the limitation of the memory.
4. Both the architecture and the reference implementation of the system shall be compatible with Palm OS versions 3.1 and above. In addition, it shall be compatible with other PDAs, such as Handspring and Sony Clie.
5. Software developers shall be able to make an application PAMD aware within n number of days.
6. Software developers shall be able to transform a functional procedure into a plug-in within n number of days.

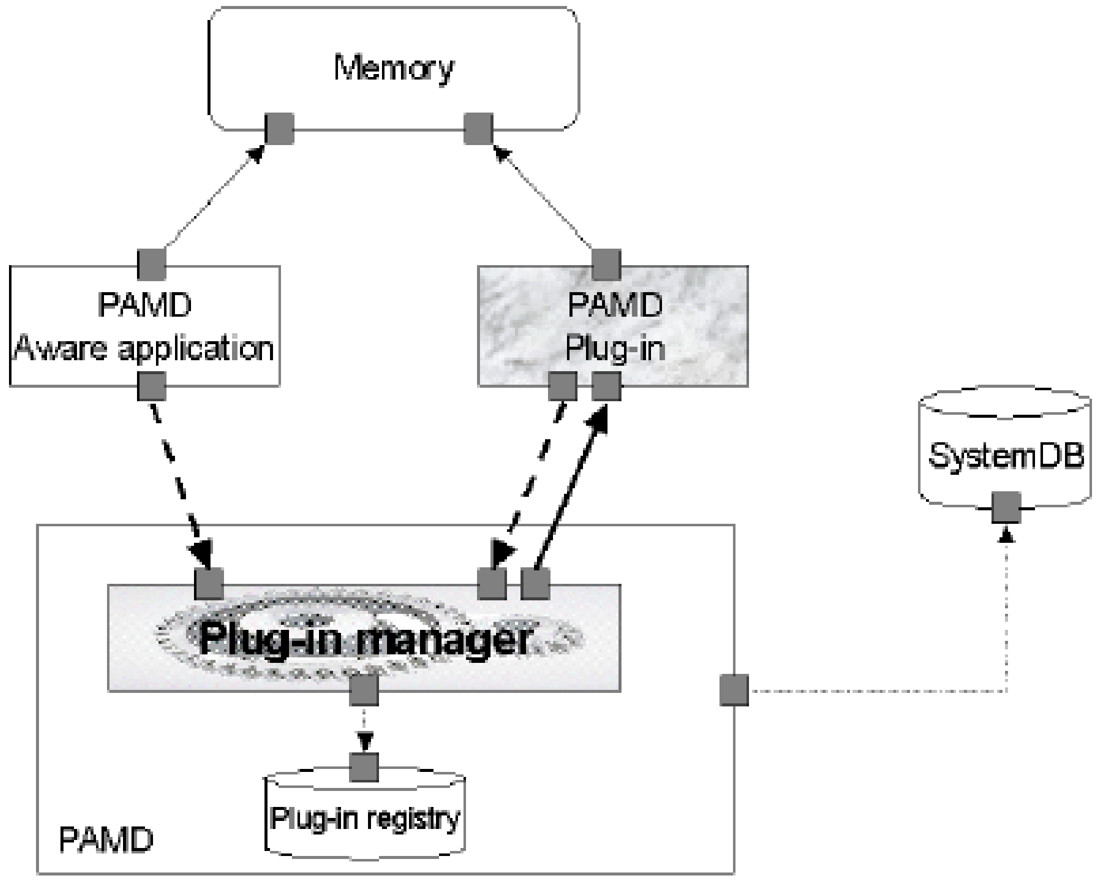
PAMD – Quality attributes

Quality Attribute	Description	Metric
Usability	The API provided by PAMD shall be easy to learn and use. Otherwise, a steep learning curve may discourage software developers from using PAMD	Days required to adapt an application to be PAMD aware Days required to create a plug-in
Performance	PAMD has to provide its services to Palm users in a reasonable amount of time. If users perceive that a PDA is sluggish when using PAMD they will stop using it.	Seconds to retrieve a list of compatible plug-ins
Availability	PAMD will be available for service every time a PDA is turned on, because it is the only way by which applications can communicate with plugins.	Mean time to failure (MTF)/MTF + Mean time to repair.
Portability	PAMD, PAMD-aware applications, and PAMD plug-ins shall work on Palm OS versions 3.1 to 4.x. More than one company offers products that utilize extended versions of the Palm OS. PAMD shall work on any of these, but PAMD shall not work on competing PDA operating systems like Windows CE.	Number of Palm OS versions supported. Number of Palm OS PDA brands supported.

PAMD – Architecture elements

- **Plug-in manager**
  - Coordinates communication between the plug-ins and applications
  - Interacts with multiple applications
  - Interacts with multiple plug-ins
- **Application**
  - Must be PAMD aware
  - Functionality:
    - request service from a plug-in
    - receive service from a plug-in
- **Plug-ins**
  - Conforms to the PAMD plug-in specification
  - Functionality:
    - provide service to a plug-in
    - provide service to an application
    - request service from a plug-in
    - receive service from a plug-in
- **Shared Memory**
  - provide storage for data exchange
- **Database**
  - stores plug-in information

PAMD – candidate architecture 1



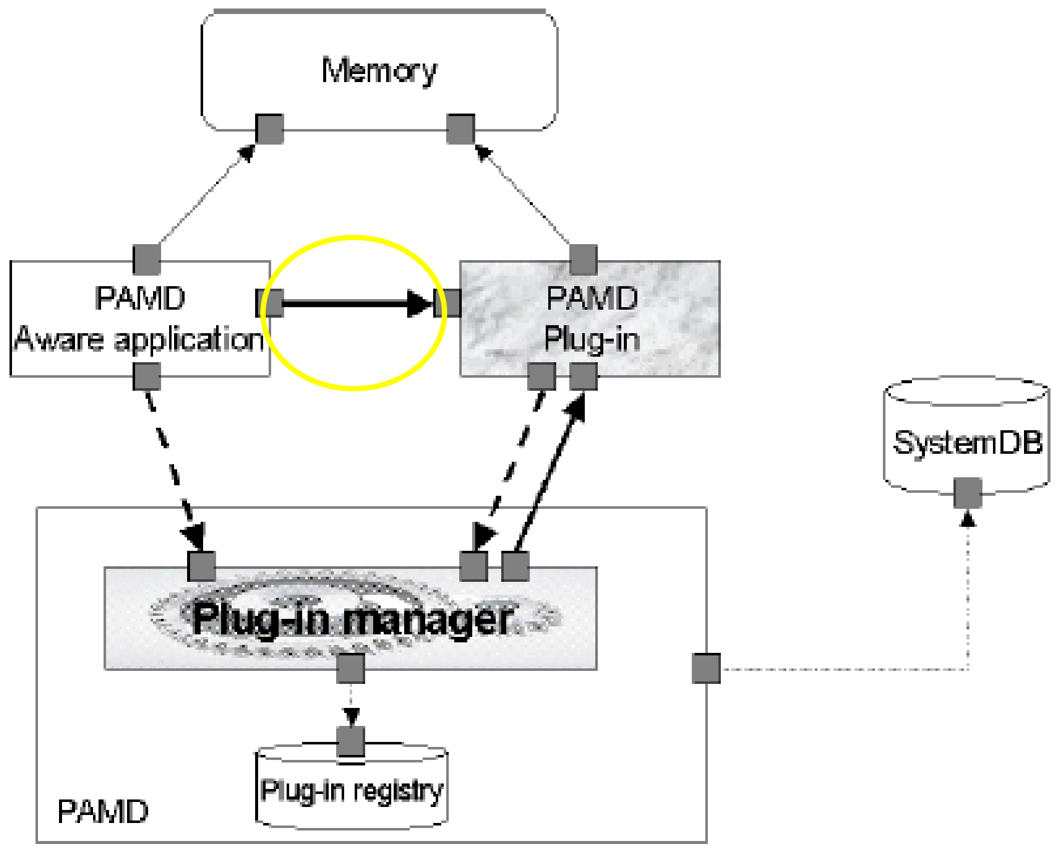
PAMD- Candidate architecture 1

- **General Architecture Description**
  - An application and a plug-in communicate with each other via the plug-in manager. The plug-in manager is responsible for maintaining the plug-in registry, which holds the information about the installed plug-ins. The registry stores information about the type of information that can be handled by any given plug-in. The PAMD system uses that information to judge the availability state of every registered plug-in. The type of information that can be processed by a given plugin serves as a selection criteria for matching plug-ins and applications.
- **Specific Architecture Tradeoff Points**
  - This candidate architecture uses an indirect invocation method to call plug-ins.
  - (+) Availability: Plug-in manager will handle errors for applications and plug-ins. It improves availability.
  - (+) Usability: An application developer doesn’t need to worry about error handling when executing plug-ins.
  - (–) Performance: The performance factor associated with the speed of plug-in execution may degrade.

PAMD – candidate architecture 1

- In this candidate architecture all plug-in calls are made through the plug-in manager, thus it does not allow an application to execute a plug-in that has been deleted. This is because the plug-in manager knows about the state of every plug-in. This plug-in existence checking can be handled transparently to the user as a service to all applications requesting execution of a plug-in.
- The Palm OS is not robust enough to gracefully handle fatal faults described in the above scenarios. This architectural approach increases the availability of PAMD by ensuring that the non-existent plug-ins are not executed.
- Usability is improved from the application developer’s perspective, because this type of exception handling is managed by PAMD.

PAMD – candidate architecture 2



PAMD – candidate architecture 2

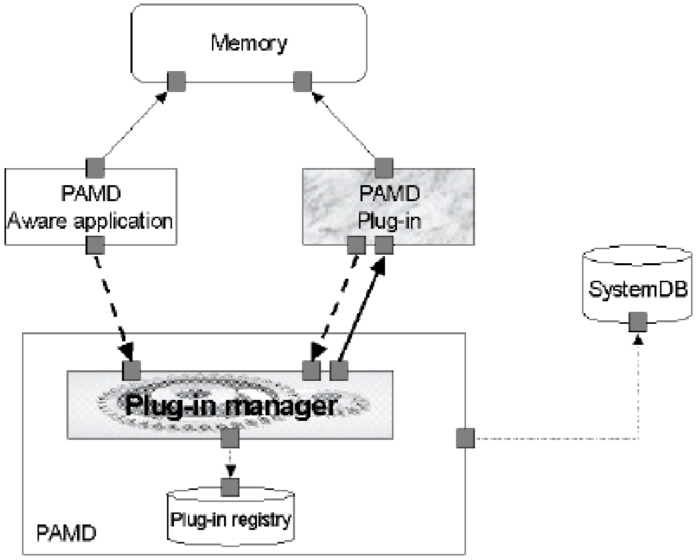
- **General Architecture Description**
  - The component composition of this candidate architecture is the same as presented above. The connector composition is different. In this case an application directly accesses a plug-in instead of invoking methods through the plug-in manager. The role of the plug-in manager is restricted to maintaining the plug-in registry and for passing a list of compatible plug-ins and their attributes to an application. The plug-in manager doesn't participate in the execution of a plug-in.
- **Specific Architecture Tradeoff Points**
  - This candidate architecture also uses an indirect invocation method to call plug-ins.
  - (-) Availability: Appropriate execution of a plug-in cannot be guaranteed. Every application is responsible to handling a possibility of an illegal execution of unavailable plug-in as outlined in candidate architecture one.
  - (+) Performance: Direct invocation of a plug-in from an application increases the launch performance of a plug-in, since there's no need to route the invocation of the plug-in through the plug-in manger.
  - (-) Usability: An application developer cares about error handling when executing plug-ins.

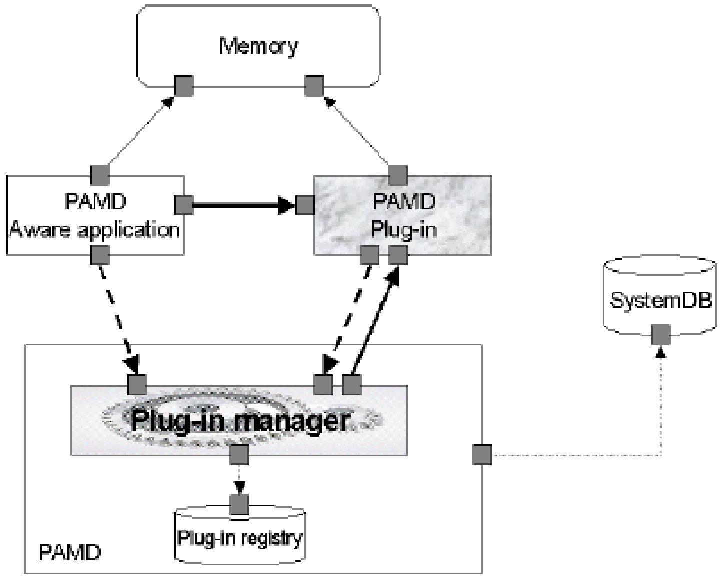
PAMD – candidate architecture 2

– Lack of transparent exception handling mechanism can negatively impact the availability of the system and decrease the usability for application developers and users. An application developer needs to spend more time reinventing the wheel by building customer exception handling mechanism for handling deleted plug-ins. Frequent system crashes reduce usability from value of the system from the end user's perspective.

PAMD – side by side

- Ca
  -
  -





- Candidate 2
  - Better performance
  - Worse availability, usability

Summary

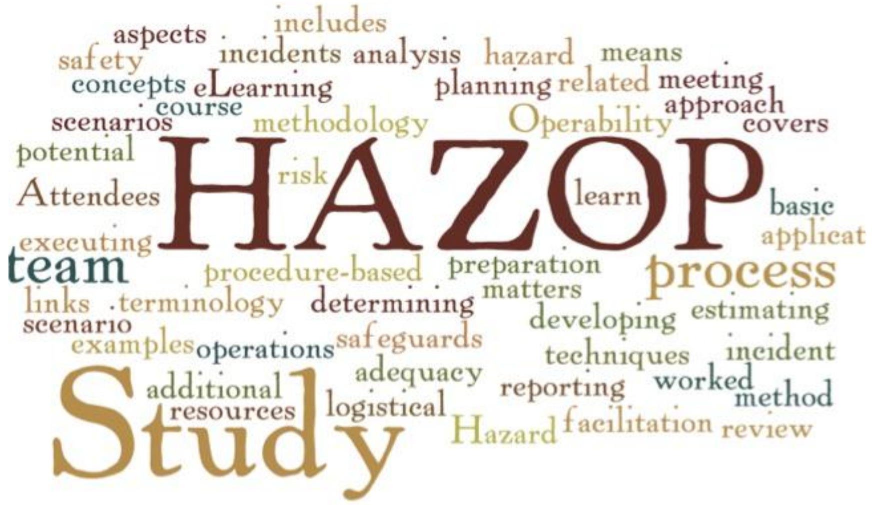
- Alternative methods for invoking a plug-in have architectural consequences
- Evaluate the alternative architectures by their effect on system non-functional characteristics
- Does not decide which is better – simply identifies the consequences of each architecture

UNIVERSITY OF TECHNOLOGY SYDNEY

HAZOP ANALYSIS

HAZOP analysis

- Hazard and Operability (HAZOP)
- UK Defence Standard 00-58
- It is a formal inspection of an architecture requiring formal documentation
- Checklist based
- Useful for life-critical real-time systems



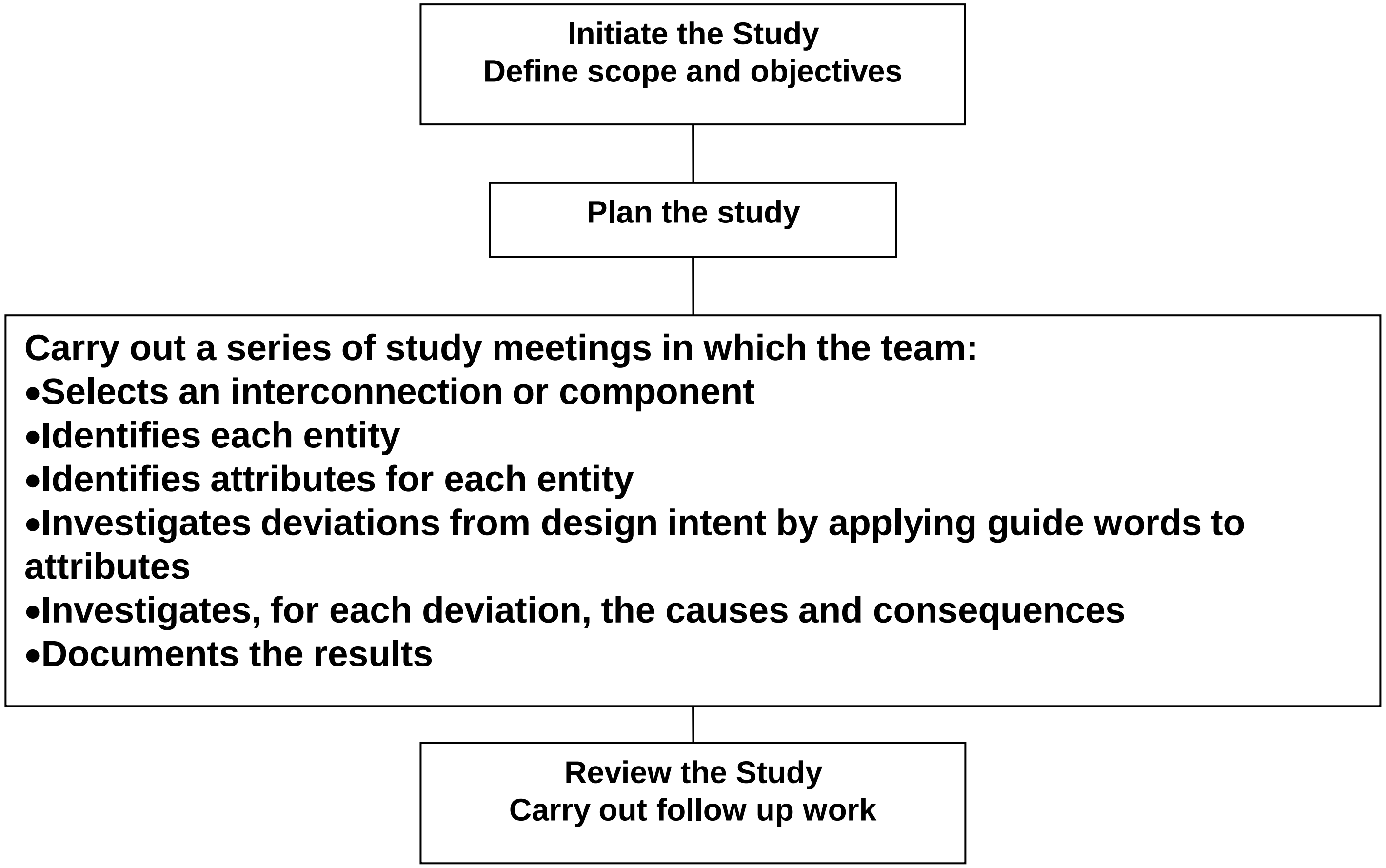
HAZOP Scope and applicability

1. This Standard is intended to be used by those who, in conforming to the requirements of Def Stan 00-56, choose to use a HAZOP Study as part of their method of hazard analysis. More generally, it is a self-contained guide to carrying out a HAZOP Study of any system. It is a detailed guide for those who need it, and a reference for experienced HAZOP practitioners.
2. Individual HAZOP Studies may be carried out at various stages of a system's life cycle. This Standard is applicable to HAZOP Studies whenever they are carried out.
3. It is often useful for the results of a HAZOP Study to be supported by the products of other Studies, such as fault trees. Further, a HAZOP Study may be preceded and followed by other aspects of safety analysis. Such additional work is covered by Def Stan 00-56.
4. HAZOP Studies are concerned with both hazard identification and safe operation of equipment. However, they do not cover the ability of staff to operate a system. These training and human factors issues are addressed in Def Stan 00-56 and Def Stan 00-25.
5. HAZOP Studies are a team activity. Their effectiveness relies on the management of the team, the knowledge of the team members, the interaction of the individuals and thus, to a large extent, on 'human factors' in the HAZOP process. A number of clauses in this Standard therefore address these aspects.
6. A HAZOP Study typically requires several HAZOP Study Meetings.

HAZOP general requirements

- A number of HAZOP Studies carried out over the system life cycle can contribute significantly to achieving the level of confidence required in a hazard identification process. Hazard identification is a process which may be applied at any stage from the feasibility study through to disposal, as described in Def Stan 00-56.
- A HAZOP Study can be carried out at any level of design representation. Given a suitable representation, a HAZOP Study can also be applied to a requirements expression.

HAZOP - Overview



Initiate the Study
Define scope and objectives

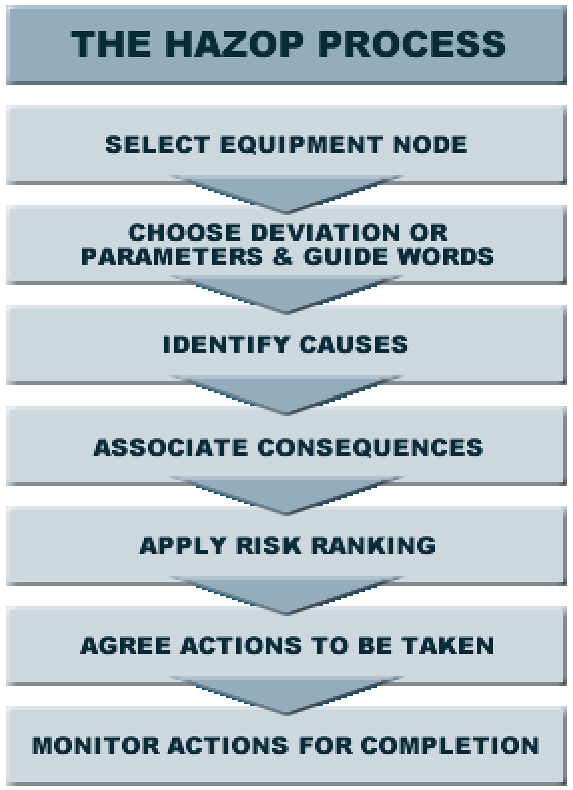
Plan the study

Carry out a series of study meetings in which the team:
- **Selects an interconnection or component**
- **Identifies each entity**
- **Identifies attributes for each entity**
- **Investigates deviations from design intent by applying guide words to attributes**
- **Investigates, for each deviation, the causes and consequences**
- **Documents the results**

Review the Study
Carry out follow up work

HAZOP – activity

- A rigorous and systematic procedure shall be used
- to investigate the possible deviations of attributes from their design intent.
- This is based on the use of ‘guide words’ which are the core of HAZOP Studies.
- A guide word is a word or phrase which expresses or defines a specific type of deviation.
- The role of the guide word is as a prompt, to focus the Study and elicit ideas and discussion and, thus, to maximise the chance of completeness.


THE HAZOP PROCESS
SELECT EQUIPMENT NODE
CHOOSE DEVIATION OR PARAMETERS & GUIDE WORDS
IDENTIFY CAUSES
ASSOCIATE CONSEQUENCES
APPLY RISK RANKING
AGREE ACTIONS TO BE TAKEN
MONITOR ACTIONS FOR COMPLETION

HAZOP – guide words

No	No data or control signal passed
More	More data is passed than intended
Less	Not used here because this is already covered by ‘part of’
As well as	Not used here because this is already covered by ‘more’
Part of	The data or control signals are incomplete
Reverse	Normally not relevant
Other than	The data signals are complete but incorrect
Early	The signal arrives too early with reference to clock time
Late	The signal arrives too late with reference to clock time
Before	The signal arrives earlier than intended within a sequence
After	The signal arrives later than intended within a sequence

Summary

- Hazop analysis is a formal inspection of an architecture requiring formal documentation
- Hazop analysis is checklist based
- Concentrates on the flow of information (or material)
- Each architectural element is tested against the checklist
- Rigorous, intended for life critical systems