SSW-540: Fundamentals of Software Engineering

Software Dependability and Reliability Engineering

Dr. Richard Ens
School of Systems and Enterprises

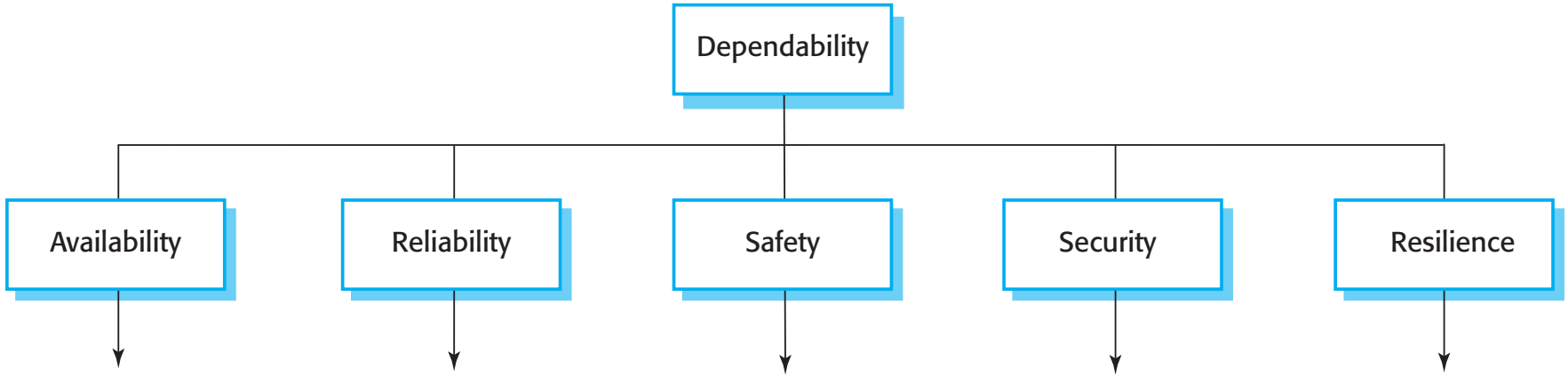
Python Pointers - RegEx

- In Unix, and in Python, regular expressions are one of the most powerful tools as your disposal.
- The library for regular expressions, **re**, is defined fully for Python at http://docshttps://docs.python.org/[23]/library/re.html.
  - Note my use of a regular expression in the URL above in order to point at both the library for Python 2(.7.11) as well as that for 3(.5.1). You should use only the 2 or the 3, not both!
- At this documentation page you will find
  - The complete list of special characters and sequences as well as the special sequences using the escape \ character.
- Bookmark that page! I've been using regular expressions for over 30 years and I'm still looking up the meaning of special characters and sequences.

Software system dependability importance

- Often for computer-based systems, the most important system property is the dependability of the system.
- The dependability of a system reflects the user's degree of trust that the system will operate as users expect and that it will not 'fail' in normal use.
- Dependability covers the related systems attributes of reliability, availability, safety and security. These are all inter-dependent.
    - System failures may have widespread effects with large numbers of people affected by the failure.
    - Systems that are not dependable and are unreliable, unsafe or insecure may be rejected by their users.
    - The costs of system failure may be very high if the failure leads to economic losses or physical damage.
    - Undependable systems may cause information loss with a high consequent recovery cost.

Principal dependability properties



The probability that the system will be up and running and able to deliver useful services to users.

The probability that the system will correctly deliver services as expected by users.

A judgment of how likely it is that the system will cause damage to people or its environment.

A judgment of how likely it is that the system can resist accidental or deliberate intrusions.

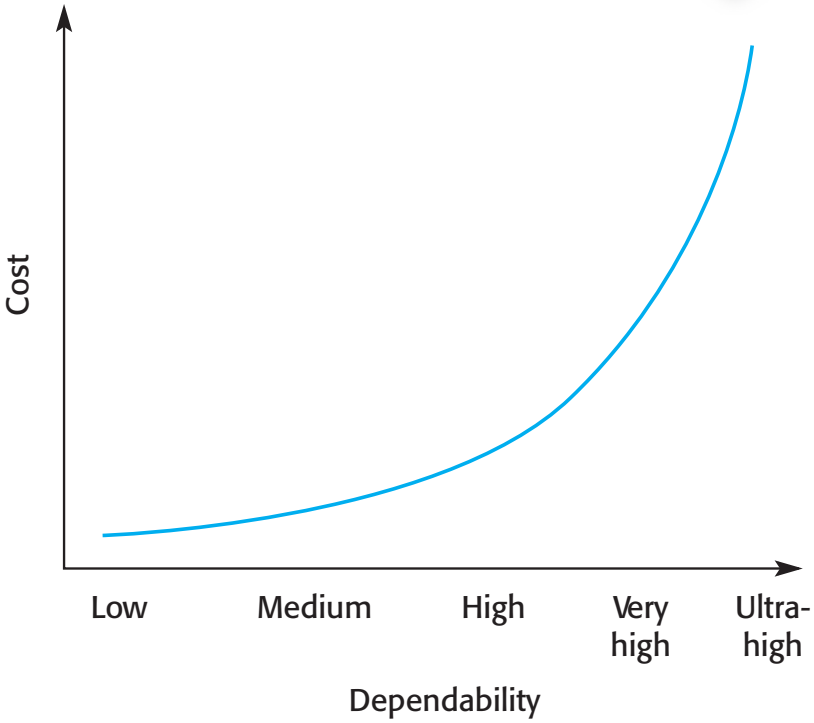
A judgment of how well a system can maintain safety and the continuity of its critical services in the presence of disruptive events such as equipment failure and cyberattacks.

Achieving dependability

- Avoid the introduction of accidental errors when developing the system.
- Design V & V (verification and validation) processes that are effective in discovering residual errors in the system.
- Design systems to be fault tolerant so that they can continue in operation when faults occur
- Design protection mechanisms that guard against external attacks.
- Configure the system correctly for its operating environment.
- Include system capabilities to recognize and resist cyber-attacks.
- Include recovery mechanisms to help restore normal system service after a failure.

Dependability costs

- Dependability costs tend to increase exponentially as increasing levels of dependability are required.
- There are two reasons for this
  - The use of more expensive development techniques and hardware that are required to achieve the higher levels of dependability.
  - The increased testing and system validation that is required to convince the system client and regulators that the required levels of dependability have been achieved.



Dependability economics

- Because of very high costs of dependability achievement, it may be more cost effective to accept untrustworthy systems and pay for failure costs.
- This will depend on social and political factors. A reputation for products that can't be trusted may lose future business.
- Depends on system type - for business systems in particular, modest levels of dependability may be adequate (but for social media to succeed…).
- In critical situations, software systems must be fault tolerant.
- Fault tolerance is required where there are high availability requirements or where system failure costs are very high (e.g., SpaceX shuttles).
- Fault tolerance means that the system can continue in operation in spite of software failure (e.g., self-driving autos).
- Even if the system has been proved to conform to its specification, it must also be fault tolerant as there may be specification errors or the validation may be incorrect.

Software redundancy and diversity

- Fault-tolerant systems architectures are used in situations where fault tolerance is essential. These architectures are generally all based on redundancy and diversity.
- Redundancy
  - Keep more than a single version of critical components so that if one fails then a backup is available.
- Diversity
  - Provide the same functionality in different ways in different components so that they will not fail in the same way.
- Redundant and diverse components should be independent so that they will not suffer from 'common-mode' failures
  - For example, components implemented in different programming languages means that a compiler fault will not affect all of them.

Process diversity and redundancy

- Process activities, such as validation, should not depend on a single approach, such as testing, to validate the system.
- Redundant and diverse process activities are important especially for verification and validation.
- Multiple, different process activities the complement each other and allow for cross-checking help to avoid process errors, which may lead to errors in the software.

Problems with redundancy and diversity

- Adding diversity and redundancy to a system increases the system complexity.
- This can increase the chances of error because of unanticipated interactions and dependencies between the redundant system components.
- Some engineers therefore advocate simplicity and extensive V & V as a more effective route to software dependability.
- Airbus FCS architecture is redundant/diverse; Boeing 777 FCS architecture has no software diversity.
- Useful software diversity is hard to achieve (we can try different programming languages, different design methods and tools, explicit specification of different algorithms)
  - But software teams tend to tackle problems in the same way.
  - A common specification may be wrong

Dependable process characteristics

- **Explicitly defined**
  - A process with a underlying defined process model, used to drive the software production process (e.g., a software development process).
  - Data is collected during the process that proves that the development team has followed the process as defined in the process model.
- **Repeatable**
  - A process that does not rely on individual interpretation and judgment.
  - The process can be repeated across projects and with different team members, irrespective of who is involved in the development.
- Additional dependable process attributes include
  - **Auditability**, **diversity**, **documentable**, r**obust** and s**tandardized**.

Some dependable process activities

- Requirements reviews
- Requirements management
- Formal specification, where a mathematical model of the software is created and analyzed.
- System modeling, where the software design is explicitly documented as a set of graphical models, and the links between the requirements and these models are documented.
- Design and program inspections
- Static analysis
- Test planning and management

(Some conflict with the general approach taken in agile development)

And dependable programming

Dependable programming guidelines

1. Limit the visibility of information in a program
2. Check all inputs for validity
3. Provide a handler for all exceptions
4. Minimize the use of error-prone constructs
5. Provide restart capabilities
6. Check array bounds
7. Include timeouts when calling external components
8. Name all constants that represent real-world values
9. Use architecture, design and code reviews
10. Minimize code complexity

Formal specification

- Formal methods are approaches to software development that are based on mathematical representation and analysis of software.
- Formal methods include

  Formal specification;

  Specification analysis and proof;

  Transformational development;

  Program verification.
- Formal methods significantly reduce some types of programming errors and can be cost-effective for dependable systems engineering.

Formal approaches

- Verification-based approaches

  Different representations of a software system such as a specification and a program implementing that specification are proved to be equivalent.

  This demonstrates the absence of implementation errors.

- Refinement-based approaches

  A representation of a system is systematically transformed into another, lower-level representation; e.g. a specification is transformed automatically into an implementation.

  This means that, if the transformation is correct, the representations are equivalent.

Use of formal methods

- The principal benefits of formal methods are in reducing the number of faults in systems.
- Consequently, their main area of applicability is in dependable systems engineering. There have been several successful projects where formal methods have been used in this area.
- In this area, the use of formal methods is most likely to be cost-effective because high system failure costs must be avoided.
- Formal methods for software specification and verification have been used successfully in aerospace, avionics and transportation domains.

Classes of error

- Specification and design errors and omissions.
  - Developing and analysing a formal model of the software may reveal errors and omissions in the software requirements. If the model is generated automatically or systematically from source code, analysis using model checking can find undesirable states that may occur such as deadlock in a concurrent system.
- Inconsistences between a specification and a program.
  - If a refinement method is used, mistakes made by developers that make the software inconsistent with the specification are avoided. Program proving discovers inconsistencies between a program and its specification.

Benefits of formal specification

- Developing a formal specification requires the system requirements to be analyzed in detail. This helps to detect problems, inconsistencies and incompleteness in the requirements.
- As the specification is expressed in a formal language, it can be automatically analyzed to discover inconsistencies and incompleteness.
- If you use a formal method such as the B method, you can transform the formal specification into a 'correct' program.*
- Program testing costs may be reduced if the program is formally verified against its specification.

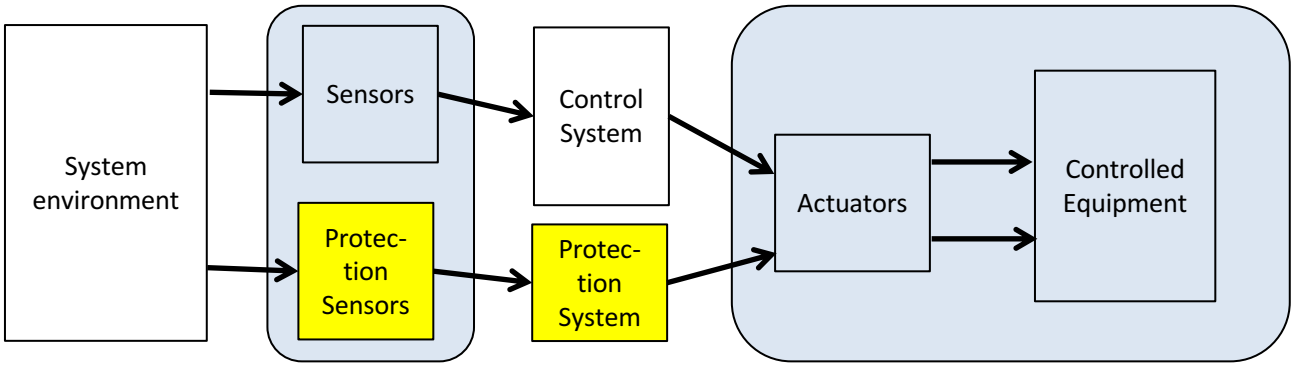
* See *The B-Method of Formal Specification* reference on Canvas

Acceptance of formal methods

- Formal methods have had limited impact on practical software development:
  - Problem owners cannot understand a formal specification and so cannot assess if it is an accurate representation of their requirements.
  - It is easy to assess the costs of developing a formal specification but harder to assess the benefits. Managers may therefore be unwilling to invest in formal methods.
  - Software engineers are unfamiliar with this approach and are therefore reluctant to propose the use of Formal Methods.
  - Formal methods are still hard to scale up to large systems.
  - Formal specification is not really compatible with agile development methods.

Software is not always reliable

- Failures are a usually a result of system errors that are derived from faults in the system.
- However, faults do not necessarily result in system errors.
  - The erroneous system state resulting from the fault may be transient and 'corrected' before an error arises.
  - The faulty code may never be executed.
- Errors do not necessarily lead to system failures.
  - The error can be corrected by built-in error detection and recovery.
  - The failure can be protected against by built-in protection facilities. These may, for example, protect system resources from system errors.



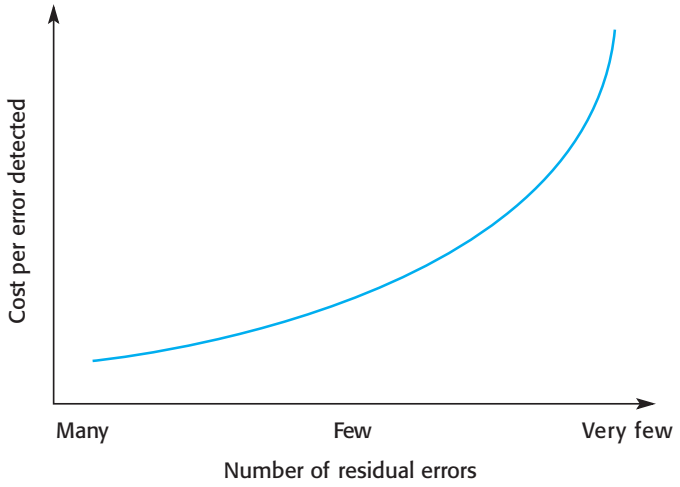
Protection System Architecture

Faults, errors and failures

Term	Description	Example
Human error or mistake	Human behavior that results in the introduction of faults into a system.	In the wilderness weather system, a programmer decided to compute the time for the next transmission by adding 1 hour to the current time. This works except when the transmission time is between 23.00 and midnight (midnight is 00.00 in the 24-hour clock).
System fault	A characteristic of a software system that can lead to a system error.	The fault is the inclusion of the code to add 1 hour to the time of the last transmission, without a check if the time is greater than or equal to 23.00.
System error	An erroneous system state that can lead to system behavior that is unexpected by system users.	The value of transmission time is set incorrectly (to 24.XX rather than 00.XX) when the faulty code is executed.
System failure	An event that occurs at some point in time when the system does not deliver a service as expected by its users.	No weather data is transmitted because the time is invalid.

Achieving reliability

- Fault avoidance
  - Development techniques are used that either minimize the possibility of mistakes or trap mistakes before they result in the introduction of system faults (e.g., ASSERT statements).
- Fault detection and removal
  - Verification and validation techniques are used that increase the probability of detecting and correcting errors before the system goes into service are used (e.g., reviews).
- Fault tolerance
  - Run-time techniques are used to ensure that system faults do not result in system errors and/or that system errors do not lead to system failures (e.g., bounding input/output).



Availability and reliability

- Reliability
  - The probability of failure-free system operation over **a specified time** in a given environment for a given purpose
- Availability
  - The probability that a system, at a point in time, will be operational and able to deliver the requested services
- Both of these attributes can be expressed quantitatively; e.g. availability of 0.999 means that the system is up and running for 99.9% of the time.
- While we formally define reliability with respect to a system specification, perceived reliability is more important in practice.
  - Usage of a system in an office environment is likely to be quite different from usage of the same system in a university environment.
  - The consequences of system failures affects the perception of reliability

Availability perception

- Availability is usually expressed as a percentage of the time that the system is available to deliver services e.g. 99.95%.
- However, this does not take into account:
  - The number of users affected by the service outage.
    - Loss of service in the middle of the night is less important for many systems than loss of service during peak usage periods.
  - The length of the outage.
    - The longer the outage, the more the disruption.
    - Several short outages are less likely to be disruptive than 1 long outage.
    - Long repair times are a particular problem.

Reliability in use

- Removing X% of the faults in a system will not necessarily improve the reliability by X%.
- Program defects may be in rarely executed sections of the code so they may never be encountered by users. Removing these does not affect the perceived reliability.
- Users adapt their behavior to avoid system features that may fail for them.
- A program with known faults may therefore still be perceived as reliable by its users.
- And in reality, we are never able to remove all of the faults in a complex system.

System reliability requirements

- Reliability is a measurable system attribute so non-functional reliability requirements may be specified quantitatively. These define the number of failures that are acceptable during normal use of the system or the time in which the system must be available.
- Functional reliability requirements define system and software functions that avoid, detect or tolerate faults in the software and so ensure that these faults do not lead to system failure.
- Software reliability requirements may also be included to cope with hardware failure or operator error.
- The more extensive the reliability requirements, the more time should be allocated for reviews, static analysis and dynamic testing.

Reliability metrics

- Reliability metrics are units of measurement of system reliability.
- System reliability is measured by counting the number of operational failures and, where appropriate, relating these to the demands made on the system and the time that the system has been operational.
- A long-term measurement program is required to assess the reliability of critical systems.
- Metrics
  - Probability of failure on demand (POFOD)– used for protection systems, measuring the probability that the system will fail when a service request is made.
  - Rate of occurrence of failures/Mean time to failure – used for systems that process a large number of similar requests in a short time (MTTF is the reciprocal of ROCOF)
  - Availability (AVAIL)– taking repair and restart times into account, availability measures the fraction of time the system in available for use. Very relevant for continuously running systems like communication and signalling systems.

Benefits of specifying reliability

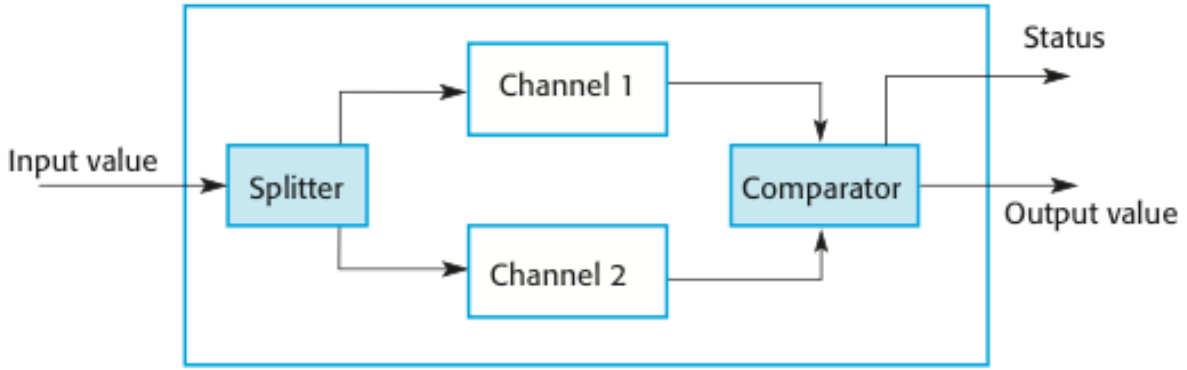
- The process of deciding the required level of the reliability helps to clarify what stakeholders really need.
  - Different types of failures may have different reliability requirements.
  - Different services may have different reliability needs.
- It provides a basis for assessing when to stop testing a system. You stop when the system has reached its required reliability level.
- It is a means of assessing different design strategies intended to improve the reliability of a system.
- If a regulator has to approve a system (e.g. all systems that are critical to flight safety on an aircraft are regulated), then evidence that a required reliability target has been met is important for system certification.

Functional reliability requirements

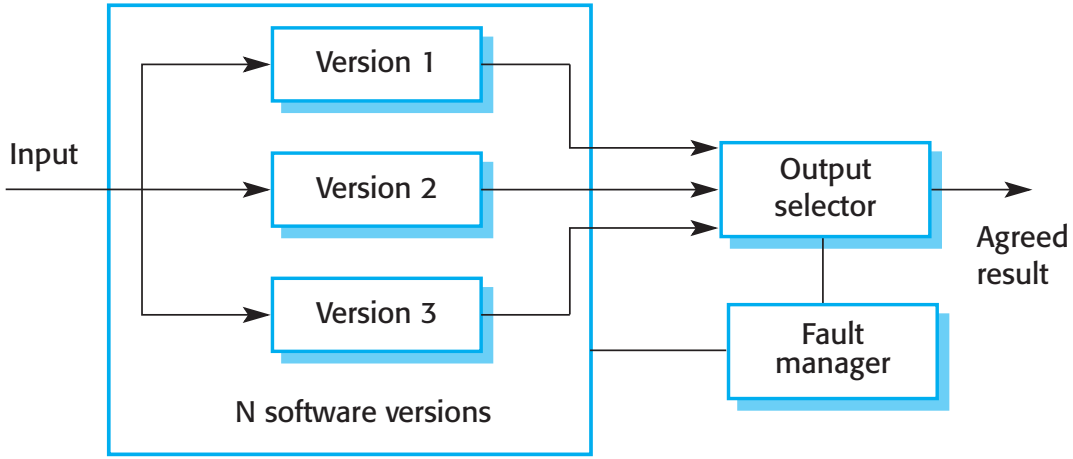
- Functional reliability requirements define system and software functions that avoid, detect or tolerate faults in the software and so ensure that these faults do not lead to system failure.
  - **Checking requirements** that identify checks to ensure that incorrect data is detected before it leads to a failure.
  - **Recovery requirements** that are geared to help the system recover after a failure has occurred.
  - **Redundancy requirements** that specify redundant features of the system to be included.
- Process requirements for reliability which specify the development process to be used may also be included.

Additional architectures for reliability

Self-monitoring architectures



N-version programming (like hardware's triple modular redundancy)



Reliability measurement

- To assess the reliability of a system, you have to collect data about its operation. The data required may include:
  - The number of system failures given a number of requests for system services irrespective of the time over which the demands are made. (POFOD – Probability of Failure on Demand)
  - The time or the number of transactions between system failures plus the total elapsed time or total number of transactions. (ROCOF – Rate of Occurrence of Failures and MTTF – Mean Time to Failure)
  - The repair or restart time after a system failure that leads to loss of service (AVAIL - Availability). Availability does not just depend on the time between failures but also on the time required to get the system back into operation.
- We can test software for reliability in addition to testing for defects. This requires input data that differs from that used in defect tests.

Reliability measurement problems

- Operational profile uncertainty
  - The operational profile may not be an accurate reflection of the real use of the system.
- High costs of test data generation
  - Costs can be very high if the test data for the system cannot be generated automatically.
- Statistical uncertainty
  - You need a statistically significant number of failures to compute the reliability but highly reliable systems will rarely fail.
- Recognizing failure
  - It is not always obvious when a failure has occurred as there may be conflicting interpretations of a specification.

Classroom Activity – within your EMSS project teams

Activity:

Identify the measurement and dependability requirements of the EMSS associated with reliability and availability.

Submission to complete the Canvas Assignment EMSS Week 10:

Submit those requirements and your plans for fulfilling them.

Quiz 4 results

High score: 97% Low score: 63% Average: 83%

Most missed questions:

- Effort represents BUDGET, not time!
- Risk exposure and leverage
  - 50% chance of needing $10,000; spend $2,000;
  - New: 25% chance of needing $10,000
- The Unadjusted Actor Weight (UAW)
  - 7 use cases where consumer is the only actor
  - 4 use cases where system manager is the only actor
  - 2 use cases where the consumer and an external credit card system (API) are the actors.